intel

Embedded Controller Handbook Volume II 16-Bit

1988

intel

Embedded Controller Handbook

Volume II 16-Bit

Order Number: 210918-006

LITERATURE

To order Intel literature write or call:

Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

Toll Free Number:
(800) 548-4725*

Use the order blank on the facing page or call our **Toll Free** Number listed above to order literature. Remember to add your local sales tax and a 10% postage charge for U.S. and Canada customers, 20% for customers outside the U.S. Prices are subject to change.

1988 HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

NAME	ORDER NUMBER	**PRICE IN U.S. DOLLARS
COMPLETE SET OF 8 HANDBOOKS Save $50.00 off the retail price of $175.00	**231003**	**$125.00**
AUTOMOTIVE HANDBOOK (Not included in handbook Set)	231792	**$20.00**
COMPONENTS QUALITY/RELIABILITY HANDBOOK (Available in July)	210997	**$20.00**
EMBEDDED CONTROLLER HANDBOOK (2 Volume Set)	210918	**$23.00**
MEMORY COMPONENTS HANDBOOK	210830	**$18.00**
MICROCOMMUNICATIONS HANDBOOK	231658	**$22.00**
MICROPROCESSOR AND PERIPHERAL HANDBOOK (2 Volume Set)	230843	**$25.00**
MILITARY HANDBOOK (Not included in handbook Set)	210461	**$18.00**
OEM BOARDS AND SYSTEMS HANDBOOK	280407	**$18.00**
PROGRAMMABLE LOGIC HANDBOOK	296083	**$18.00**
SYSTEMS QUALITY/RELIABILITY HANDBOOK	231762	**$20.00**
PRODUCT GUIDE Overview of Intel's complete product lines	210846	N/C
DEVELOPMENT TOOLS CATALOG	280199	N/C
INTEL PACKAGING OUTLINES AND DIMENSIONS Packaging types, number of leads, etc.	231369	N/C
LITERATURE PRICE LIST List of Intel Literature	210620	N/C

*Good in the U.S. and Canada

**These prices are for the U.S. and Canada only. In Europe and other international locations, please contact your local Intel Sales Office or Distributor for literature prices.

About Our Cover:

A piano keyboard is deceptively simple. Essentially an array of on-off switches, with the application of human intelligence it can produce an infinite variety of sound. Embedded controllers provide the "hidden intelligence" found in thousands of products we use everyday.

Intel Literature **update** Service

Get Intel's Latest Technical Literature, Automatically!

Exclusive, Intel Literature Update Service

Take advantage of Intel's year-long, low cost Literature Update Service and you will receive your first package of information followed by automatic quarterly updates on all the latest product and service news from Intel.

Choose one or all five product categories update

Each product category update listed below covers in depth, all the latest Handbooks, Data Sheets, Application Notes, Reliability Reports, Errata Reports, Article Reprints, Promotional Offers, Brochures, Benchmark Reports, Technical Papers and much more. . .

1. Microprocessors

Product line handbooks on Microprocessors, Embedded Controllers and Component Quality/Reliability, *Plus,* the Product Guide, Literature Guide, Packaging Information and 3 quarterly updates. $70.00 Order Number: **555110**

2. Peripherals

Product line handbooks on Peripherals, Microcommunications, Embedded Controllers, and Component Quality/Reliability, *Plus,* the Product Guide, Literature Guide, Packaging Information and 3 quarterly updates. $50.00 Order Number: **555111**

3. Memories

Product line handbooks on Memory Components, Programmable Logic and Components Quality/Reliability, *Plus,* the Product Guide, Literature Guide, Packaging Information and 3 quarterly updates. $50.00 Order Number: **555112**

4. OEM Boards and Systems

Product line handbooks on OEM Boards & Systems, Systems Quality/Reliability, *Plus,* the Product Guide, Literature Guide, Packaging Information and 3 quarterly updates. $50.00 Order Number: **555113**

5. Software

Product line handbooks on Systems Quality/Reliability, Development Tools Catalog, *Plus,* the Product Guide, Literature Guide, Packaging Information and 3 quarterly updates. $35.00 Order Number: **555114**

To subscribe, rush the Literature Order Form in this handbook, or call today, toll free (800) 548-4725.*

Subscribe by March 31, 1988 and receive a valuable free gift.

The charge for this service covers our printing, postage and handling cost only.

Please note: Product manuals are not included in this offer.

Customers outside the U.S. and Canada should order directly from the U.S.

Offer expires 12/31/88.

*Good in the U.S. and Canada.

intel

intel

LITERATURE SALES ORDER FORM

NAME: ____________________

COMPANY: ____________________

ADDRESS: ____________________

CITY: ____________________ STATE: __________ ZIP: __________

COUNTRY: ____________________

PHONE NO.: (______) ____________________

ORDER NO.	TITLE	QTY.	PRICE	TOTAL
□□□□□□	____________	______ ×	______ =	______
□□□□□□	____________	______ ×	______ =	______
□□□□□□	____________	______ ×	______ =	______
□□□□□□	____________	______ ×	______ =	______
□□□□□□	____________	______ ×	______ =	______
□□□□□□	____________	______ ×	______ =	______
□□□□□□	____________	______ ×	______ =	______
□□□□□□	____________	______ ×	______ =	______
□□□□□□	____________	______ ×	______ =	______
□□□□□□	____________	______ ×	______ =	______

Subtotal __________

Must Add Your
Local Sales Tax __________

Must add appropriate postage to subtotal
(10% U.S. and Canada, 20% all other) → Postage __________

Total __________

Pay by Visa, MasterCard, American Express, Check, Money Order, or company purchase order payable to Intel Literature Sales. Allow 2-4 weeks for delivery.

□ Visa □ MasterCard □ American Express Expiration Date __________

Account No. ____________________

Signature: ____________________

Mail To: Intel Literature Sales
P.O. Box 58130
Santa Clara, CA
95052-8130

International Customers outside the U.S. and Canada should contact their local Intel Sales Office or Distributor listed in the back of most Intel literature.

European Literature Order Form in back of book.

Call Toll Free: (800) 548-4725 for phone orders

Prices good until 12/31/88.

Source HB

intel®

Intel the Microcomputer Company:

When Intel invented the microprocessor in 1971, it created the era of microcomputers. Whether used as microcontrollers in automobiles or microwave ovens, or as personal computers or supercomputers, Intel's microcomputers have always offered leading-edge technology. In the second half of the 1980s, Intel architectures have held at least a 75% market share of microprocessors at 16 bits and above. Intel continues to strive for the highest standards in memory, microcomputer components, modules, and systems to give its customers the best possible competitive advantages.

EMBEDDED CONTROLLER HANDBOOK

1988

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

> Above, BITBUS, COMMputer, CREDIT, Data Pipeline, FASTPATH, GENIUS, i, î, ICE, iCEL, iCS, iDBP, iDIS, I^{2}ICE, iLBX, i$_m$, iMDDX, iMMX, Inboard, Insite, Intel, int$_e$l, int$_e$lBOS, Intel Certified, Intelevision, int$_e$ligent Identifier, int$_e$ligent Programming, Intellec, Intellink, iOSP, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MAP-NET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, PC-BUBBLE, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, SugarCube, SupportNET, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix, 4-SITE.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Distribution
Mail Stop SC6-59
3065 Bowers Avenue
Santa Clara, CA 95051

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

Chapter 20

Chapter 21

DATA SHEETS

APPLICATION NOTES

DEVELOPMENT SUPPORT TOOLS

80186 FAMILY

Chapter 22

DATA SHEETS

APPLICATION NOTES

DEVELOPMENT SUPPORT TOOLS

Alphanumeric Index

Alphanumeric Index (Continued)

CUSTOMER SUPPORT

CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, and consulting services. For more information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It includes factory repair services and worldwide field service offices providing hardware repair services, software support services, customer training classes, and consulting services.

HARDWARE SUPPORT SERVICES

Intel is committed to providing an international service support package through a wide variety of service offerings available from Intel Hardware Support.

SOFTWARE SUPPORT SERVICES

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and COMMENTS Magazine). Basic support includes updates and the subscription service. Contracts are sold in environments which represent product groupings (i.e., iRMX environment).

CONSULTING SERVICES

Intel provides field systems engineering services for any phase of your development or support effort. You can use our systems engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training, and customizing or tailoring an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, bitbus and LAN applications.

MCS®-96 Architectural Overview

17

MCS®-96 ARCHITECTURAL OVERVIEW

There are two groups of parts within the MCS®-96 family: the standard 8X9X parts and the 8X9XBH parts. There are several enhancements on the 8X9XBH parts that are not on the 8X9X parts. This manual is written about the 8X9XBH parts, generically referred to as an 8096BH. Where the standard 8X9X parts differ from the 8096BH, notations will be made.

The 8096BH can be separated into several sections for the purpose of describing its operation. There is a 16-bit CPU, a programmable High Speed I/O Unit, an analog to digital converter, a serial port, and a Pulse Width Modulated (PWM) output for digital to analog conversion. In addition to these functional units, there are some sections which support overall operation of the chip such as the clock generator. The CPU and the programmable I/O make the 8096BH very different from any other microcontroller. Let us first examine the CPU.

1.0 CPU OPERATION

The major components of the CPU on the 8096BH are the Register File and the RALU. Communication with the outside world is done through either the Special Function Registers (SFRs) or the Memory Controller. The RALU (Register/Arithmetic Logic Unit) does not use an accumulator, it operates directly on the 256-byte register space made up of the Register File and the SFRs. Efficient I/O operations are possible by directly controlling the I/O through the SFRs. The main benefits of this structure are the ability to quickly change context, the absence of accumulator bottleneck, and fast throughput and I/O times.

1.1 CPU Buses

A "Control Unit" and two busses connect the Register File and RALU. Figure 1 shows the CPU with its

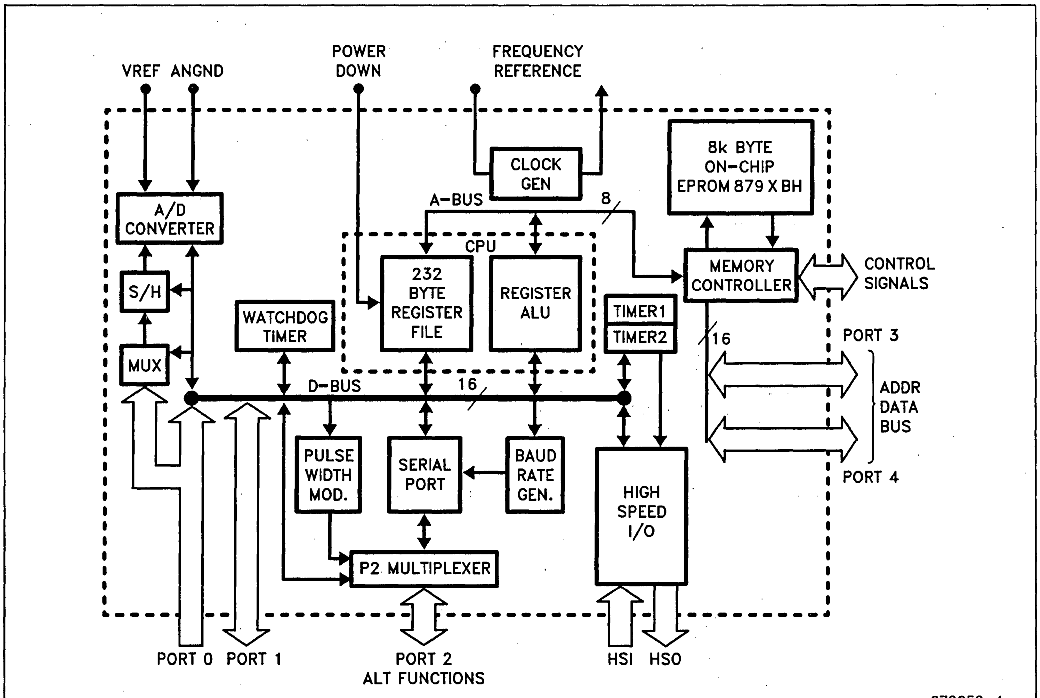


Figure 1. Block Diagram

major bus connections. The two buses are the "A-Bus" which is 8-bits wide, and the "D-Bus" which is 16-bits wide. The D-Bus transfers data only between the RALU and the Register File or Special Function Registers (SFRs). The A-Bus is used as the address bus for the above transfers or as a multiplexed address/data bus connecting to the "Memory Controller". Any accesses of either the internal ROM or external memory are done through the Memory Controller.

Within the memory controller is a slave program counter (Slave PC) which keeps track of the PC in the CPU. By having most program fetches from memory referenced to the slave PC, the processor saves time as addresses seldom have to be sent to the memory controller. If the address jumps sequence then the slave PC is loaded with a new value and processing continues. Data fetches from memory are also done through the memory controller, but the slave PC is bypassed for this operation.

1.2 CPU Register File

The Register File contains 232 bytes of RAM which can be accessed as bytes, words, or double-words. Since each of these locations can be used by the RALU, there are essentially 232 "accumulators". The first word in the Register File is reserved for use as the stack pointer so it can not be used for data when stack manipulations are taking place. Addresses for accessing the Register File and SFRs are temporarily stored in two 8-bit address registers by the CPU hardware.

1.3 RALU Control

Instructions to the RALU are taken from the A-Bus and stored temporarily in the instruction register. The Control Unit decodes the instructions and generates the correct sequence of signals to have the RALU perform the desired function. Figure 1 shows the instruction register and the control unit.

1.4 RALU

Most calculations performed by the 8096BH take place in the RALU. The RALU, shown in Figure 2, contains a 17-bit ALU, the Program Status Word (PSW), the Program Counter (PC), a loop counter, and three temporary registers. All of the registers are 16-bits or 17-bits (16+ sign extension) wide. Some of the registers have the ability to perform simple operations to offload the ALU.

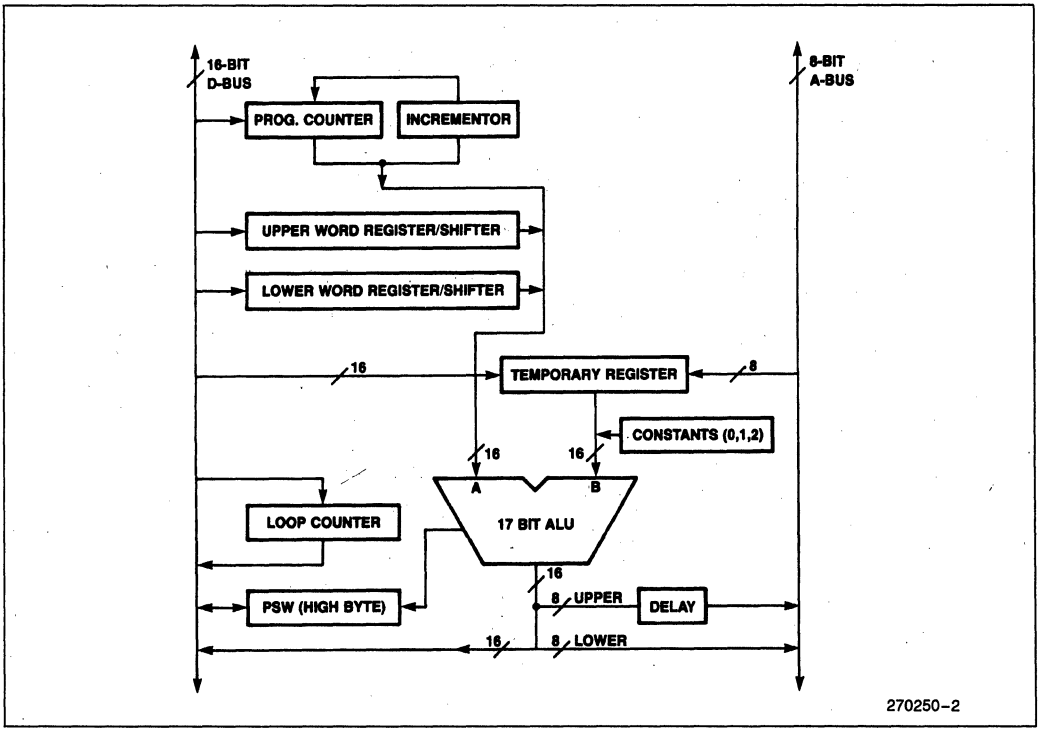


Figure 2. RALU Block Diagram

A separate incrementor is used for the PC; however, jumps must be handled through the ALU. Two of the temporary registers have their own shift logic. These registers are used for the operations which require logical shifts, including Normalize, Multiply, and Divide. The "Lower Word" register is used only when double-word quantities are being shifted, the "Upper Word" register is used whenever a shift is performed or as a temporary register for many instructions. Repetitive shifts are counted by the 5-bit "Loop Counter".

A temporary register is used to store the second operand of two operand instructions. This includes the multiplier during multiplications and the divisor during divisions. To perform subtractions, the output of this register can be complemented before being placed into the "B" input of the ALU.

The DELAY shown in Figure 2 is used to convert the 16-bit bus into an 8-bit bus. This is required as all addresses and instructions are carried on the 8-bit A-Bus. Several constants, such as 0, 1 and 2 are stored in the RALU for use in speeding up certain calculations. These come in handy when the RALU needs to make a 2's complement number or perform an increment or decrement instruction.

1.5 Internal Timing

The 8096BH requires an input clock frequency of between 6.0 MHz and 12 MHz to function. This frequency can be applied directly to XTAL1. Alternatively, since XTAL1 and XTAL2 are inputs and outputs of an inverter, it is also possible to use a crystal to generate the clock. A block diagram of the oscillator section is shown in Figure 3. Details of the circuit and suggestions for its use can be found in Section 1 of the Hardware Design chapter.

The crystal or external oscillator frequency is divided by 3 to generate the three internal timing phases as shown in Figure 4. Each of the internal phases repeat every 3 oscillator periods: 3 oscillator periods are referred to as one "state time", the basic time measurement for 8096BH operations. Most internal operations are synchronized to either Phase A, B or C, each of which have a 33% duty cycle. Phase A is represented externally by CLKOUT, a signal available on the 68-pin part. Phases B and C are not available externally. The relationships of XTAL1, CLKOUT, and Phases A, B, and C are shown in Figure 4. It should be noted that propagation delays have not been taken into account in this diagram. Details on these and other timing relationships can be found in the Hardware Design chapter.

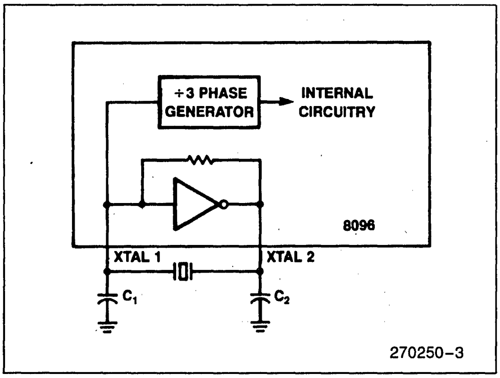


Figure 3. Block Diagram of Oscillator

The RESET line can be used to start the 8096BH at an exact time to provide for synchronization of test equipment and multiple chip systems. Use of this feature is fully explained under RESET, Section 13.

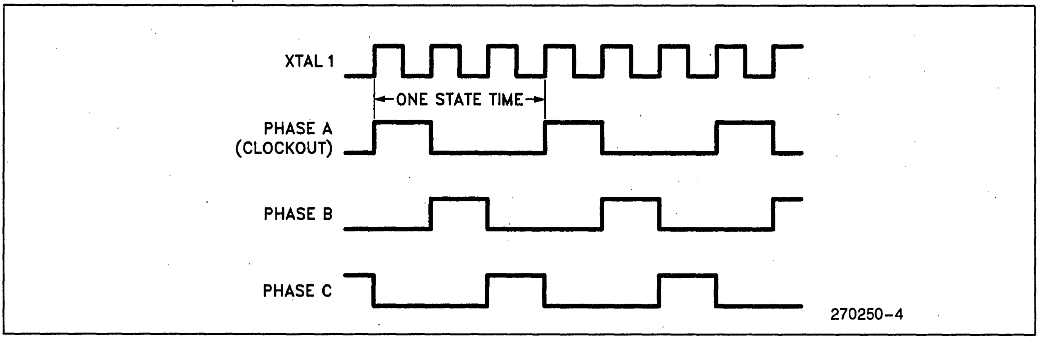


Figure 4. Internal Timings Relative to XTAL 1

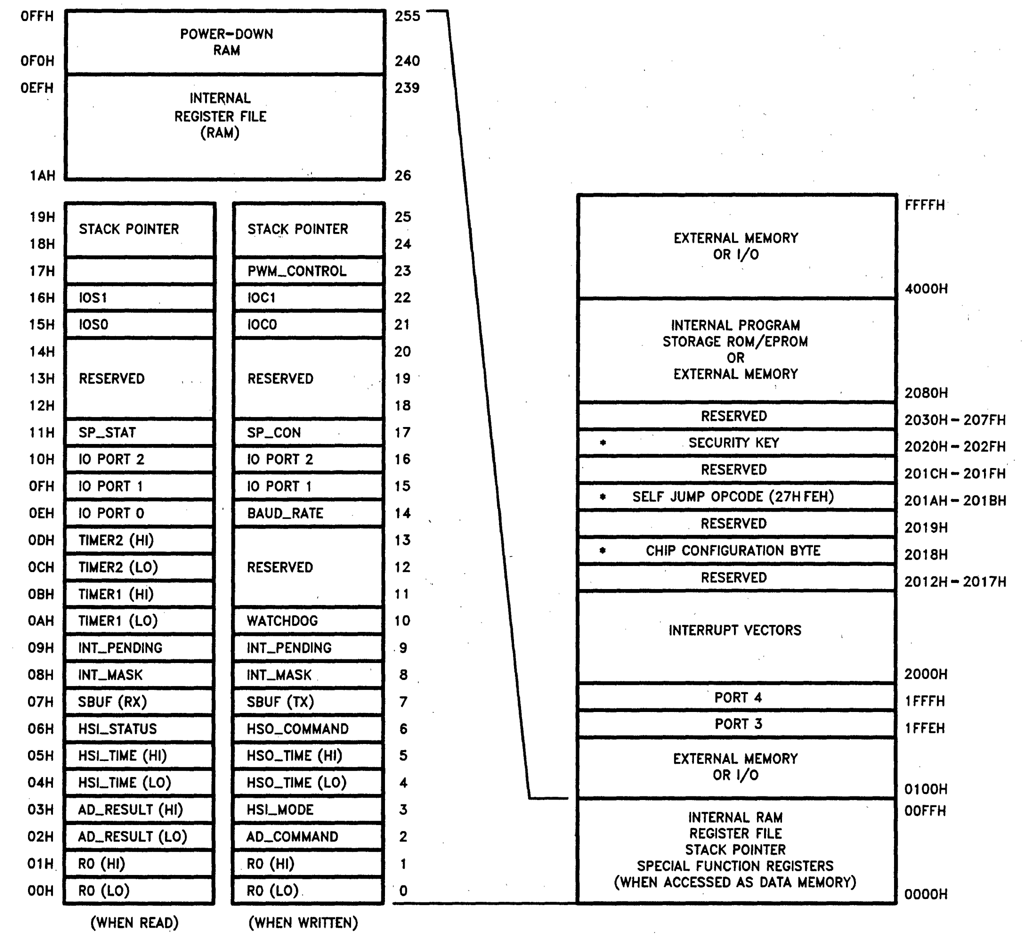
2.0 MEMORY SPACE

The addressable memory space on the 8096BH consists of 64K bytes, most of which is available to the user for program or data memory. Locations which have special purposes are 0000H through 00FFH and 1FFEH through 2080H. All other locations can be used for either program or data storage or for memory mapped peripherals. A memory map is shown in Figure 5.

2.1 Register File

Locations 00H through 0FFH contain the Register File and Special Function Registers, (SFRs). No code can be executed from this internal RAM section. If an attempt to execute instructions from locations 000H through 0FFH is made, the instructions will be fetched from *external* memory. This section of external memory is reserved for use by Intel development tools. Execution of a nonmaskable interrupt (NMI) will force a call to external location 0000H, therefore, the NMI instruction is also reserved for Intel development tools.

The RALU can operate on any of the 256 internal register locations. Locations 00H through 17H are used to access the SFRs. Locations 18H and 19H contain the stack pointer. These are not SFRs, and may be used as standard RAM if stack operations are not being performed. The stack pointer must be initialized by the



	(WHEN READ)	(WHEN WRITTEN)	
0FFH 0F0H	POWER-DOWN RAM	POWER-DOWN RAM	255 240
0EFH 1AH	INTERNAL REGISTER FILE (RAM)	INTERNAL REGISTER FILE (RAM)	239 26
19H 18H	STACK POINTER	STACK POINTER	25 24
17H		PWM_CONTROL	23
16H	IOS1	IOC1	22
15H	IOS0	IOC0	21
14H 13H 12H	RESERVED	RESERVED	20 19 18
11H	SP_STAT	SP_CON	17
10H	IO PORT 2	IO PORT 2	16
0FH	IO PORT 1	IO PORT 1	15
0EH	IO PORT 0	BAUD_RATE	14
0DH	TIMER2 (HI)	RESERVED	13
0CH	TIMER2 (LO)		12
0BH	TIMER1 (HI)		11
0AH	TIMER1 (LO)	WATCHDOG	10
09H	INT_PENDING	INT_PENDING	9
08H	INT_MASK	INT_MASK	8
07H	SBUF (RX)	SBUF (TX)	7
06H	HSI_STATUS	HSO_COMMAND	6
05H	HSI_TIME (HI)	HSO_TIME (HI)	5
04H	HSI_TIME (LO)	HSO_TIME (LO)	4
03H	AD_RESULT (HI)	HSI_MODE	3
02H	AD_RESULT (LO)	AD_COMMAND	2
01H	R0 (HI)	R0 (HI)	1
00H	R0 (LO)	R0 (LO)	0

EXTERNAL MEMORY OR I/O	FFFFH 4000H
INTERNAL PROGRAM STORAGE ROM/EPROM OR EXTERNAL MEMORY	2080H
RESERVED	2030H – 207FH
* SECURITY KEY	2020H – 202FH
RESERVED	201CH – 201FH
* SELF JUMP OPCODE (27H FEH)	201AH – 201BH
RESERVED	2019H
* CHIP CONFIGURATION BYTE	2018H
RESERVED	2012H – 2017H
INTERRUPT VECTORS	2000H
PORT 4	1FFFH
PORT 3	1FFEH
EXTERNAL MEMORY OR I/O	0100H
INTERNAL RAM REGISTER FILE STACK POINTER SPECIAL FUNCTION REGISTERS (WHEN ACCESSED AS DATA MEMORY)	00FFH 0000H

270250–5

NOTE:
*Registers marked by an asterisk are not present on 8X9X devices

Figure 5. Memory Map

user program and can point anywhere in the 64K memory space. The stack builds down. There are no restrictions on the use of the remaining 230 locations except that code cannot be executed from them.

2.2 Special Function Registers

All of the I/O on the 8096BH is controlled through the SFRs. Many of these registers serve two functions; one if they are read from, the other if they are written to. Figure 5 shows the locations and names of these registers. A summary of the capabilities of each of these registers is shown in Figure 6, with complete descriptions reserved for later sections.

There are several restrictions on using special function registers.

Neither the source or destination addresses of the Multiply and Divide instructions can be a writable special function register.

These registers may not be used as base or index registers for indirect or indexed instructions.

These registers can only be accessed as bytes unless otherwise specified in Figure 6. Note that some of these registers can only be accessed as words, and not as bytes.

Within the SFR space are several registers labeled "RESERVED". These registers are reserved for future expansion and test purposes. Operations should not be performed with these registers as reads from them and writes to them may produce unexpected results. For example, in some versions of the 8096 writing to location 0CH will set both timers to 0FFFXH. This may not be the case in future products, so it should not be used as a feature.

2.3 Power Down

The upper 16 RAM locations (0F0H through 0FFH) receive their power from the V_{PD} pin. If it is desired to keep the memory in these locations alive during a power down situation, one need only keep voltage on the V_{PD} pin. The current required to keep the RAM alive is approximately 1 milliamp (refer to the data sheet for the exact specification). Both V_{CC} and V_{PD} must have power applied for normal operation. If V_{PD} is not applied the power down RAM will not function properly, even if V_{CC} is applied.

To place the 8096BH into a power down mode, the $\overline{\text{RESET}}$ pin is pulled low. Two state times later the part will be in reset. This is necessary to prevent the part from writing into RAM as the power goes down. The power may now be removed from the V_{CC} pin, the V_{PD} pin must remain within specifications. The 8096BH can remain in this state for any amount of time and the 16 RAM bytes will retain their values.

To bring the 8096BH out of power down, $\overline{\text{RESET}}$ is held low while V_{CC} is applied. Two state times after the oscillator has stabilized, the $\overline{\text{RESET}}$ pin can be pulled high. *On the 8X9X devices the back-bias generator must also stabilize. This requires approximately 1 millisecond.* The 8096BH will begin to execute code at location 02080H 10 state times after $\overline{\text{RESET}}$ is pulled high. Figure 7 shows a timing diagram of the power down sequence. To ensure that the 2 state time minimum reset time (synchronous with CLKOUT) is met, it is recommended that 10 XTAL1 cycles be used. Suggestions for actual hardware connections are given in the Hardware Design Chapter. Reset is discussed in Section 13.

To determine if a reset is a return from power down or a complete cold start a "key" can be written into power-down RAM while the part is running. This key can be checked on reset to determine which type of reset has occurred. In this way the validity of the power-down RAM can be verified. The length of this key determines the probability that this procedure will work, however, there is always a statistical chance that the RAM will power up with a replica of the key.

Under most circumstances, the power-fail indicator which is used to initiate a power-down condition must come from the unfiltered, unregulated section of the power supply. The power supply must have sufficient storage capacity to operate the 8096BH until it has completed its reset operation.

Register	Description	Section
R0	Zero Register — Always reads as a zero, useful for a base when indexing and as a constant for calculations and compares.	3
AD_RESULT	A/D Result Hi/Low — Low and high order Results of the A/D converter (byte read only)	8
AD_COMMAND	A/D Command Register — Controls the A/D	8
HSI_MODE	HSI Mode Register — Sets the mode of the High Speed Input unit.	6
HSI_TIME	HSI Time Hi/Lo — Contains the time at which the High Speed Input unit was triggered. (word read only)	6
HSO_TIME	HSO Time Hi/Lo — Sets the time or count for the High Speed Output to execute the command in the Command Register. (word write only)	7
HSO_COMMAND	HSO Command Register — Determines what will happen at the time loaded into the HSO Time registers.	7
HSI_STATUS	HSI Status Registers — Indicates which HSI pins were detected at the time in the HSI Time registers and the current state of the pins.	6
SBUF (TX)	Transmit buffer for the serial port, holds contents to be outputted.	9
SBUF (RX)	Receive buffer for the serial port, holds the byte just received by the serial port.	9
INT_MASK	Interrupt Mask Register — Enables or disables the individual interrupts.	4
INT_PENDING	Interrupt Pending Register — Indicates that an interrupt signal has occurred on one of the sources and has not been serviced.	4
WATCHDOG	Watchdog Timer Register — Written to periodically to hold off automatic reset every 64K state times.	12
TIMER1	Timer 1 Hi/Lo — Timer 1 high and low bytes. (word read only)	5
TIMER2	Timer 2 Hi/Lo — Timer 2 high and low bytes. (word read only)	5
IOPORT0	Port 0 Register — Levels on pins of port 0.	10
BAUD_RATE	Register which determines the baud rate, this register is loaded sequentially.	9
IOPORT1	Port 1 Register — Used to read or write to Port 1.	10
IOPORT2	Port 2 Register — Used to read or write to Port 2.	10
SP_STAT	Serial Port Status — Indicates the status of the serial port.	9
SP_CON	Serial Port Control — Used to set the mode of the serial port.	9
IOS0	I/O Status Register 0 — Contains information on the HSO status	11
IOS1	I/O Status Register 1 — Contains information on the status of the timers and of the HSI.	11
IOC0	I/O Control Register 0 — Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.	11
IOC1	I/O Control Register 1 — Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.	11
PWM_CONTROL	Pulse Width Modulation Control Register — Sets the duration of the PWM pulse.	8

Figure 6. SFR Summary

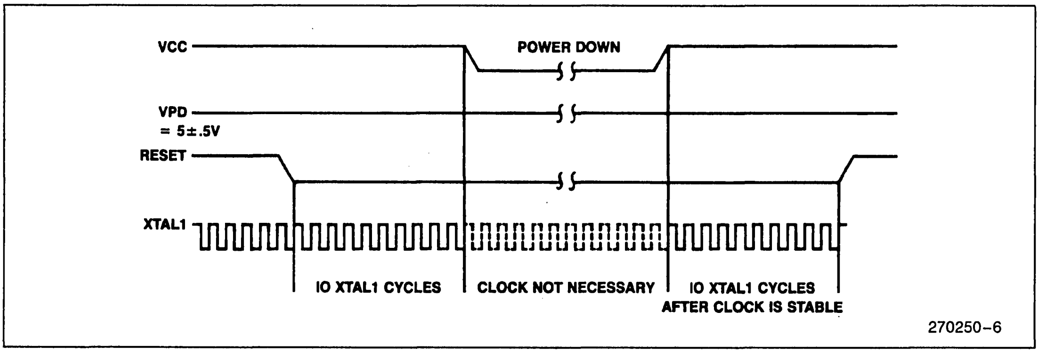


Figure 7. Power Down Timing

2.4 Reserved Memory Spaces

A listing of locations with special significance is shown in Figure 8. The locations marked "Reserved" are reserved by Intel for use in testing or future products. They must be filled with the Hex value FFH to insure compatibility with future parts.

Locations 1FFEH and 1FFFH are reserved for Ports 3 and 4 respectively. This is to allow easy reconstruction of these ports if external memory is used in the system. An example of reconstructing the I/O ports is given in section 7 of the Hardware Design chapter. If ports 3 and 4 are not going to be reconstructed, these locations can be treated as any other external memory location.

The 9 interrupt vectors are stored in locations 2000H through 2011H. The 9th vector is used by Intel development systems, as explained in Section 4.

Locations 2012H through 2017H are reserved for future use. Location 2018H is the Chip Configuration byte which will be discussed in the next section. The Jump-To-Self opcodes at locations 201AH and 201BH are provided for EPROM programming as detailed in the Hardware Design chapter. Locations 2020H through 202FH are the security key used with the ROM Lock feature which will be discussed in the next section. All unspecified addresses in locations 2000H through 207FH, including those marked Reserved, should be considered reserved for use by Intel.

Resetting the 8096BH causes instructions to be fetched starting from location 2080H. This location was chosen to allow a system to have up to 8K of RAM continuous with the register file. Further information on reset can be found in Section 13.

0000H-	0017H	Register Mapped I/O (SFRs)
0018H-	0019H	Stack Pointer
1FFEH-	1FFFH	Ports 3 and 4
2000H-	2011H	Interrupt Vectors
2012H-	2017H	Reserved
2018H		Chip Configuration Byte
2019H		Reserved
201AH-	201BH	"Jump to Self" Opcode (27H FEH)
201CH-	201FH	Reserved
2020H-	202FH	Security Key
2030H-	207FH	Reserved
2080H		Reset Location

Figure 8. Registers with Special Significance

2.5 Internal ROM and EPROM

When a ROM part is ordered, or an EPROM part is programmed, the internal memory locations 2080H through 3FFFH are user specified, as are the interrupt vectors, Chip Configuration Register and Security Key in locations 2000H through 202FH.

Instruction and data fetches from the internal ROM or EPROM occur only if the part has a ROM or EPROM, $\overline{EA}$ is tied high, and the address is between 2000H and 3FFFH. At all other times data is accessed from either the internal RAM space or external memory and instructions are fetched from external memory. The $\overline{EA}$ pin is latched on $\overline{RESET}$ rising. Information on programming EPROMs can be found in Section 10 of the Hardware Design chapter.

Do not execute code out of the last three locations of internal ROM/EPROM.

2.6 Memory Controller

The RALU talks to the memory (except for the locations in the register file and SFR space) through the memory controller which is connected to the RALU by the A-Bus and several control lines. Since the A-Bus is eight bits wide, the memory controller uses a Slave Program Counter to avoid having to always get the instruction location from the RALU. This slave PC is incremented after each fetch. When a jump or call occurs, the slave PC must be loaded from the A-Bus before instruction fetches can continue.

In addition to holding a slave PC, the memory controller contains a 3 byte queue to help speed execution. This queue is transparent to the RALU and to the user unless wait states are forced during external bus cycles. The instruction execution times shown in Section 14.8 show the normal execution times with no wait states added and the 16-bit bus selected. Reloading the slave PC and fetching the first byte of the new instruction stream takes 4 state times. This is reflected in the jump taken/not-taken times shown in the table.

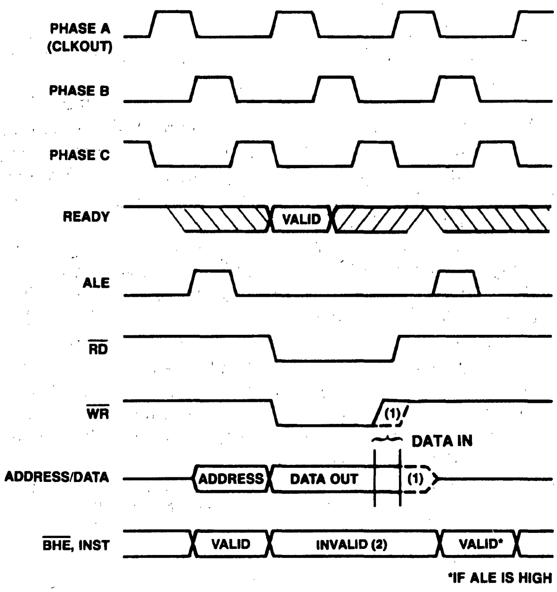
2.7 System Bus

There are several operating modes on the 8096BH. The standard bus mode uses a 16-bit multiplexed address/data bus. Other bus modes include an 8-bit mode and a mode in which the bus size can dynamically be switched between 8-bits and 16-bits. In addition, there are several options available on the type of control signals used by the bus. *8X9X devices only operate in the standard mode.*

In the standard mode, external memory is addressed through lines AD0 through AD15 which form a 16-bit multiplexed (address/data) data bus. These lines share pins with I/O Ports 3 and 4. The falling edge of the Address Latch Enable (ALE) line is used to provide a clock to a transparent latch (74LS373) in order to demultiplex the bus. A typical circuit and the required timings are shown in Section 7 of the Hardware Design chapter. Since the 8096BH's external memory can be addressed as either bytes or words, the decoding is controlled with two lines, Bus High Enable ($\overline{\text{BHE}}$) and Address/Data Line 0 (AD0). *On the 8X9X devices the $\overline{\text{BHE}}$ line must be transparently latched, just as the addresses are.*

To avoid confusion during the explanation of the memory system it is reasonable to give names to the demultiplexed address/data signals. The address signals will be called MA0 through MA15 (Memory Address), and the data signals will be called MD0 through MD15 (Memory Data).

When $\overline{\text{BHE}}$ is active (low), the memory connected to the high byte of the data bus should be selected. When MA0 is low the memory connected to the low byte of



NOTES:
1. These sections are not valid on 8X9X devices (16-bit Mode)
2. $\overline{\text{BHE}}$/INST are valid for the entire bus cycle on the 8096BH

Figure 9. External Memory Timings

the data bus should be selected. In this way accesses to a 16-bit wide memory can be to the low (even) byte only (MA0=0, $\overline{\text{BHE}}$=1), to the high (odd) byte only (MA0=1, $\overline{\text{BHE}}$=0), or to both bytes (MA0=0, $\overline{\text{BHE}}$=0). When a memory block is being used only for reads, $\overline{\text{BHE}}$ and MA0 need not be decoded.

TIMINGS

Figure 9 shows the idealized waveforms related to the following description of external memory manipulations. For exact timing specifications please refer to the latest data sheet. When an external memory fetch begins, the address latch enable (ALE) line rises, the address is put on AD0–AD15 and $\overline{\text{BHE}}$ is set to the required state. ALE then falls, the address is taken off the pins, and the $\overline{\text{RD}}$ (Read) signal goes low. When $\overline{\text{RD}}$ falls, external memory should present its data to the 8096BH.

READ

The data from the external memory must be on the bus and stable for a minimum of the specified set-up time before the rising edge of $\overline{\text{RD}}$. The rising edge of $\overline{\text{RD}}$ latches the information into the 8096BH. If the read is for data, the INST pin will be low when the address is valid, if it is for an instruction the INST pin will be high during this time. The 48-lead part does not have the INST pin. The INST pin will be low for the Chip Configuration Byte and Interrupt Vector fetches.

WRITE

Writing to external memory requires timings that are similar to those required when reading from it. The main difference is that the write ($\overline{\text{WR}}$) signal is used instead of the $\overline{\text{RD}}$ signal. The timings are the same until the falling edge of the $\overline{\text{WR}}$ line. At this point the 8096BH removes the address and places the data on the bus. When the $\overline{\text{WR}}$ line goes high the data should be latched to the external memory. In systems which can write to byte locations, the AD0 and $\overline{\text{BHE}}$ lines must be used to decode $\overline{\text{WR}}$ into WRite to Low byte ($\overline{\text{WRL}}$) and WRite to High byte ($\overline{\text{WRH}}$) signals. INST is always low during a write, as instructions cannot be written. The exact timing specifications for memory accesses can be found in the data sheet.

READY

A ready line is available on the 8096BH to extend the width of the RD and WR pulses in order to allow access of slow memories or for DMA purposes. If the READY line is low by the specified time after ALE falls, the 8096BH will hold the bus lines to their values at the falling edge of CLKOUT. When the READY line rises the bus cycle will continue with the next falling edge of CLKOUT.

Since the bus is synchronized to CLKOUT, it can be held only for an integral number of state times. If more than TYLYH nanoseconds are added the processor will act unpredictably.

There are several set-up and hold times associated with the READY signal. If these timings are not met, the part may not respond with the proper number of wait states,

For falling edges of READY, sampling is done internally on the falling edge of Phase A. Since Phase A generates CLKOUT, (after some propagation delay) the sample will be taken prior to CLKOUT falling. The timing specification for this is given as TLLYV, the time between when ALE falls and READY must be valid. If READY changes between TLLYV max and the falling edge of CLKOUT (TLLYH MIN on 48-lead devices) it would be possible to have the READY signal transitioning as it is being sampled.

This situation could cause a metastable condition which could make the device operate unpredictably.

For the rising edge of READY, sampling is done internally on the rising edge of Phase A. The rising edge logic is fully synchronized, so it is not possible to cause a metastable condition once the device is in a valid not-ready condition. To cause one wait state to occur the rising edge of READY must occur before TLLYH MAX after ALE falls. If the signal is brought up after this time two wait states may occur. If two wait states are desired, READY should be brought high within the TLLYH specification + 3 Tosc. Additional wait states can be caused by adding additional state times to the READY low time. The maximum amount of time that a device may be held not-ready is specified as TYLYH.

The 8096BH has the ability to internally limit the number of wait states to 1, 2, or 3 as determined by the value in the Chip Configuration Register, (CCR). Using the CCR for ready timing is discussed at the end of this section. If a ready limit is set, the TLLYH MAX specification is not used. *The 8X9X devices do not have internal ready control.*

OPERATING MODES

The 8096BH supports a variety of options to simplify memory systems, interfacing requirements and ready control. Bus flexibility is provided by allowing selection of bus control signal definitions and runtime selection of the external bus width. In addition, several ready control modes are available to simplify the external hardware requirements for accessing slow devices. The Chip Configuration Register (CCR) is used to store the operating mode information.

Since there is no CCR on 8X9X devices, they can only be configured in the standard mode. This is the mode the 8096BH will run in if the CCR is loaded with 0FFH.

CHIP CONFIGURATION REGISTER (CCR)

Configuration information is stored in the Chip Configuration Register (CCR). Four of the bits in the register specify the bus control mode and ready control mode. Two bits also govern the level of ROM/EPROM protection and one bit is NANDed with the BUSWIDTH pin every bus cycle to determine the bus size. The CCR bit map is shown in Figure 10. The functions associated with each bit are described in this section.

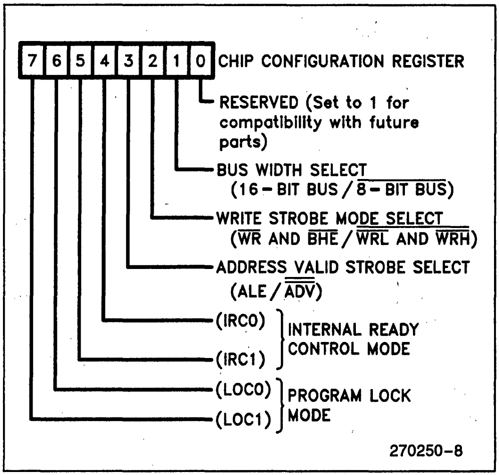

7 6 5 4 3 2 1 0
CHIP CONFIGURATION REGISTER
RESERVED (Set to 1 for compatibility with future parts)
BUS WIDTH SELECT (16-BIT BUS / $\overline{8-BIT BUS}$)
WRITE STROBE MODE SELECT ($\overline{WR}$ AND $\overline{BHE}$ / $\overline{WRL}$ AND $\overline{WRH}$)
ADDRESS VALID STROBE SELECT (ALE / $\overline{ADV}$)
(IRC0)
(IRC1)
INTERNAL READY CONTROL MODE
(LOC0)
(LOC1)
PROGRAM LOCK MODE
270250-8

Figure 10. Chip Configuration Register

The CCR is loaded on reset with the Chip Configuration Byte, located at address 2018H. The CCR register is a non-memory mapped location that can only be written to during the reset sequence; once it is loaded it cannot be changed until the next reset occurs. The 8096BH will correctly read this location in every bus mode.

If the $\overline{EA}$ pin is set to a logical 0, the access to 2018H comes from external memory. If $\overline{EA}$ is a logical 1, the access comes from internal ROM/EPROM. If $\overline{EA}$ is + 12.5V, the CCR is loaded with a byte from a separate non-memory-mapped location called PCCB (Programming CCB). The Programming mode is described in Section 10 of the Hardware Design chapter.

The CCR is not present on 8X9X devices, but the Chip Configuration Byte at location 2018H should contain the hex value 0FFH to provide future compatibility. 8X9X devices do not access location 2018H on reset.

BUS WIDTH

The 8096BH external bus width can be run-time configured to operate as a standard 16-bit multiplexed address/data bus, or as an 8051 style 16-bit address/8-bit data bus.

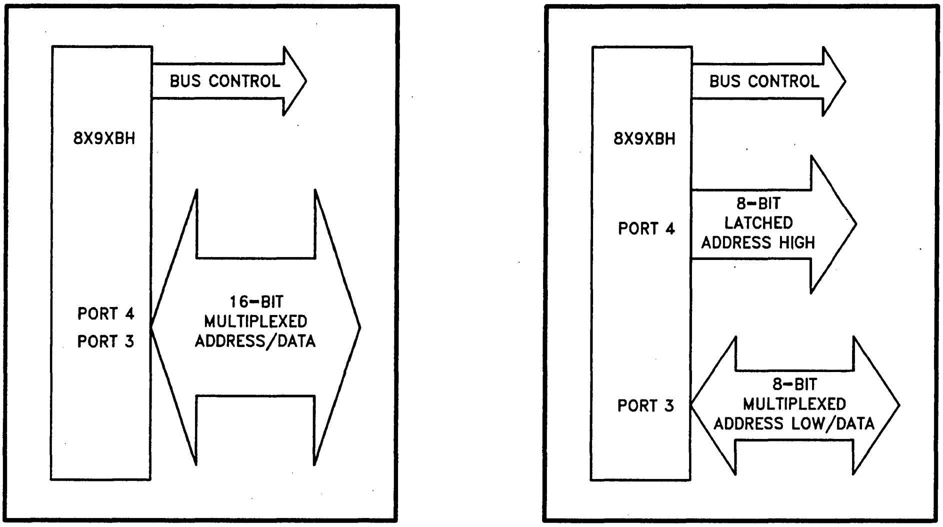
During 16-bit bus cycles, Ports 3 and 4 contain the address multiplexed with data using ALE to latch the address. In 8-bit bus cycles, Port 3 is multiplexed address/data while Port 4 is address bits 8 through 15. The address bits on Port 4 are valid throughout an 8-bit bus cycle. Figure 11 shows the two options.

The bus width can be changed each bus cycle and is controlled using bit 1 of the CCR with the BUSWIDTH pin. If either CCR.1 or BUSWIDTH is a 0, external accesses will be over a 16-bit address/8-bit data bus. If both CCR.1 and BUSWIDTH are 1s, external accesses will be over a 16-bit address/16-bit data bus. Internal accesses are always 16-bits wide.

The bus width can be changed every external bus cycle if a 1 was loaded into CCR bit 1 at reset. If this is the case, changing the value of the BUSWIDTH pin at runtime will dynamically select the bus width. For example, the user could feed the INST line into the BUSWIDTH pin, thus causing instruction accesses to be word wide from EPROMs while data accesses are byte wide to and from RAMs. A second example would be to place an inverted version of Address bit 15 on the BUSWIDTH pin. This would make half of external memory word wide, while half is byte wide.

Since BUSWIDTH is sampled after address decoding has had time to occur, even more complex memory maps could be constructed. See the timing specifications for an exact description of BUSWIDTH timings. The bus width will be determined by bit 1 of the CCR alone on 48-pin parts since they do not have a BUSWIDTH pin.

When using an 8-bit bus, some performance degradation is to be expected. On the 8096BH, instruction execution times with an 8-bit bus will slow down if any of three conditions occur. First, word writes to external memory will cause the executing instruction to take two extra state times to complete. Second, word reads from external memory will cause a one state time extension of instruction execution time. Finally, if the prefetch queue is empty when an instruction fetch is requested, instruction execution is lengthened by one state time for each byte that must be externally acquired (worst case is the number of bytes in the instruction minus one.)



270250-9

16-Bit Bus

270250-10

8-Bit Bus

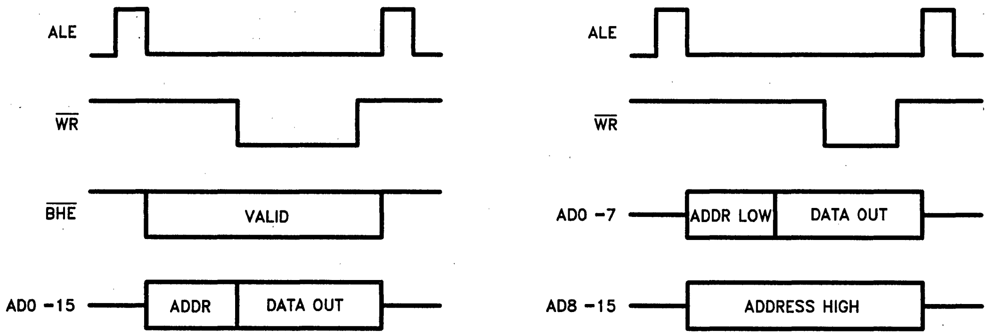
Figure 11. Bus Width Options

BUS CONTROL

Using the CCR, the 8096BH can be made to provide bus control signals of several types. Three control lines have dual functions designed to reduce external hardware. Bits 2 and 3 of the CCR specify the functions performed by these control lines. Figures 12–15 show the signals which can be modified by changing bits in the CCR, all other lines will operate as shown in Figure 9.

Standard Bus Control

If CCR bits 2 and 3 are 1s, then the standard 8096BH control signals $\overline{\text{WR}}$, $\overline{\text{BHE}}$ and ALE are provided (Figure 12). $\overline{\text{WR}}$ will come out for every write. $\overline{\text{BHE}}$ will be valid throughout the bus cycle and can be combined with $\overline{\text{WR}}$ and address line 0 to form $\overline{\text{WRL}}$ and $\overline{\text{WRH}}$. ALE will rise as the address starts to come out, and will fall to provide the signal to externally latch the address. *The control signals on 8X9X devices are similar, but not identical to those shown here. Figure 9 shows the 8X9X timings.*



270250-11

16-Bit Bus Cycle

270250-12

8-Bit Bus Cycle

Figure 12. Standard Bus Control

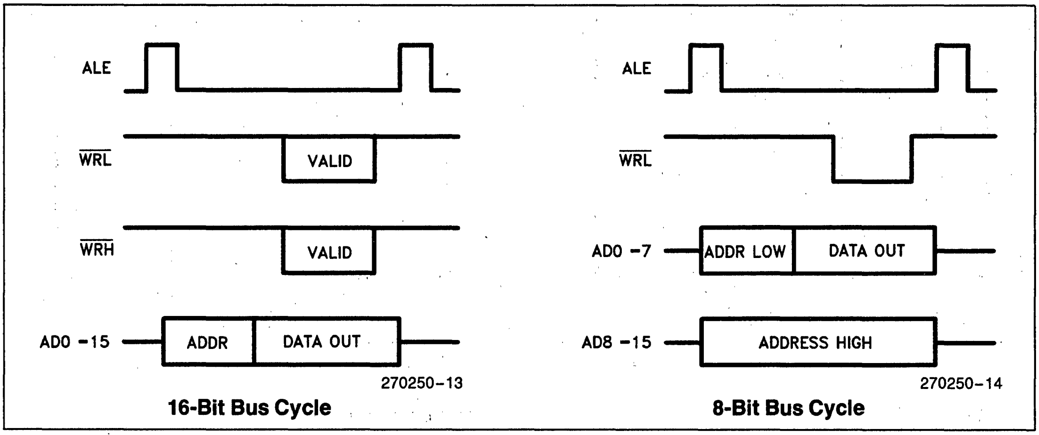
Write Strobe Mode

The Write Strobe Mode eliminates the necessity to externally decode for odd or even byte writes. If CCR bit 2 is a 0, and the bus is in a 16-bit cycle, $\overline{\text{WRL}}$ and $\overline{\text{WRH}}$ signals are provided in place of $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ (Figure 13). $\overline{\text{WRL}}$ will go low for all byte writes to an even address and all word writes. $\overline{\text{WRH}}$ will go low for all byte writes to an odd address and all word writes.

$\overline{\text{WRL}}$ is provided for all 8-bit bus write cycles.

Address Valid Strobe Mode

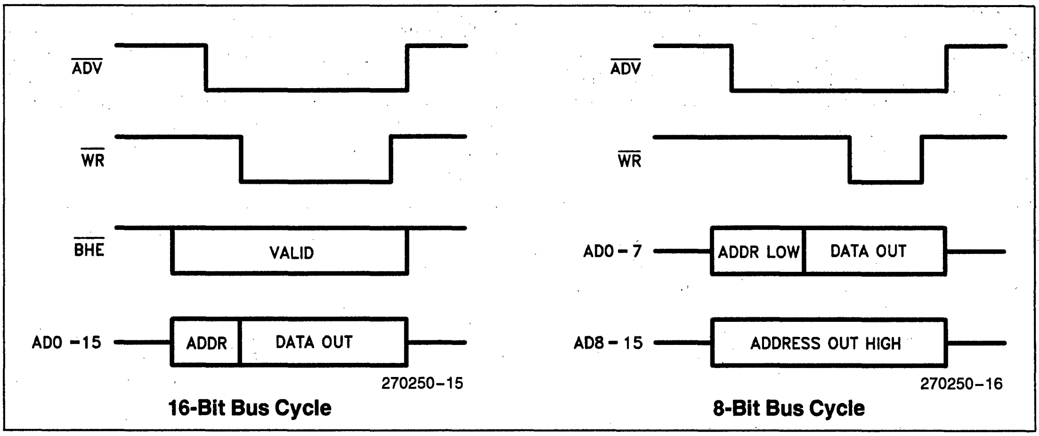
If CCR bit 3 is a 0, then an Address Valid strobe is provided in the place of ALE (Figure 14). When the address valid mode is selected, $\overline{\text{ADV}}$ will go low after an external address is set up. It will stay low until the end of the bus cycle, where it will go inactive high. This can be used to provide a chip select for external memory.



16-Bit Bus Cycle

8-Bit Bus Cycle

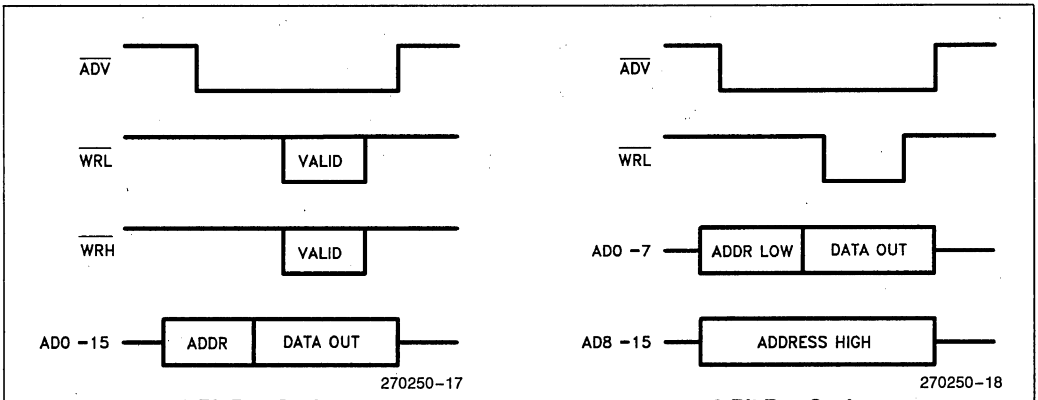
Figure 13. Write Strobe Mode



16-Bit Bus Cycle

8-Bit Bus Cycle

Figure 14. Address Valid Strobe Mode



16-Bit Bus Cycle

8-Bit Bus Cycle

Figure 15. Write Strobe with Address Valid Strobe

Address Valid with Write Strobe

If both CCR bits 2 and 3 are 0s, both the Address Valid strobe and the Write Strobes will be provided for bus control. Figure 15 shows these signals.

READY CONTROL

To simplify ready control, four modes of internal ready control logic have been provided. The modes are chosen by properly configuring bits 4 and 5 of the CCR.

The internal ready control logic can be used to limit the number of wait states that slow devices can insert into the bus cycle. When the READY pin is pulled low, wait states will be inserted into the bus cycle until the READY pin goes high, or the number of wait states equals the number specified by CCR bits 4 and 5, whichever comes first. Table 1 shows the number of wait states that can be selected. Internal Ready control can be disabled by loading 11 into bits 4 and 5 of the CCR.

Table 1. Internal Ready Control

IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

This feature provides for simple ready control. For example, every slow memory chip select line could be ORed together and be connected to the READY pin with CCR bits 4 and 5 programmed to give the desired number of wait states to the slow devices.

ROM/EPROM LOCK

Four modes of program memory lock are available on the 839XBH and 879XBH parts. CCR bits 6 and 7 (LOC0, LOC1) select whether internal program memory can be read (or written in EPROM parts) by a program executing from external memory. The modes are shown in Table 2. Internal ROM/EPROM addresses 2020H through 3FFFH are protected from reads while 2000H through 3FFFH are protected from writes, as set by the CCR.

Table 2. Program Lock Modes

LOC1	LOC0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

Only code executing from internal memory can read protected internal memory, while a write protected memory can not be written to, even from internal execution. As a result of 8096BH prefetching of instructions, however, accesses to protected memory are not allowed for instructions located above 3FFAH. This is because the lock protection mechanism is gated off of the Memory Controller's slave program counter and not the CPU program counter. If the bus controller receives a request to perform a read of protected memory, the read sequence occurs with indeterminate data being returned to the CPU. Note that the interrupt vectors and the CCR are not protected.

To provide verification and testing when the program lock feature is enabled, the 839XBH and 879XBH verify the security key before programming or test modes are allowed to read from protected memory. Before protected memory can be read, the chip reads external memory locations 4020H through 402FH and compares the values found to the internal security key located from 2020H through 202FH. Only when the values exactly match will accesses to protected memory be allowed. The details of ROM/EPROM accessing are discussed in Section 10 of the Hardware Design chapter.

3.0 SOFTWARE OVERVIEW

This section provides information on writing programs to execute in the 8096BH. Additional information can be found in the following documents:

MCS®-96 MACRO ASSEMBLER USER'S GUIDE
Order Number 122048 (Intel Systems)
Order Number 122351 (DOS Systems)

MCS®-96 UTILITIES USER'S GUIDE
Order Number 122049 (Intel Systems)
Order Number 122356 (DOS Systems)

PL/M-96 USER'S GUIDE
Order Number 122134 (Intel Systems)
Order Number 122361 (DOS Systems)

Throughout this section, short sections of code are used to illustrate the operation of the device. For these sections it has been assumed that a set of temporary registers have been predeclared. The names of these registers have been chosen as follows:

AX, BX, CX, and DX are 16-bit registers.

AL is the low byte of AX, AH is the high byte.

BL is the low byte of BX

CL is the low byte of CX

DL is the low byte of DX

These are the same as the names for the general data registers used in the 8086BH. It is important to note, however, that in the 8096, these are not dedicated registers but merely the symbolic names assigned by the programmer to an eight byte region within the onboard register file.

3.1 Operand Types

The MCS®-96 architecture provides support for a variety of data types which are likely to be useful in a control application. In the discussion of these operand types that follows, the names adopted by the PLM-96 programming language will be used where appropriate. To avoid confusion, the name of an operand type will be capitalized. A "BYTE" is an unsigned eight bit variable; a "byte" is an eight bit unit of data of any type.

BYTES

BYTES are unsigned 8-bit variables which can take on the values between 0 and 255. Arithmetic and relational operators can be applied to BYTE operands but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTES are applied bitwise. Bits within BYTES are labeled from 0 to 7, with 0 being the least significant bit. There are no alignment restrictions for BYTES, so they may be placed anywhere in the MCS-96 address space.

WORDS

WORDS are unsigned 16-bit variables which can take on the values between 0 and 65535. Arithmetic and relational operators can be applied to WORD operands but the result must be interpreted modulo 65536. Logical operations on WORDS are applied bitwise. Bits within words are labeled from 0 to 15 with 0 being the least significant bit. WORDS must be aligned at even byte boundaries in the MCS-96 address space. The least significant byte of the WORD is in the even byte address and the most significant byte is in the next higher (odd) address. The address of a word is the address of its least significant byte. Word operations to odd addresses are not guaranteed to operate in a consistent manner.

SHORT-INTEGERS

SHORT-INTEGERS are 8-bit signed variables which can take on the values between −128 and +127. Arithmetic operations which generate results outside of the range of a SHORT-INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT-INTEGERS so they may be placed anywhere in the MCS-96 address space.

INTEGERS

INTEGERS are 16-bit signed variables which can take on the values between −32,768 and 32,767. Arithmetic operations which generate results outside of the range of an INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on WORD variables. INTEGERS conform to the same alignment and addressing rules as do WORDS.

BITS

BITS are single-bit operands which can take on the Boolean values of true and false. In addition to the normal support for bits as components of BYTE and WORD operands, the 8096 provides for the direct testing of any bit in the internal register file. The MCS-96 architecture requires that bits be addressed as components of BYTES or WORDS, it does not support the direct addressing of bits that can occur in the MCS-51 architecture.

DOUBLE-WORDS

DOUBLE-WORDS are unsigned 32-bit variables which can take on the values between 0 and 4,294,967,295. The MCS-96 architecture provides direct support for this operand type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply. For these operations a DOUBLE-WORD variable must reside in the on-board register file of the 8096 and be aligned at an address which is evenly divisible by 4. A DOUBLE-WORD operand is addressed by the address of its least significant byte. DOUBLE-WORD operations which are not directly supported can be easily implemented with two WORD operations. For consistency with Intel provided software the user should adopt the conventions for addressing DOUBLE-WORD operands which are discussed in Section 3.5.

LONG-INTEGERS

LONG-INTEGERS are 32-bit signed variables which can take on the values between −2,147,483,648 and 2,147,483,647. The MCS-96 architecture provides direct support for this data type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply.

LONG-INTEGERS can also be normalized. For these operations a LONG-INTEGER variable must reside in the onboard register file of the 8096 and be aligned at an address which is evenly divisible by 4. A LONG-INTEGER is addressed by the address of its least significant byte.

LONG-INTEGER operations which are not directly supported can be easily implemented with two INTEGER operations. For consistency with Intel provided software, the user should adopt the conventions for addressing LONG operands which are discussed in Section 3.5.

3.2 Operand Addressing

Operands are accessed within the address space of the 8096 with one of six basic addressing modes. Some of the details of how these addressing modes work are hidden by the assembly language. If the programmer is to take full advantage of the architecture, it is important that these details be understood. This section will describe the addressing modes as they are handled by the hardware. At the end of this section the addressing modes will be described as they are seen through the assembly language. The six basic address modes which will be described are termed register-direct, indirect, indirect with auto-increment, immediate, short-indexed, and long-indexed. Several other useful addressing operations can be achieved by combining these basic addressing modes with specific registers such as the ZERO register or the stack pointer.

REGISTER-DIRECT REFERENCES

The register-direct mode is used to directly access a register from the 256 byte on-board register file. The register is selected by an 8-bit field within the instruction and register address and must conform to the alignment rules for the operand type. Depending on the instruction, up to three registers can take part in the calculation.

Examples

```
ADD    AX,BX,CX      ; AX:=BX+CX
MUL    AX,BX         ; AX:=AX*BX
INCB   CL            ; CL:=CL+1
```

INDIRECT REFERENCES

The indirect mode is used to access an operand by placing its address in a WORD variable in the register file. The calculated address must conform to the alignment rules for the operand type. Note that the indirect address can refer to an operand anywhere within the address space of the 8096, including the register file. The register which contains the indirect address is selected by an eight bit field within the instruction. An instruction can contain only one indirect reference and the remaining operands of the instruction (if any) must be register-direct references.

Examples

```
LD     AX,[AX]      ; AX:=MEM_WORD(AX)
ADDB   AL,BL,[CX]   ; AL:=BL+MEM_BYTE(CX)
POP    [AX]         ; MEM_WORD(AX):=MEM_WORD(SP); SP:=SP+2
```

INDIRECT WITH AUTO-INCREMENT REFERENCES

This addressing mode is the same as the indirect mode except that the WORD variable which contains the indirect address is incremented *after* it is used to address the operand. If the instruction operates on BYTES or SHORT-INTEGERS the indirect address variable will be incremented by one, if the instruction operates on WORDS or INTEGERS the indirect address variable will be incremented by two.

Examples

```
LD    AX,[BX]+     ; AX:=MEM_WORD(BX) ; BX:=BX+2
ADDB  AL,BL,[CX]+  ; AL:=BL+MEM_BYTE(CX) ; CX:=CX+1
PUSH  [AX]+        ; SP:=SP-2;
                   ;   MEM_WORD(SP):=MEM_WORD(AX)
                   ;   AX:=AX+2
```

IMMEDIATE REFERENCES

This addressing mode allows an operand to be taken directly from a field in the instruction. For operations on BYTE or SHORT-INTEGER operands this field is eight bits wide, for operations on WORD or INTEGER operands the field is 16 bits wide. An instruction can contain only one immediate reference and the remaining operand(s) must be register-direct references.

Examples

```
ADD  AX,#340  ; AX:=AX+340
PUSH #1234H   ; SP:=SP-2; MEM_WORD(SP):=1234H
DIVB AX,#10   ; AL:=AX/10; AH:=AX MOD 10
```

SHORT-INDEXED REFERENCES

In this addressing mode an eight bit field in the instruction selects a WORD variable in the register file which is assumed to contain an address. A second eight bit field in the instruction stream is sign-extended and summed with the WORD variable to form the address of the operand which will take part in the calculation. Since the eight bit field is sign-extended, the effective address can be up to 128 bytes before the address in the WORD variable and up to 127 bytes after it. An instruction can contain only one short-indexed reference and the remaining operand(s) must be register-direct references.

Examples

```
LD    AX,12[BX]     ; AX:=MEM_WORD(BX+12)
MULB  AX,BL,3[CX]   ; AX:=BL*MEM_BYTE(CX+3)
```

LONG-INDEXED REFERENCES

This addressing mode is like the short-indexed mode except that a *16-bit* field is taken from the instruction and added to the WORD variable to form the address of the operand. No sign extension is necessary. An instruction can contain only one long-indexed reference and the remaining operand(s) must be register-direct references.

Examples

```
AND  AX,BX,TABLE[CX]     ; AX:=BX AND MEM_WORD(TABLE+CX)
ST   AX,TABLE[BX]        ; MEM_WORD(TABLE+BX):=AX
ADDB AL,BL,LOOKUP[CX]    ; AL:=BL+MEM_BYTE(LOOKUP+CX)
```

ZERO REGISTER ADDRESSING

The first two bytes in the register file are fixed at zero by the 8096 hardware. In addition to providing a fixed source of the constant zero for calculations and comparisons, this register can be used as the WORD variable in a long-indexed reference. This combination of register selection and address mode allows any location in memory to be addressed directly.

Examples

```
ADD  AX,1234[0]   ; AX:=AX+MEM_WORD(1234)
POP  5678[0]      ; MEM_WORD(5678):=MEM_WORD(SP)
                  ;  SP:=SP+2
```

STACK POINTER REGISTER ADDRESSING

The system stack pointer in the 8096 can be accessed as register 18H of the internal register file. In addition to providing for convenient manipulation of the stack pointer, this also facilitates the accessing of operands in the stack. The top of the stack, for example, can be accessed by using the stack pointer as the WORD variable in an indirect reference. In a similar fashion, the stack pointer can be used in the short-indexed mode to access data within the stack.

Examples

```
PUSH  [SP]       ; DUPLICATE TOP_OF_STACK
LD    AX,2[SP]   ; AX:=NEXT_TO_TOP
```

ASSEMBLY LANGUAGE ADDRESSING MODES

The 8096 assembly language simplifies the choice of addressing modes to be used in several respects:

Direct Addressing. The assembly language will choose between register-direct addressing and long-indexed with the ZERO register depending on where the operand is in memory. The user can simply refer to an operand by its symbolic name; if the operand is in the register file, a register-direct reference will be used, if the operand is elsewhere in memory, a long-indexed reference will be generated.

Indexed Addressing. The assembly language will choose between short and long indexing depending on the value of the index expression. If the value can be expressed in eight bits then short indexing will be used, if it cannot be expressed in eight bits then long indexing will be used.

The use of these features of the assembly language simplifies the programming task and should be used wherever possible.

3.3 Program Status Word

The program status word (PSW) is a collection of Boolean flags which retain information concerning the state of the user's program. The format of the PSW is shown in Figure 16. The information in the PSW can be broken down into two basic categories; interrupt control and condition flags. The PSW can be saved in the system stack with a single operation (PUSHF) and restored in a like manner (POPF).

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	<Interrupt Mask Reg>							

Figure 16. PSW Register

INTERRUPT FLAGS

The lower eight bits of the PSW are used to individually mask the various sources of interrupt to the 8096. A logical '1' in these bit positions enables the servicing of the corresponding interrupt. These mask bits can be accessed as an eight bit byte (INT_MASK—address 8) in the on-board register file. Bit 9 in the PSW is the global interrupt disable. If this bit is cleared then all interrupts will be locked out except for the Non Maskable Interrupt (NMI). Note that the various interrupts are collected in the INT_PENDING register even if they are locked out. Execution of the corresponding service routines will procede according to their priority when they become enabled. Further information on the interrupt structure of the 8096 can be found in Section 4.

CONDITION FLAGS

The remaining bits in the PSW are set as side effects of instruction execution and can be tested by the conditional jump instructions.

Z. The Z (Zero) flag is set to indicate that the operation generated a result equal to zero. For the add-with-carry (ADDC) and subtract-with-borrow (SUBC) operations the Z flag is cleared if the result is non-zero but is never set. These two instructions are normally used in conjunction with the ADD and SUB instructions to perform multiple precision arithmetic. The operation of the Z flag for these instructions leaves it indicating the proper result for the entire multiple precision calculation.

N. The N (Negative) flag is set to indicate that the operation generated a negative result. Note that the N flag will be set to the algebraically correct state even if the calculation overflows. When the NEGB instruction is performed on a byte register containing 80H, or the NEG instruction is performed on a word register containing 8000H, the N flag is set.

V. The V (overflow) flag is set to indicate that the operation generated a result which is outside the range that can be expressed in the destination data type. For the SHL, SHLB and SHLL instructions, the V flag will be set if the most significant bit of the operand changes at any time during the shift.

VT. The VT (oVerflow Trap) flag is set whenever the V flag is set but can only be cleared by an instruction which explicitly operates on it such as the CLRVT or JVT instructions. The operation of the VT flag allows for the testing for a possible overflow condition at the end of a sequence of related arithmetic operations. This is normally more efficient than testing the V flag after each instruction.

C. The C (Carry) flag is set to indicate the state of the arithmetic carry from the most significant bit of the ALU for an arithmetic operation or the state of the last bit shifted out of the operand for a shift. Arithmetic Borrow after a subtract operation is the complement of the C flag (i.e. if the operation generated a borrow then C = 0).

ST. The ST (STicky bit) flag is set to indicate that during a right shift a 1 has been shifted first into the C flag and then been shifted out. The ST flag is undefined after a multiply operation. The ST flag can be used along with the C flag to control rounding after a right shift. Consider multiplying two eight bit quantities and then scaling the result down to 12 bits:

```
MULUB   AX,CL,DL    ;AX:=CL*DL
SHR     AX,#4       ;Shift right 4 places
```

If the C flag is set after the shift, it indicates that the bits shifted off the end of the operand were greater-than or equal-to one half the least significant bit (LSB) of the result. If the C flag is clear after the shift, it indicates that the bits shifted off the end of the operand were less than half the LSB of the result. Without the ST flag, the rounding decision must be made on the basis of this information alone. (Normally the result would be rounded up if the C flag is set.) The ST flag allows a finer resolution in the rounding decision:

C ST	Value of the Bits Shifted Off
0 0	Value = 0
0 1	0 < Value < ½ LSB
1 0	Value = ½ LSB
1 1	Value > ½ LSB

Figure 17. Rounding Alternatives

Imprecise rounding can be a major source of error in a numerical calculation; use of the ST flag improves the options available to the programmer.

3.4 Instruction Set

The MCS-96 instruction set contains a full set of arithmetic and logical operations for the 8-bit data types BYTE and SHORT INTEGER and for the 16-bit data types WORD and INTEGER. The DOUBLE-WORD and LONG data types (32 bits) are supported for the products of 16 by 16 multiplies and the dividends of 32 by 16 divides and for shift operations. The remaining operations on 32-bit variables can be implemented by combinations of 16-bit operations. As an example the sequence:

```
ADD     AX,CX
ADDC    BX,DX
```

performs a 32-bit addition, and the sequence

```
SUB     AX,CX
SUBC    BX,DX
```

performs a 32-bit subtraction. Operations on REAL (i.e. floating point) variables are not supported directly by the hardware but are supported by the floating point library for the 8096 (FPAL-96) which implements a single precision subset of the proposed IEEE standard for floating point operations. The performance of this software is significantly improved by the 8096 NORML instruction which normalizes a 32-bit variable and by the existence of the ST flag in the PSW.

In addition to the operations on the various data types, the 8096 supports conversions between these types. LDBZE (load byte zero extended) converts a BYTE to a WORD and LDBSE (load byte sign extended) converts a SHORT-INTEGER into an INTEGER. WORDS can be converted to DOUBLE-WORDS by simply clearing the upper WORD of the DOUBLE-WORD (CLR) and INTEGERS can be converted to LONGS with the EXT (sign extend) instruction.

The MCS-96 instructions for addition, subtraction, and comparison do not distinguish between unsigned words and signed integers. Conditional jumps are provided to allow the user to treat the results of these operations as either signed or unsigned quantities. As an example, the CMPB (compare byte) instruction is used to compare both signed and unsigned eight bit quantities. A JH (jump if higher) could be used following the compare if unsigned operands were involved or a JGT (jump if greater-than) if signed operands were involved.

Table 3 summarizes the operation of each of the instructions. Complete descriptions of each instruction and its timings can be found in the Instruction Set chapter. A summary of instruction opcodes and timing is included in the quick reference section at the end of this chapter. Examples of using the instruction set of the MCS-96 family can be found in Application Note AP-248, "Using the 8096", included in this handbook.

Table 3. Instruction Summary

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	✓	✓	✓	✓	↑	—	
ADD/ADDB	3	D ← B + A	✓	✓	✓	✓	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	✓	✓	✓	↑	—	
SUB/SUBB	2	D ← D − A	✓	✓	✓	✓	↑	—	
SUB/SUBB	3	D ← B − A	✓	✓	✓	✓	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	✓	✓	✓	↑	—	
CMP/CMPB	2	D − A	✓	✓	✓	✓	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	✓	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	✓	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	
AND/ANDB	2	D ← D and A	✓	✓	0	0	—	—	
AND/ANDB	3	D ← B and A	✓	✓	0	0	—	—	
OR/ORB	2	D ← D or A	✓	✓	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	✓	✓	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ✓	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR [indirect]	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Table 3. Instruction Summary (Continued)

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 − D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign(D)	✓	✓	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — — lsb ← 0	✓	?	✓	✓	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — — lsb → C	✓	?	✓	0	—	✓	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — — lsb → C	✓	✓	✓	0	—	✓	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left shift till msb = 1; D ← shift count	✓	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

3.5 Software Standards and Conventions

For a software project of any size it is a good idea to modularize the program and to establish standards which control the communication between these modules. The nature of these standards will vary with the needs of the final application. A common component of all of these standards, however, must be the mechanism for passing parameters to procedures and returning results from procedures. In the absence of some overriding consideration which prevents their use, it is suggested that the user conform to the conventions adopted by the PLM-96 programming language for procedure linkage. It is a very usable standard for both the assembly language and PLM-96 environment and it offers compatibility between these environments. Another advantage is that it allows the user access to the same floating point arithmetics library that PLM-96 uses to operate on REAL variables.

REGISTER UTILIZATION

The MCS-96 architecture provides a 256 byte register file. Some of these registers are used to control register-mapped I/O devices and for other special functions such as the ZERO register and the stack pointer. The remaining bytes in the register file, some 230 of them, are available for allocation by the programmer. If these registers are to be used effectively, some overall strategy for their allocation must be adopted. PLM-96 adopts the simple and effective strategy of allocating the eight bytes between addresses 1CH and 23H as temporary storage. The starting address of this region is called PLMREG. The remaining area in the register file is treated as a segment of memory which is allocated as required.

ADDRESSING 32-BIT OPERANDS

These operands are formed from two adjacent 16-bit words in memory. The least significant word of the double word is always in lower address, even when the data is in the stack (which means that the most significant word must be pushed into the stack first). A double word is addressed by the address of its least significant byte. Note that the hardware supports some operations on double words (e.g. normalize and divide). For these operations the double word must be in the internal register file and must have an address which is evenly divisible by four.

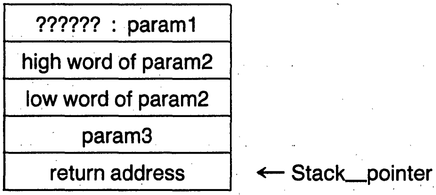
SUBROUTINE LINKAGE

Parameters are passed to subroutines in the stack. Parameters are pushed into the stack in the order that they are encountered in the scanning of the source text. Eight-bit parameters (BYTES or SHORT-INTEGERS) are pushed into the stack with the high order byte undefined. Thirty-two bit parameters (LONG-INTEGERS, DOUBLE-WORDS, and REALS) are pushed into the stack as two 16-bit values; the most significant half of the parameter is pushed into the stack first.

As an example, consider the following PLM-96 procedure:

```
example_procedure: PROCEDURE
(param1,param2,param3);
  DECLARE param1 BYTE,
          param2 DWORD,
          param3 WORD;
```

When this procedure is entered at run time the stack will contain the parameters in the following order:



?????? : param1

high word of param2

low word of param2

param3

return address ← Stack_pointer

Figure 18. Stack Image

If a procedure returns a value to the calling code (as opposed to modifying more global variables) then the result is returned in the variable PLMREG. PLMREG is viewed as either an 8-, 16- or 32-bit variable depending on the type of the procedure.

The standard calling convention adopted by PLM-96 has several key features:

a) Procedures can always assume that the eight bytes of register file memory starting at PLMREG can be used as temporaries within the body of the procedure.

b) Code which calls a procedure must assume that the eight bytes of register file memory starting at PLMREG are modified by the procedure.

c) The Program Status Word (PSW—see Section 3.3) is not saved and restored by procedures so the calling code must assume that the condition flags (Z, N, V, VT, C, and ST) are modified by the procedure.

d) Function results from procedures are always returned in the variable PLMREG.

PLM-96 allows the definition of INTERRUPT procedures which are executed when a predefined interrupt occurs. These procedures do not conform to the rules of a normal procedure. Parameters cannot be passed to these procedures and they cannot return results. Since they can execute essentially at any time (hence the term interrupt), these procedures must save the PSW and PLMREG when they are entered and restore these values before they exit.

4.0 INTERRUPT STRUCTURE

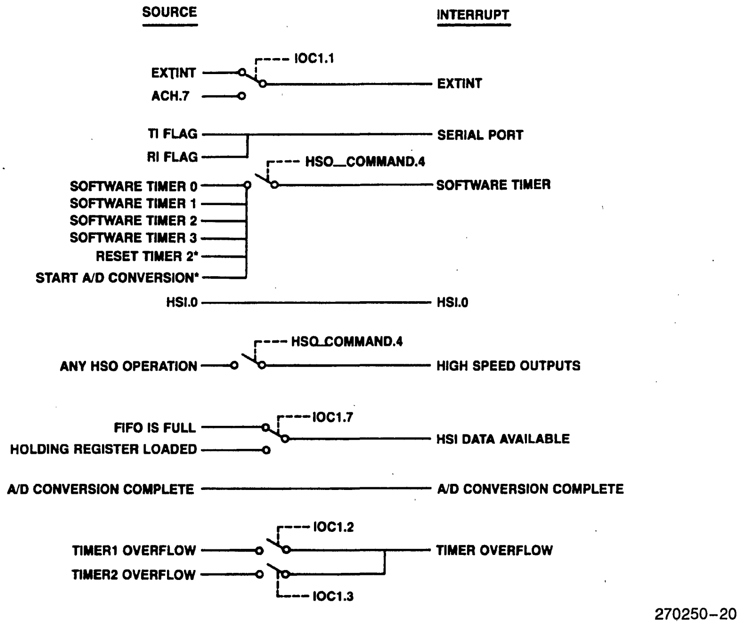
There are 21 sources of interrupts on the 8096BH. These sources are gathered into 8 interrupt types as indicated in Figure 19. The I/O control registers which control some of the sources are indicated in the figure. Each of the eight types of interrupts has its own interrupt vector as listed in Figure 20. In addition to the 8 standard interrupts, there is a TRAP instruction which acts as a software generated interrupt. This instruction is not currently supported by the MCS-96 Assembler and is reserved for use in Intel development systems.

The programmer must initialize the interrupt vector table with the starting address of the appropriate interrupt service routine. It is suggested that any unused interrupts be vectored to an error handling routine. The error routine should contain recovery code that will not further corrupt an already erroneous situation. In a debug environment, it may be desirable to have the routine lock into a jump to self loop which would be easily traceable with emulation tools. More sophisticated routines may be appropriate for production code recoveries.

Three registers control the operation of the interrupt system: Interrupt Pending, Interrupt Mask, and the PSW which contains a global disable bit. A block diagram of the system is shown in Figure 21. The transition detector looks for 0 to 1 transitions on any of the sources. External sources have a maximum transition speed of one edge every state time. If this is exceeded the interrupt may not be detected.

Vector	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 20. Interrupt Vector Locations



NOTE:
*Only when initiated by the HSO unit.

Figure 19. All Possible Interrupt Sources

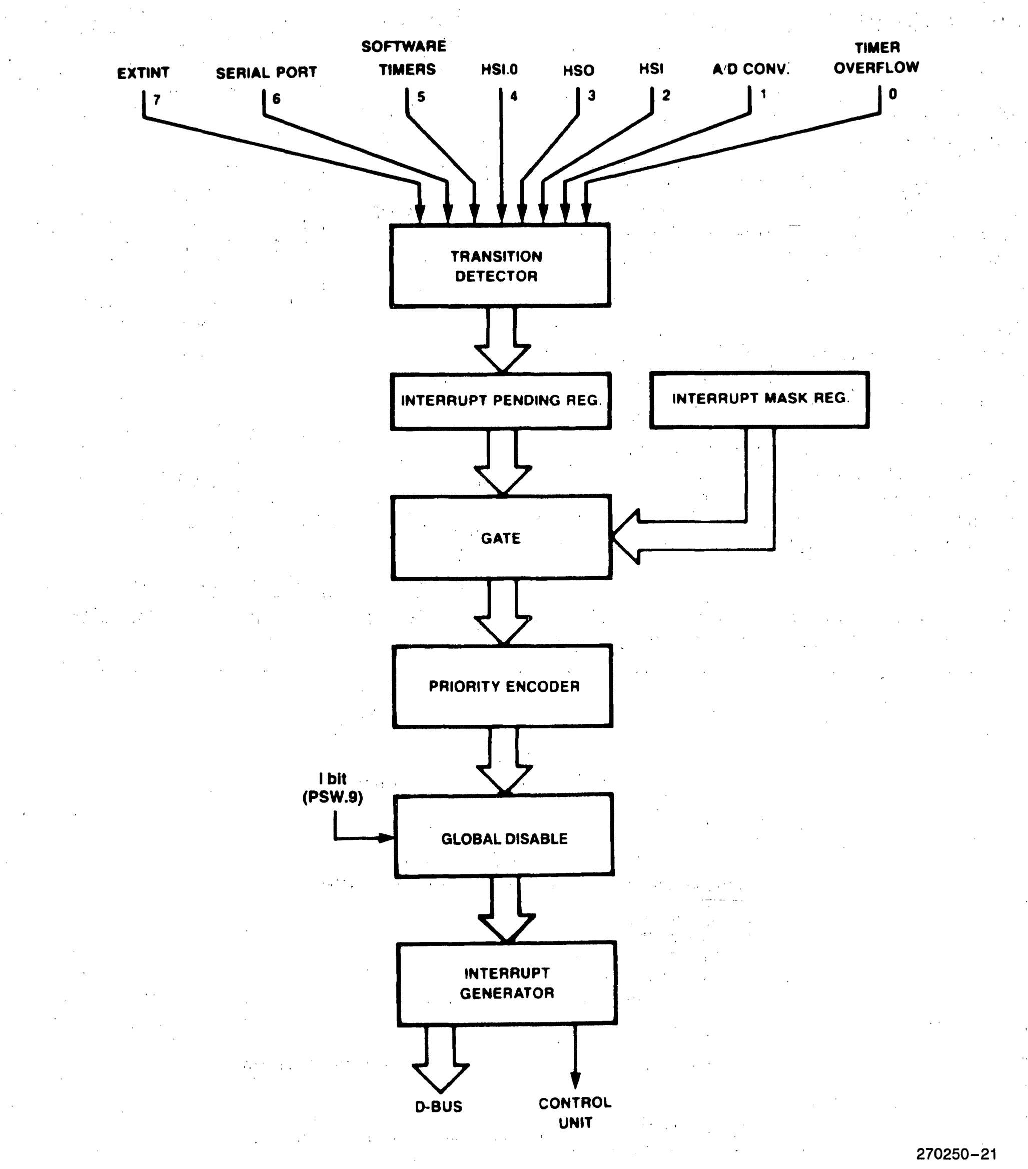


Figure 21. Block Diagram of Interrupt System

4.1 Interrupt Control

Interrupt Pending Register

When the hardware detects one of the eight interrupts it sets the corresponding bit in the pending interrupt register (INT_PENDING-09H). When the interrupt vector is taken, the pending bit is cleared. This register, the format of which is shown in Figure 22, can be read or modified as a byte register. It can be read to determine which of the interrupts are pending at any given time or modified to either clear pending interrupts or generate interrupts under software control. Any software which modifies the INT_PENDING register should ensure that the entire operation is indivisible. The easiest way to do this is to use the logical instructions in the two or three operand format, for example:

```
ANDB  INT_PENDING,#11111101B
            ; Clears the A/D Interrupt
ORB   INT_PENDING,#00000010B
            ; Sets the A/D Interrupt
```

Caution must be used when writing to the pending register to clear interrupts. If the interrupt has already been acknowledged when the bit is cleared, a 4 state time "partial" interrupt cycle will occur. This is because the 8096BH will have to fetch the next instruction of the normal instruction flow, instead of proceeding with the interrupt processing as it was going to. The effect on the program will be essentially that of an extra NOP. This can be prevented by clearing the bits using a 2 operand immediate logical, as the 8096BH holds off acknowledging interrupts during these "read/modify/write" instructions.

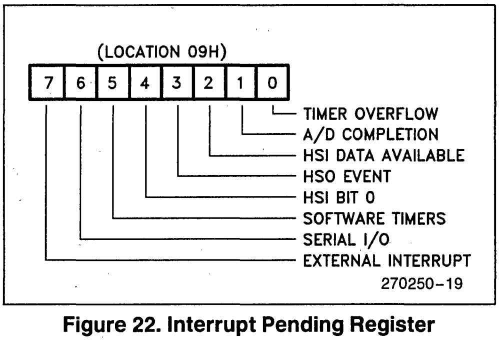


Figure 22. Interrupt Pending Register

Interrupt Mask Register

Individual interrupts can be enabled or disabled by setting or clearing bits in the interrupt mask register (INT_MASK-08H). The format of this register is the same as that of the Interrupt Pending Register shown in Figure 22.

The INT_MASK register can be read or written as byte register. A one in any bit position will enable the corresponding interrupt source and a zero will disable the source. The hardware will save any interrupts that occur by setting bits in the pending register, even if the interrupt mask bit is cleared. The INT_MASK register also can be accessed as the lower eight bits of the PSW so the PUSHF and POPF instructions save and restore the INT_MASK register as well as the global interrupt lockout and the arithmetic flags.

GLOBAL DISABLE

The processing of all interrupts can be disabled by clearing the I bit in the PSW. Setting the I bit will enable interrupts that have mask register bits which are set. The I bit is controlled by the EI (Enable Interrupts) and DI (Disable Interrupts) instructions. Note that the I bit only controls the actual servicing of interrupts. Interrupts that occur during periods of lockout will be held in the pending register and serviced on a prioritized basis when the lockout period ends.

4.2 Interrupt Priorities

The priority encoder looks at all of the interrupts which are both pending and enabled, and selects the one with the highest priority. The priorities are shown in Figure 20 (7 is highest, 0 is lowest). The interrupt generator then forces a call to the location in the indicated vector location. This location would be the starting location of the Interrupt Service Routine (ISR).

This priority selection controls the order in which pending interrupts are passed to the software via interrupt calls. The software can then implement its own priority structure by controlling the mask register (INT_MASK). To see how this is done, consider the case of a serial I/O service routine which must run at a priority level which is lower than the HSI data available interrupt but higher than any other source. The "preamble" and exit code for this interrupt service routine would look like this:

```
serial_io_isr:
 PUSHF          ; Save the PSW
                  (Includes INT_MASK)
 LDB    INT_MASK,#00000100B
 EI             ; Enable interrupts again
 ;  ⎫
 ;  ⎪
 ;  ⎪
 ;  ⎬     Service the interrupt
 ;  ⎪
 ;  ⎪
 ;  ⎭
 POPF           ; Restore the PSW
 RET
```

Note that location 200CH in the interrupt vector table would have to be loaded with the value of the label serial__io__isr and the interrupt be enabled for this routine to execute.

There is an interesting chain of instruction side-effects which makes this (or any other) 8096 interrupt service routine execute properly:

a) After the hardware decides to procesз an interrupt, it generates and executes a special interrupt-call instruction, which pushes the current program counter onto the stack and then loads the program counter with the contents of the vector table entry corresponding to the interrupt. The hardware will not allow another interrupt to be serviced immediately following the interrupt-call. This guarantees that once the interrupt-call starts, the first instruction of the interrupt service routine will execute.

b) The PUSHF instruction, which is now guaranteed to execute, saves the PSW in the stack and then clears the PSW. The PSW contains, in addition to the arithmetic flags, the INT__MASK register and the global disable flag (I). The hardware will not allow an interrupt following a PUSHF instruction and, by the time the LD instruction starts, all of the interrupt enable flags will be cleared. Now there is guaranteed execution of the LD INT__MASK instruction.

c) The LD INT__MASK instruction enables those interrupts that the programmer chooses to allow to interrupt the serial I/O interrupt service routine. In this example only the HSI data available interrupt will be allowed to do this but any interrupt or combination of interrupts could be enabled at this point, even the serial interrupt. It is the loading of the INT__MASK register which allows the software to establish its own priorities for interrupt servicing independently from those that the hardware enforces.

d) The EI instruction reenables the processing of interrupts.

e) The actual interrupt service routine executes within the priority structure established by the software.

f) At the end of the service routine the POPF instruction restores the PSW to its state when the interrupt-call occurred. The hardware will not allow interrupts to be processed following a POPF instruction so the execution of the last instruction (RET) is guaranteed before further interrupts can occur. The reason that this RET instruction must be protected in this fashion is that it is quite likely that the POPF instruction will reenable an interrupt which is already pending. If this interrupt were serviced before the RET instruction, then the return address to the code that was executing when the original interrupt occurred would be left on the stack. While this does not present a problem to the program flow, it could result in a stack overflow if interrupts are occurring at a high frequency. The POPF instruction also pops the INT__MASK register (part of the PSW), so any changes made to this register during a routine which ends with a POPF will be lost.

Notice that the "preamble" and exit code for the interrupt service routine does not include any code for saving or restoring registers. This is because it has been assumed that the interrupt service routine has been allocated its own private set of registers from the onboard register file. The availability of some 230 bytes of register storage makes this quite practical.

4.3 Critical Regions

Interrupt service routines must share some data with other routines. Whenever the programmer is coding those sections of code which access these shared pieces of data, great care must be taken to ensure that the integrity of the data is maintained. Consider clearing a bit in the interrupt pending register as part of a non-interrupt routine:

```
LDB       AL,INT_PENDING
ANDB      AL,#bit_mask
STB       AL,INT_PENDING
```

This code works if no other routines are operating concurrently, but will cause occasional but serious problems if used in a concurrent environment. (All programs which make use of interrupts must be considered to be part of a concurrent environment.) To demonstrate this problem, assume that the INT__PENDING register contains 00001111B and bit 3 (HSO event interrupt pending) is to be reset. The code does work for this data pattern but what happens if an HSI interrupt occurs somewhere between the LDB and the STB instructions? Before the LDB instruction INT__PENDING contains 00001111B and after the LDB instruction so does AL. If the HSI interrupt service routine executes at this point then INT__PENDING will change to 00001011B. The ANDB changes AL to 00000111B and the STB changes INT__PENDING to 00000111B. It should be 00000011B. This code sequence has manged to generate a false HSI interrupt The same basic process can generate an amazing assortment of problems and headaches. These problems can be avoided by assuring mutual exclusion which basically means that if more than one routine can change a variable, then the programmer must ensure exclusive access to the variable during the entire operation on the variable.

In many cases the instruction set of the 8096 allows the variable to be modified with a single instruction. The code in the above example can be implemented with a single instruction.

```
ANDB        INT_PENDING,#bit_mask
```

Instructions are indivisible so mutual exclusion is ensured in this case. Changes to the INT_PENDING register must be made as a single instruction, since bits can be changed in this register even if interrupts are disabled. Depending on system configurations, several other SFRs might also need to be changed in a single instruction for the same reason.

When variables must be modified without interruption, and a single instruction can not be used, the programmer must create what is termed a critical region in which it is safe to modify the variable. One way to do this is to simply disable interrupts with a DI instruction, perform the modification, and then re-enable interrupts with an EI instruction. The problem with this approach is that it leaves the interrupts enabled even if they were not enabled at the start. A better solution is to enter the critical region with a PUSHF instruction which saves the PSW and also clears the interrupt enable flags. The region can then be terminated with a POPF instruction which returns the interrupt enable to the state it was in before the code sequence. It should be noted that some system configurations might require more protection to form a critical region. An example is a system in which more than one processor has access to a common resource such as memory or external I/O devices.

4.4 Interrupt Timing

Interrupts are not always acknowledged immediately. If the interrupt signal does not occur prior to 4 state-times before the end of an instruction, the interrupt will not be acknowledged until after the next instruction has been executed. This is because an instruction is fetched and prepared for execution a few state times before it is actually executed.

There are 6 instructions which always inhibit interrupts from being acknowledged until after the next instruction has been executed. These instructions are:

EI, DI — Enable and Disable Interrupts

POPF, PUSHF— Pop and Push Flags

SIGND — Prefix to perform signed multiply and divide (Note that this is not an ASM-96 Mnemonic, but is used for signed multiply and divide)

TRAP — Software interrupt

When an interrupt is acknowledged, the interrupt pending bit is cleared, and a call is forced to the location indicated by the specified interrupt vector. This call occurs after the completion of the instruction in process, except as noted above. The procedure of getting the vector and forcing the call requires 21 state times. If the stack is in external RAM an additional 3 state times are required.

The maximum number of state times required from the time an interrupt is generated (not acknowledged) until the 8096 begins executing code at the desired location is the time of the longest instruction, NORML (Normalize — 42 state times), plus the 4 state times prior to the end of the previous instruction, plus the response time (21 to 24 state times). Therefore, the maximum response time is 70 (42 + 4 + 24) state times. This does not include the 12 state times required for PUSHF if it is used as the first instruction in the interrupt routine or additional latency caused by having the interrupt masked or disabled. Refer to Figure 22A, Interrupt Response Time, to visualize an example of worst case scenario.

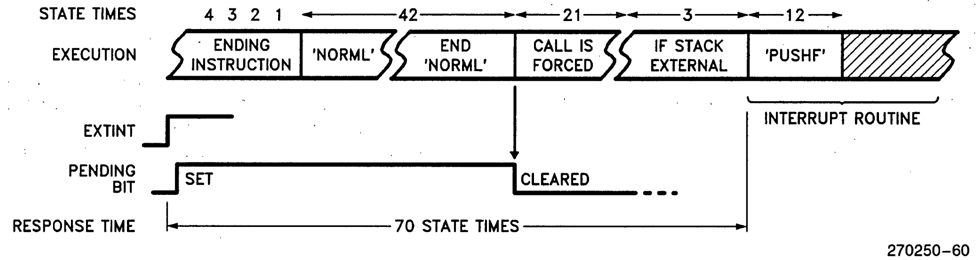


Figure 22A. Interrupt Response Time

Interrupt latency time can be reduced by careful selection of instructions in areas of code where interrupts are expected. Using 'EI' followed immediately by a long instruction (e.g. MUL, NORML, etc.) will increase the maximum latency by 4 state times, as an interrupt cannot occur between EI and the instruction following EI. The "DI", "PUSHF", "POPF" and "TRAP" instructions will also cause the same situation. Typically the PUSHF, POPF and TRAP instructions would only effect latency when one interrupt routine is already in process, as these instructions are seldom used at other times.

5.0 TIMERS

Two 16-bit timers are available for use on the 8096. The first is designated "Timer 1", the second, "Timer 2". Timer 1 is used to synchronize events to real time, while Timer 2 can be clocked externally and synchronizes events to external occurrences.

5.1 Timer 1

Timer 1 is clocked once every eight state times and can be cleared only by executing a reset. The only other way to change its value is by writing to 000CH but this is a test mode which sets both timers to 0FFFXH and should not be used in programs.

5.2 Timer 2

Timer 2 can be incremented by transitions (one count each transition, rising *and* falling) on either T2CLK or HSI.1. The multiple functionality of the timer is determined by the state of I/O Control Register 0, bit 7 (IOC0.7). To ensure that all CAM entries are checked each count of Timer 2, the maximum transition speed is limited to once per eight state times. Timer 2 can be cleared by: executing a reset, by setting IOC0.1, by triggering HSO channel 0EH, or by pulling T2RST or HSI.0 high. The HSO and CAM are described in Section 7 and 8. IOC0.3 and ICO0.5 control the resetting of Timer 2. Figure 23 shows the different ways of manipulating Timer 2.

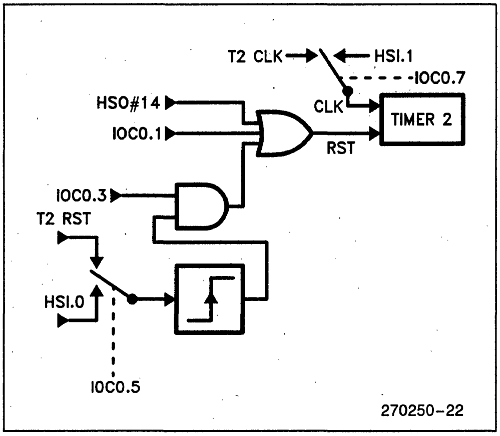


Figure 23. Timer 2 Clock and Reset Options

5.3 Timer Interrupts

Both Timer 1 and Timer 2 can be used to trigger a timer overflow interrupt and set a flag in the I/O Status Register 1 (IOS1). The interrupts are controlled by IOC1.2 and IOC1.3 respectively. The flags are set in IOS1.5 and IOS1.4, respectively.

Caution must be used when examining the flags, as any access (including Compare and Jump on Bit) of IOS1 clears bits 0 through 5 including the software timer flags. It is, therefore, recommended to write the byte to a temporary register before testing bits. The general enabling and disabling of the timer interrupts are controlled by the Interrupt Mask Register bit 0. In all cases, setting a bit enables a function, while clearing a bit disables it.

5.4 Timer Related Sections

The High Speed I/O unit is coupled to the timers in that the HSI records the value on Timer 1 when transitions occur and the HSO causes transitions to occur based on values of either Timer 1 or Timer 2. The baud rate generator can use the T2CLK pin as input to its counter. a complete listing of the functions of IOS1, IOC0, and IOC1 are in Section 11.

6.0 HIGH SPEED INPUTS

The High Speed Input Unit (HSI), can be used to record the time at which an event occurs with respect to Timer 1. There are 4 lines (HSI.0 through HSI.3) which can be used in this mode and up to a total of 8 events can be recorded. HSI.2 and HSI.3 are bidirectional pins which can also be used as HSO.4 and HSO.5. The I/O Control Registers (IOC0 and IOC1) are used to determine the functions of these pins. A block diagram of the HSI unit is shown in Figure 24.

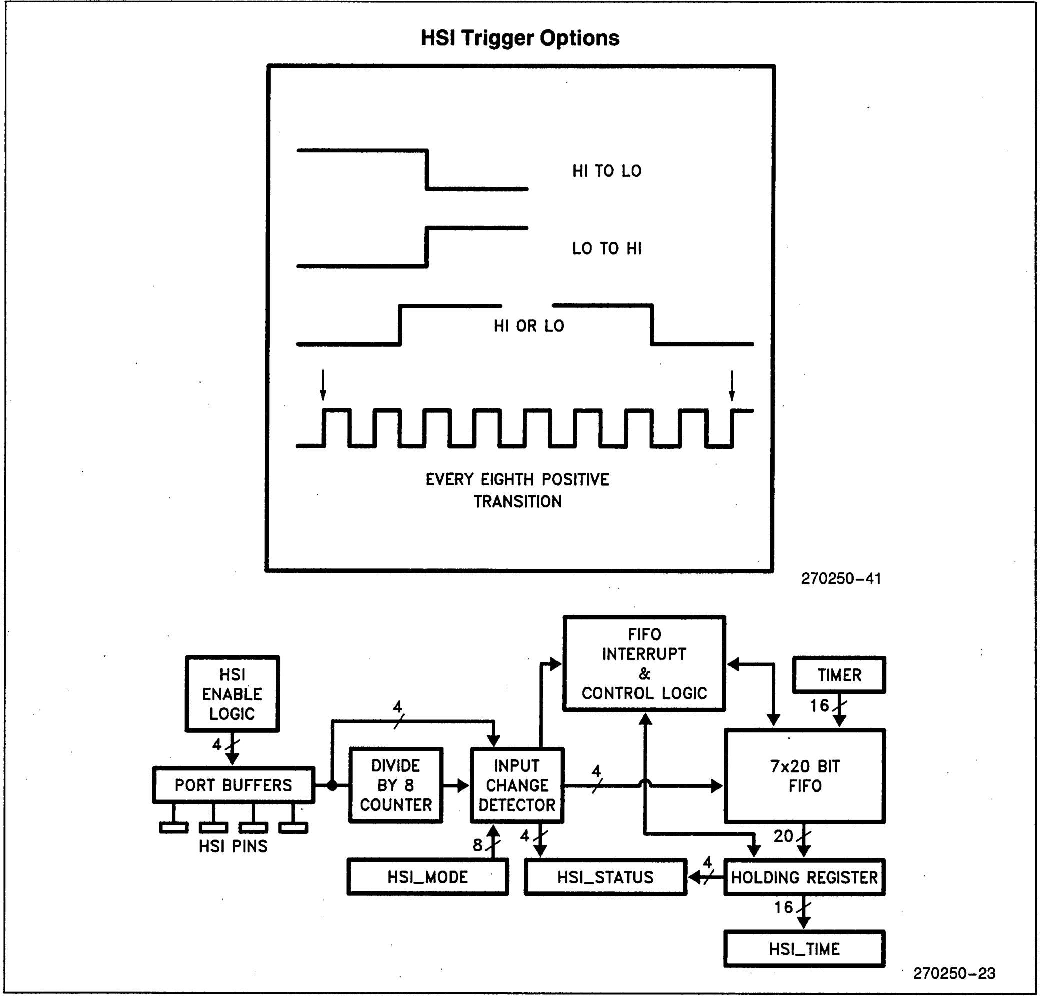


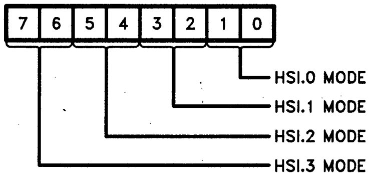
Figure 24. High Speed Input Unit

6.1 HSI Modes

There are 4 possible modes of operation for each of the HSI pins. The HSI mode register is used to control which pins will look for what type of events. The 8-bit register is set up as shown in Figure 25.

High and low levels each need to be held for at least 1 state time to ensure proper operation. The maximum input speed is 1 event every 8 state times except when the 8 transition mode is used, in which case it is 1 transition per state time. The divide by eight counter can only be zeroed in mid-count by performing a hardware reset on the 8096BH. The 8X9X counter cannot be zeroed.

HSI_Mode (03H)



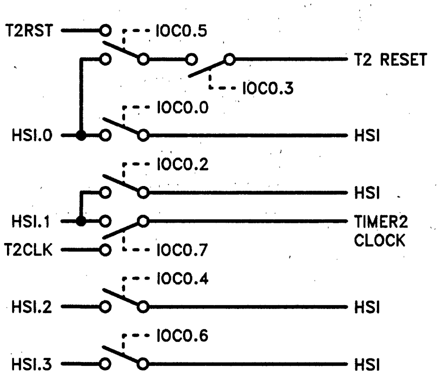
WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00	8 POSITIVE TRANSITIONS
01	EACH POSITIVE TRANSITION
10	EACH NEGATIVE TRANSITION
11	EVERY TRANSITION (POSITIVE AND NEGATIVE)

270250-24

Figure 25. HSI Mode Register Diagram

The HSI lines can be individually enabled and disabled using bits in IOC0, at location 0015H. Figure 26 shows the bit locations which control the HSI pins. If the pin is disabled, transitions will not be entered in the FIFO.



270250-25

Figure 26. IOC0 Control of HSI Pin Functions

6.2 HSI FIFO

When an HSI event occurs, a 7×20 FIFO stores the 16 bits of Timer 1 and the 4 bits indicating which pins had events. It can take up to 8 state times for this information to reach the holding register. For this reason, 8 state times must be allowed between consecutive reads of HSI_TIME. When the FIFO is full, one additional event, for a total of 8 events, can be stored by considering the holding register part of the FIFO. If the FIFO and holding register are full, any additional events will not be recorded.

6.3 HSI Interrupts

Interrupts can be generated by the HSI unit in three ways; two FIFO related interrupts and 0 to 1 transitions on the HSI.0 pin. The HSI.0 pin can generate interrupts even if it is not enabled to the HSI FIFO. Interrupts generated by this pin cause a vector through location 2008H. The FIFO related interrupts are controlled by bit 7 of I/O Control Register 1, (IOC1.7). If the bit is a 0, then an interrupt will be generated every time a value is loaded into the holding register. If it is a 1, an interrupt will only be generated when the FIFO, (independent of the holding register), has six entries in it. Since all interrupts are rising edge triggered, if IOC1.7 = 1, the processor will not be re-interrupted until the FIFO first contains 5 or less records, then contains six or more.

6.4 HSI Status

Bits 6 and 7 of the I/O Status register 1 (IOS1) indicate the status of the HSI FIFO. If bit 6 is a 1, the FIFO contains at least six entries. If bit 7 is a 1, the FIFO contains at least 1 entry and the HSI holding register has data available to be read. The FIFO may be read after verifying that it contains valid data. Caution must be used when reading or testing bits in IOS1, as this action clears bits 0–5, including the software and hardware timer overflow flags. It is best to store the byte and then test the stored value. See Section 11.

Reading the HSI is done in two steps. First, the HSI Status register is read to obtain the current state of the HSI pins and which pins had changed at the recorded time. The format of the HSI_STATUS Register is shown in Figure 27. Second, the HSI Time register is read. Reading the Time register unloads one level of the FIFO, so if the Time register is read before the Status register, the event information in the Status register will be lost. The HSI Status register is at location 06H and the HSI Time registers are in locations 04H and 05H.

If the HSI_TIME register is read without the holding register being loaded, the returned value will be indeterminate. Under the same conditions, the four bits in

HSI_STATUS indicating which events have occurred will also be indeterminate. The four HSI_STATUS bits which indicate the current state of the pins will always return the correct value.

It should be noted that many of the Status register conditions are changed by a reset, see Section 13. A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in Section 11.

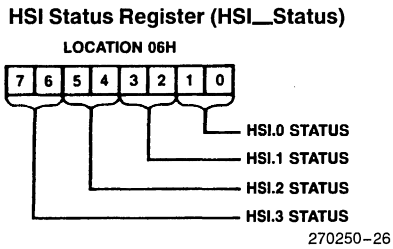
7.0 HIGH SPEED OUTPUTS

The High Speed Output unit, (HSO), is used to trigger events at specific times with minimal CPU overhead. These events include: starting an A to D conversion, resetting Timer 2, setting 4 software flags, and switching 6 output lines (HSO.0 through HSO.5). Up to eight events can be pending at one time and interrupts can be generated whenever any of these events are triggered. HSO.4 and HSO.5 are bidirectional pins which can also be used as HSI.2 and HSI.3 respectively. Bits 4 and 6 of I/O Control Register 1, (IOC1.4, IOC1.6), enable HSO.4 and HSO.5 as outputs.

The HSO unit can generate two types of interrupts. The HSO execution interrupt (vector = (2006H)) is generated (if enabled) for HSO commands which operate one or more of the six output pins. The other HSO interrupt is the software timer interrupt (vector = (200BH)) which is generated (if enabled) by any other HSO command, (e.g. triggering the A/D, resetting Timer 2 or generating a software time delay).

7.1 HSO CAM

A block diagram of the HSO unit is shown in Figure 28. The Content Addressable Memory (CAM) file is the center of control. One CAM register is compared with the timer values every state time, taking 8 state times to compare all CAM registers with the timers. This defines the time resolution of the HSO to be 8 state times (2.0 microseconds at an oscillator frequency of 12 MHz).



Where for each 2-bit status field the lower bit indicates whether or not an event has occurred on this pin at the time in HSI_TIME and the upper bit indicates the current status of the pin.

Figure 27. HSI Status Register Diagram

Each CAM register is 23 bits wide. Sixteen bits specify the time at which the action is to be carried out and 7 bits specify both the nature of the action and whether Timer 1 or Timer 2 is the reference. The format of the

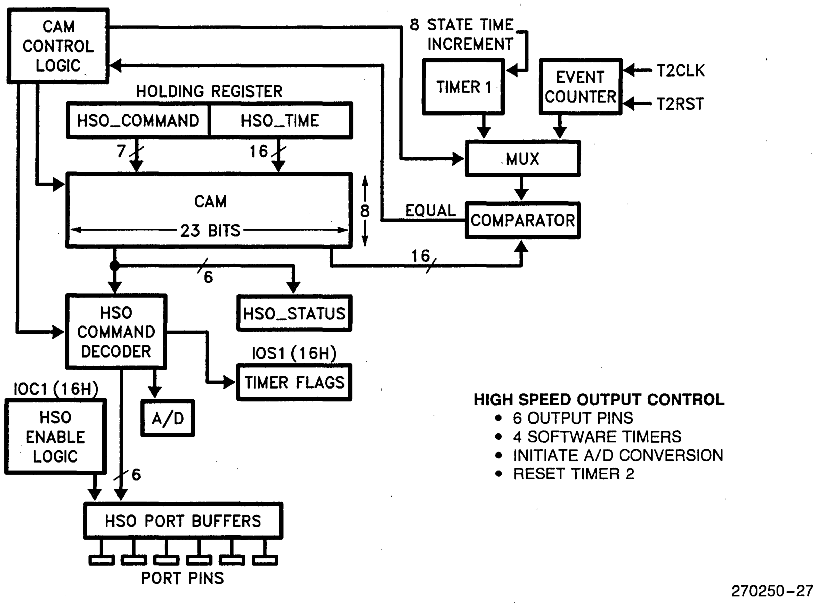


Figure 28. High Speed Output Unit

command to the HSO unit is shown in Figure 29. Note that bit 5 is ignored for command channels 8 through 0FH.

To enter a command into the CAM file, write the 7-bit "Command Tag" into location 0006H followed by the time at which the action is to be carried out into word address 0004H. The typical code would be:

```
LDB HSO_COMMAND,#what_to_do
ADD HSO_TIME,TIMER1,#when_to_do_it
```

Writing the time value loads the HSO Holding Register with both the time and the last written command tag. The command does not actually enter the CAM file until an empty CAM register becomes available.

Commands in the holding register will not execute even if their time tag is reached. Commands must be in the CAM for this to occur. Commands in the holding register can also be overwritten. Since it can take up to 8 state times for a command to move from the holding register to the CAM, 8 states must be allowed between successive writes to the CAM.

To provide proper synchronization, the minimum time that should be loaded to Timer 1 is Timer 1 + 2. Smaller values may cause the Timer match to occur 65,636 counts later than expected. A similar restriction applies if Timer 2 is used.

Care must be taken when writing the command tag for the HSO. If an interrupt occurs during the time between writing the command tag and loading the time value, and the interrupt service routine writes to the HSO time register, the command tag used in the interrupt routine will be written to the CAM at both the time specified by the interrupt routine and the time specified by the main program. The command tag from the main program will not be executed. One way of avoiding this problem would be to disable interrupts when writing commands and times to the HSO unit. See also Section 4.5.

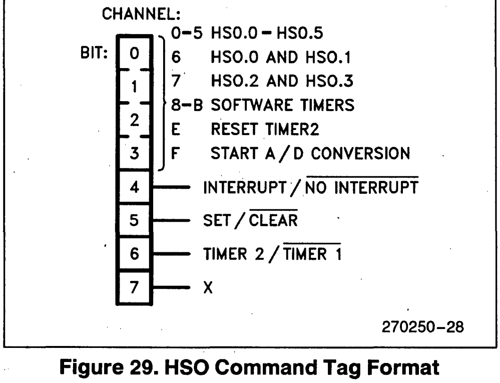


Figure 29. HSO Command Tag Format

7.2 HSO Status

Before writing to the HSO, it is desirable to ensure that the Holding Register is empty. If it is not, writing to the HSO will overwrite the value in the Holding Register. I/O Status Register 0 (IOSO) bits 6 and 7 indicate the status of the HSO unit. This register is described in Section 11. If ISO0.6 equals 0, the holding register is empty and at least one CAM register is empty. If IOS0.7 equals 0, the holding register is empty.

The programmer should carefully decide which of these two flags is the best to use for each application.

7.3 Clearing the HSO

All 8 CAM locations of the HSO are compared before any action is taken. This allows a pending external event to be cancelled by simply writing the opposite event to the CAM. However, once an entry is placed in the CAM, it cannot be removed until either the specified timer matches the written value or the chip is reset. If, as an example, a command has been issued to set HSO.1 when TIMER 1 = 1234, then entering a second command which clears HSO.1 when TIMER 1 = 1234 will result in no operation on HSO.1. Both commands will remain in the CAM until TIMER 1 = 1234.

Internal events are not synchronized to Timer 1, and therefore cannot be cleared. This includes events on HSO channels 8 through F and all interrupts. Since interrupts are not synchronized it is possible to have multiple interrupts at the same time value.

7.4 Using Timer 2 with the HSO

Timer 1 is incremented only once every 8 state-times. When it is being used as the reference timer for an HSO action, the comparator has a chance to look at all 8 CAM registers before Timer 1 changes its value. Following the same reasoning, Timer 2 has been synchronized to allow it to change at a maximum rate of once per 8 state-times. Timer 2 increments on both edges of the input signal.

When using Timer 2 as the HSO reference, caution must be taken that Timer 2 is not reset prior to the highest value for a Timer 2 match in the CAM. This is because the HSO CAM will hold an event pending until a time match occurs, if that match is to a time value on Timer 2 which is never reached, the event will remain pending in the CAM until the part is reset.

Additional caution must be used when Timer 2 is being reset using the HSO unit, since resetting Timer 2 using the HSO is an internal event and can therefore happen at any time within the eight-state-time window. This situation arises when the event is set to occur when

Timer 2 is equal to zero. If HSI.0 or the T2RST pin is used to clear Timer 2, and Timer 2 equal to zero triggers the event, then the event may not occur. This is because HSI.0 and T2RST clear Timer 2 asynchronously, and Timer 2 may then be incremented to one before the HSO CAM entry can be read and acted upon. This can be avoided by setting the event to occur when Timer 2 is equal to one. This method will ensure that there is enough time for the CAM entry recognition.

The same asynchronous nature can affect events scheduled to occur at the same time as an internal Timer 2 reset. These events should be logged into the CAM with a Timer 2 value of zero. When using this method to make a programmable modulo counter, the count will stay at the maximum Timer 2 value only until the Reset T2 command is recognized. The count will stay at zero for the transition which would have changed the count from "N" to zero, and then changed to a one on the next transition.

7.5 Software Timers

The HSO can be programmed to generate interrupts at preset times. Up to four such "Software Timers" can be in operation at a time. As each preprogrammed time is reached, the HSO unit sets a Software Timer Flag. If the interrupt bit in the command tag was set then a Software Timer Interrupt will also be generated. The interrupt service routine can then examine I/O Status register 1 (IOS1) to determine which software timer expired and caused the interrupt. When the HSO resets Timer 2 or starts an A to D conversion, it can also be programmed to generate a software timer interrupt but there is no flag to indicate that this has occurred.

If more than one software timer interrupt occurs in the same time frame it is possible that multiple software timer interrupts will be generated.

Each read or test of any bit in IOS1 will clear bits 0 through 5. Be certain to save the byte before testing it unless you are only concerned with 1 bit. See also Section 11.5.

A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in Section 11. The Timers are described in Section 5 and the HSI is described in Section 6.

8.0 ANALOG INTERFACE

The 8096H can easily interface to analog signals using its Analog to Digital Converter and its Pulse-Width-Modulated (PWM) output and HSO Unit. Analog inputs are accepted by the 8-input, 10-bit A to D converter. The PWM and HSO units provide digital signals which can be filtered for use as analog outputs.

8.1 Analog Inputs

A to D conversion is performed on one of the 8 inputs at a time using successive approximation with a result equal to the ratio of the input voltage divided by the analog supply voltage. If the ratio is 1.00, then the result will be all ones. The A/D converter is available on selected members of the MCS-96 family. See Section 14 for the device selection matrix.

Each conversion on the 8096BH requires 88 state-times (22 μs at 12 MHz) independent of the accuracy desired or value of input voltage. The input voltage must be in the range of 0 to VREF, the analog reference and supply voltage. For proper operation, VREF (the reference voltage and analog power supply) must be held nominally at 5V. The A/D result is calculated from the formula:

1023 × (input voltage-ANGND)/(VREF-ANGND)

It can be seen from this formula that changes in VREF or ANGND effect the output of the converter. This can be advantageous if a ratiometric sensor is used since these sensors have an output that can be measured as a proportion of VREF.

ANGND must be tied to VSS (digital ground) in order for the 8096BH to operate properly. This common connection should be made as close to the chip as possible, and using good bulk and high frequency by-pass capacitors to decouple power supply variations and noise from the circuit. Analog design rules call for one and only one common connection between analog and digital returns to eliminate unwanted ground variations.

A sample and hold is provided on the A/D converter of the 8X97BH and 8X95BH. The sampling window is open for 4 state times which are included in the 88 state-time conversion period. The exact timings of the A/D converter can be found in Section 3 of the Hardware Design chapter.

The 8X9X devices do not have a sample and hold, so the input voltage must be held constant through the entire conversion. The conversion time is 168 state times (42 μs at 12 MHz) on the 8X9X devices.

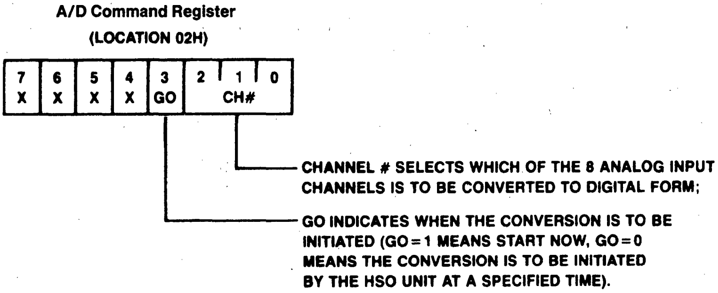
8.2 A/D Commands

Analog signals can be sampled by any one of the 8 analog input pins (ACH0 through ACH7) which are shared with Port 0. ACH7 can also be used as an external interrupt if IOC1.1 is set (see Sections 4 and 11). The A/D Command Register, at location 02H, selects which channel is to be converted and whether the conversion should start immediately or when the HSO (Channel #0FH) triggers it. The A/D command register must be written to for each conversion, even if the HSO is used as the trigger. A to D commands are formatted as shown in Figure 30.

The command register is double buffered so it is possible to write a command to start a conversion triggered by the HSO while one is still in progress. Care must be taken when this is done since if a new conversion is started while one is already in progress, the conversion in progress is cancelled and the new one is started. When a conversion is started, the result register is cleared. For this reason the result register must be read before a new conversion is started or data will be lost.

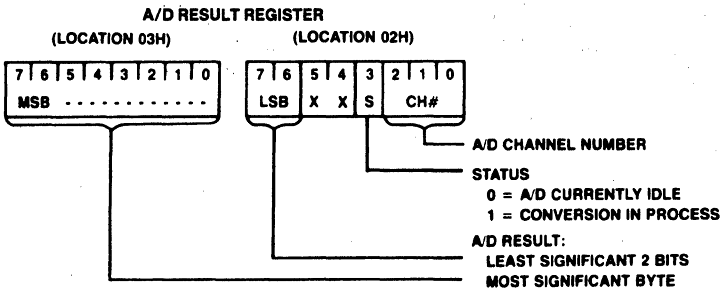
8.3 A/D Results

Results of the analog conversions are read from the A/D Result Register at locations 02H and 03H. Although these addresses are on a word boundary, they must be read as individual bytes. Information in the A/D Result register is formatted as shown in Figure 31. Note that the status bit may not be set until 8 state


A/D Command Register
(LOCATION 02H)
7 X
6 X
5 X
4 X
3 GO
2 1 0 CH#
CHANNEL # SELECTS WHICH OF THE 8 ANALOG INPUT CHANNELS IS TO BE CONVERTED TO DIGITAL FORM;
GO INDICATES WHEN THE CONVERSION IS TO BE INITIATED (GO=1 MEANS START NOW, GO=0 MEANS THE CONVERSION IS TO BE INITIATED BY THE HSO UNIT AT A SPECIFIED TIME).

270250–29

Figure 30. A/D Command Register



270250–30

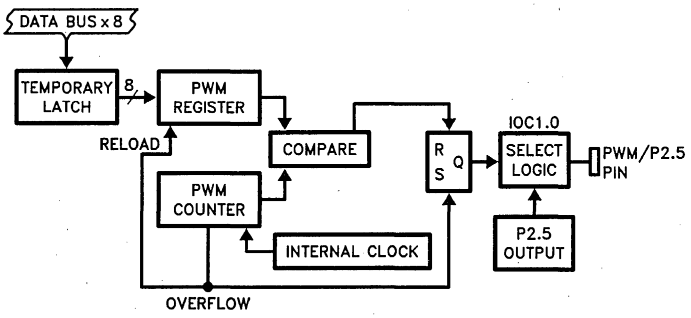
Figure 31. A/D Result Register

times after the go command, so it is necessary to wait 8 state times before testing it. Information on using the HSO is in Section 7.

8.4 Pulse Width Modulation Output (D/A)

Digital to analog conversion can be done with the Pulse Width Modulation output; a block diagram of the circuit is shown in Figure 32. The 8-bit counter is incremented every state time. When it equals 0, the PWM output is set to a one. When the counter matches the value in the PWM register, the output is switched low. When the counter overflows, the output is once again switched high. A typical output waveform is shown in Figure 33. Note that when the PWM register equals 00, the output is always low. Additionally, the PWM register will only be reloaded from the temporary latch when the counter overflows. This means that the compare circuit will not recognize a new value to compare against until the counter has expired the remainder of the current 8-bit count.

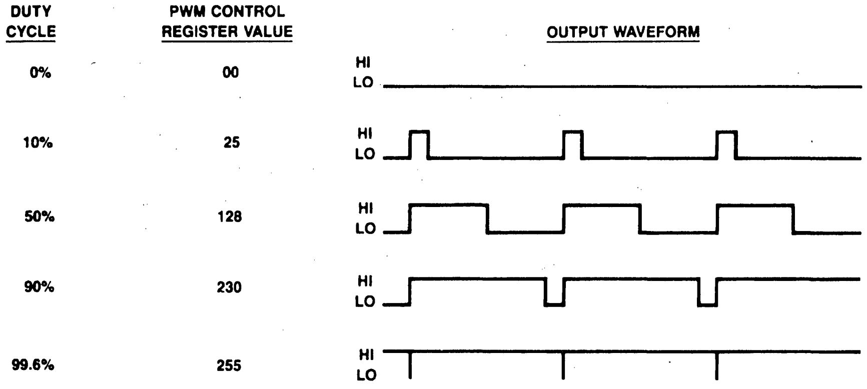
The output waveform is a variable duty cycle pulse which repeats every 256 state times (64 μs at 12 MHz). Changes in the duty cycle are made by writing to the PWM register at location 17H. There are several types of motors which require a PWM waveform for most efficient operation. Additionally, if this waveform is integrated it will produce a DC level which can be changed in 256 steps by varying the duty cycle.



270250–31

- PWM Period (XTAL = 12 MHz) = 64 μs, Frequency = 15.625 KHz
- Duty Cycle Programmable in 256 Steps

Figure 32. Pulse Width Modulated (D/A) Output



270250–32

Figure 33. Typical PWM Outputs

Details about the hardware required for smooth, accurate D/A conversion can be found in Section 4 of the Hardware Design chapter. Typically, some form of buffer and integrator are needed to obtain the most usefulness from this feature.

The PWM output shares a pin with Port 2, pin 5 so that these two features cannot be used at the same time. IOC1.0 equal to 1 selects the PWM function instead of the standard port function. More information on IOC1 is in Section 11.

8.5 PWM Using the HSO

The HSO unit can be used to generate PWM waveforms with very little CPU overhead. If the HSO is not being used for other purposes, a 4 line PWM unit can be made by loading the on and off times into the CAM in sets of 4. The CAM would then always be loaded and only 2 interrupts per PWM period would be needed. An example of using the HSO in this manner can be found AP-248, "Using The 8096". This application note is included in the MCS-96 Application Notes chapter.

9.0 SERIAL PORT

The serial port on the 8096BH has 3 asynchronous and one synchronous mode. The asynchronous modes are full duplex, meaning they can transmit and receive at the same time. The receiver is double buffered so that the reception of a second byte can begin before the first byte has been read. The port is functionally compatible with the serial port on the MCS-51 family of microcontrollers, although the software used to control the ports is different.

Control of the serial port is handled through the Serial Port Control/Status Register at location 11H. Figure 37 shows the layout of this register. The details of using it to control the serial port will be discussed in Section 9.2.

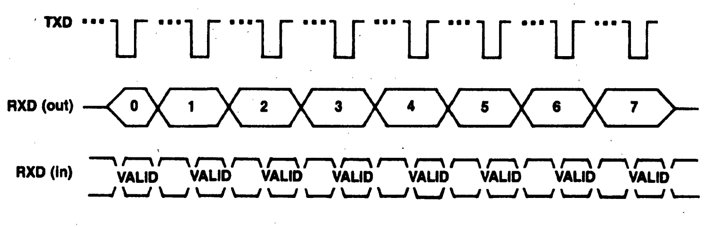
Data to and from the serial port is transferred through SBUF (rx) and SBUF (tx), both located at 07H. Although these registers share the same address, they are physically separate, with SBUF (rx) containing the data received by the serial port and SBUF (tx) used to hold data ready for transmission. The program cannot write to SBUF (rx) or read from SBUF (tx).

The baud rate at which the serial port operates is controlled by an independent baud rate generator. The inputs to this generator can be either the XTAL1 or the T2CLK pin. Details on setting up the baud rate are given in Section 9.3.

9.1 Serial Port Modes

MODE 0

Mode 0 is a synchronous mode which is commonly used for shift register based I/O expansion. In this mode the TXD pin outputs a set of 8 pulses while the RXD pin either transmits or receives data. Data is transferred 8 bits at a time with the LSB first. A diagram of the relative timing of these signals is shown in Figure 34. Note that this is the only mode which uses RXD as an output.



270250–34

Figure 34. Serial Port Mode 0 Timing

Although it is not possible to transmit and receive at the same time using this mode, two external gates and a port pin can be used to time-multiplex the two functions. An example of multiplexing transmit and receive is discussed in Section 6.1 of the Hardware Design chapter.

MODE 1

Mode 1 is the standard asynchronous communications mode. The data frame used in this mode is shown in Figure 35. It consists of 10 bits; a start bit (0), 8 data bits (LSB first), and a stop bit (1). If parity is enabled, (the PEN bit is set to a 1), an even parity bit is sent instead of the 8th data bit and parity is checked on reception.

MODE 2

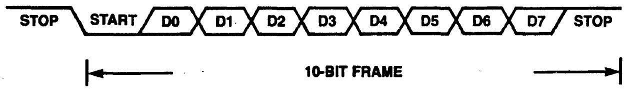
Mode 2 is the asynchronous 9th bit recognition mode. This mode is commonly used with Mode 3 for multiprocessor communications. Figure 36 shows the data frame used in this mode. It consists of a start bit (0), 9 data bits (LSB first), and a stop bit (1). When transmitting, the 9th bit can be set to a one by setting the TB8 bit in the control register before writing to SBUF (tx). The TB8 bit is cleared on every transmission, so it must be set prior to writing to SBUF (tx) each time it is desired. During reception, the serial port interrupt and the Receive Interrupt (RI) bit will not be set unless the 9th bit being received is set. This provides an easy way to have selective reception on a data link. Parity cannot be enabled in this mode.

MODE 3

Mode 3 is the asynchronous 9th bit mode. The data frame for this mode is identical to that of Mode 2. The transmission differences between Mode 3 and Mode 2 are that parity can be enabled (PEN = 1) and cause the 9th data bit to take the even parity value. The TB8 bit can still be used if parity is not enabled (PEN = 0). When in Mode 3, a reception always causes an interrupt, regardless of the state of the 9th bit. The 9th bit is stored if PEN = 0 and can be read in bit RB8. If PEN = 1 then RB8 becomes the Receive Parity Error (RPE) flag.

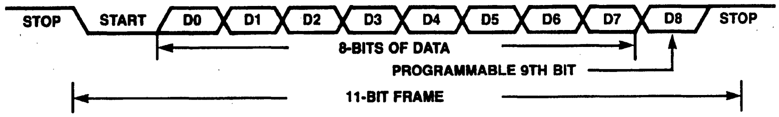
9.2 Controlling the Serial Port

Control of the serial port is done through the Serial Port Control (SP_CON) and Serial Port Status (SP_STAT) registers shown in Figure 37. Writing to location 11H accesses SP_CON while reading it access SP_STAT. Note that reads of SP_STAT will return indeterminate data in the lower 5 bits and writing to the upper 3 bits of SP_CON has no effect on chip functionality. The TB8 bit is cleared after each transmission and both TI and RI are cleared whenever SP_STAT (not SP_CON) is accessed. Whenever the TXD pin is used for the serial port it must be enabled by setting IOC1.5 to a 1. IOC1 is discussed further in Section 11.3. Information on the hardware connections and timing of the serial port is in Section 6 of the Hardware Design chapter.



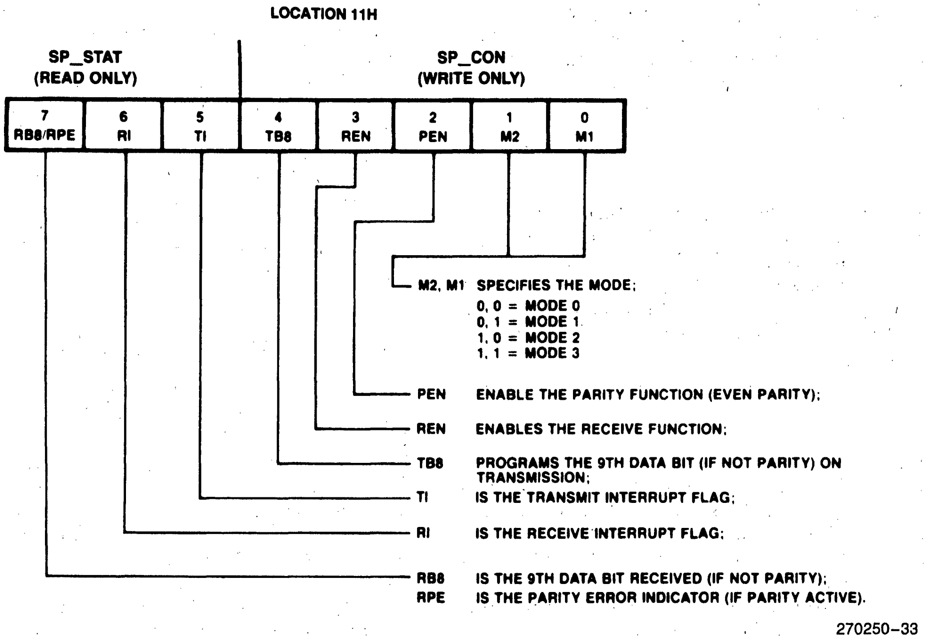
270250-35

Figure 35. Serial Port Frame—Mode 1



270250-36

Figure 36. Serial Port Frame Modes 2 and 3



M2, M1 SPECIFIES THE MODE;
0, 0 = MODE 0
0, 1 = MODE 1
1, 0 = MODE 2
1, 1 = MODE 3

PEN ENABLE THE PARITY FUNCTION (EVEN PARITY);

REN ENABLES THE RECEIVE FUNCTION;

TB8 PROGRAMS THE 9TH DATA BIT (IF NOT PARITY) ON TRANSMISSION;

TI IS THE TRANSMIT INTERRUPT FLAG;

RI IS THE RECEIVE INTERRUPT FLAG;

RB8 IS THE 9TH DATA BIT RECEIVED (IF NOT PARITY);
RPE IS THE PARITY ERROR INDICATOR (IF PARITY ACTIVE).

NOTE:
TI and RI are cleared when SP__STAT is read.

Figure 37. Serial Port Control/Status Register

In Mode 0, if REN = 1, writing to SBUF (tx) will start a transmission. Causing a rising edge on REN, or clearing RI with REN = 1, will start a reception. Setting REN = 0 will stop a reception in progress and inhibit further receptions. To avoid a partial or complete undesired reception, REN must be set to zero before RI is cleared. This can be handled in an interrupt environment by using software flags or in straight-line code by using the Interrupt Pending register to signal the completion of a reception.

In the asynchronous modes, writing to SBUF (tx) starts a transmission. A falling edge on RXD will begin a reception if REN is set to 1. New data placed in SBUF (tx) is held and will not be transmitted until the end of the stop bit has been sent.

In all modes, the RI flag is set after the last data bit is sampled approximately in the middle of the bit time. Also for all modes, the TI flag is set after the last data bit (either 8^{th} or 9^{th}) is sent, also in the middle of the bit time. The flags clear when SP__STAT is read, but do not have to be clear for the port to receive or transmit. The serial port interrupt bit is set as a logical OR of the RI and TI bits. Note that changing modes will reset the Serial Port and abort any transmission or reception in progress on the channel.

9.3 Determining Baud Rates

Baud rates in all modes are determined by the contents of a 16-bit register at location 000EH. This register must be loaded sequentially with 2 bytes (least significant byte first). The serial port will not function between the loading of the first and second bytes. The MSB of this register selects one of two sources for the input frequency to the baud rate generator. If it is a 1, the frequency on the XTAL1 pin is selected, if not, the external frequency from the T2CLK pin is used. It should be noted that the maximum speed of T2CLK is one transition every 2 state times, with a minimum period of 16 XTAL1 cycles. This provides the needed synchronization to the internal serial port clocks.

The unsigned integer represented by the lower 15 bits of the baud rate register defines a number B, where B has a maximum value of 32767. The baud rate for the four serial modes using either XTAL1 or T2CLK as the clock source is given by:

Using XTAL1:

$$\text{Mode 0: } \begin{matrix}\text{Baud}\\\text{Rate}\end{matrix} = \frac{\text{XTAL1 frequency}}{4 * (B + 1)}; \quad B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{64 * (B + 1)}$$

Using T2CLK:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{B}; \quad B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{16 * B}; \quad B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Common baud rate values, using XTAL1 at 12 MHz, are shown below.

Baud Rate	Baud Register Value	
	Mode 0	Others
9600	8137H	8013H
4800	8270H	8026H
2400	84E1H	804DH
1200	89C3H	809BH
300	A70FH	8270H

The maximum baud rates are 1.5 Mbaud synchronous and 187.5 Kbaud asynchronous with 12 MHz on XTAL1.

9.4 Multiprocessor Communications

Mode 2 and 3 are provided for multiprocessor communications. In Mode 2 if the received 9th data bit is not 1, the serial port interrupt is not activated. The way to use this feature in multiprocessor systems is described below.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address frame which identifies the target slave. An address frame will differ from a data frame in that the 9th data bit is 1 in an address frame and 0 in a data frame. No slave in Mode 2 will be interrupted by a data frame. An address frame, however, will interrupt all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave switches to Mode 3 to receive the coming data frames, while the slaves that were not addressed stay in Mode 2 and go on about their business.

10.0 I/O PORTS

There are five 8-bit I/O ports on the 8096. Some of these ports are input only, some are output only, some are bidirectional and some have alternate functions. In addition to these ports, the HSI/O unit can be used to provide extra I/O lines if the timer related features of these lines are not needed.

Input ports connect to the internal bus through an input buffer. Output ports connect through an output buffer to an internal register that hold the bits to be output. Bidirectional ports consist of an internal register, an input buffer, and an output buffer.

Port 0 is an input port which is also used as the analog input for the A to D converter. Port 1 is a quasi-bidirectional port. Port 2 contains three types of port lines: quasi-bidirectional, input and output. The input and output lines are shared with other functions in the 8096BH as shown in Table 4. Ports 3 and 4 are open-drain bidirectional ports which share their pins with the address/data bus.

Table 4. Port 2 Alternate Functions

Port	Function	Alternate Function	Controlled by
P2.0	Output	TXD (Serial Port Transmit)	IOC1.5
P2.1	Input	RXD (Serial Port Receive)	N/A
P2.2	Input	EXTINT (External Interrupt)	IOC1.1
P2.3	Input	T2CLK (Timer 2 Input)	IOC0.7
P2.4	Input	T2RST (Timer 2 Reset)	IOC0.5
P2.5	Output	PWM (Pulse-Width Modulation)	IOC1.0
P2.6	Quasi-Bidirectional		
P2.7	Quasi-Bidirectional		

Section 2 of the Hardware Design chapter contains additional information on the timing, drive capabilities, and input impedances of I/O pins.

10.1 Input Ports

Input ports and pins can only be read. There are no output drivers on these pins. The input leakage of these pins is in the microamp range. The specific values can be found in the data sheet for the device being considered.

In addition to acting as a digital input, each line of Port 0 can be selected to be the input of the A to D converter as discussed in Section 8. The pins on Port 0 are tested to have D.C. leakage of 3 microamps or less, as specified in the data sheet for the device being considered. The capacitance on these pins is approximately 5 pF and will instantaneously increase by around 5 pF when the pin is being sampled by the A to D converter.

The 8096BH samples the input to the A/D for 4 state times at the beginning of the conversion. *The 8X9X devices sample the A/D pin 10 times during a conversion.* Details on the A to D converter can be found in Section 8 of this chapter and in Section 3 of the Hardware Design chapter.

10.2 Quasi-Bidirectional Ports

Port 1, Port 2.6 and Port 2.7 are quasi-bidirectional ports. "Quasi-bidirectional" means that the port pin has a weak internal pullup that is always active and an internal pulldown which can be on to output a 0, or off to output a 1. If the internal pulldown is left off (by writing a 1 to the pin), the pin's logic level can be controlled by an external pulldown. If the external pulldown is on, it will input a 0 to the 8096BH, if it is off, a 1 will be input. From the user's point of view, the main difference between a quasi-bidirectional port and a standard input port is that the quasi-bidirectional port will source current if externally pulled low. It will also pull itself high if left unconnected.

In parallel with the weak internal pullup is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time. When this pullup is on the pin can typically source 30 milliamps to V_{SS}.

When the processor writes to the pins of a quasi-bidirectional port it actually writes into a register which in turn drives the port pin. When the processor reads these ports, it senses the status of the pin directly. If a port pin is to be used as an input then the software should write a one to its associated SFR bit, this will cause the low-impedance pull-down device to turn off and leave the pin pulled up with a relatively high impedance pullup device which can be easily driven down by the device driving the input.

If some pins of a port are to be used as inputs and some are to be used as outputs the programmer should be careful when writing to the port.

Particular care should be exercised when using XOR opcodes or any opcode which is a read-modify-write instruction. It is possible for a Quasi-Bidirectional Pin to be written as a one, but read back as a zero if an external device (i.e., a transistor base) is pulling the pin below V_{IH}. See the Hardware Design Chapter Section 2.2 for further details on using the Quasi-Bidirectional Ports.

10.3 Output Ports

Output pins include the bus control lines, the HSO lines, and some of Port 2. These pins can only be used as outputs as there are no input buffers connected to them. It is not possible to use immediate logical instructions such as XOR PORT2, #00111B to toggle these pins. The output currents on these ports is higher than that of the quasi-bidirectional ports.

10.4 Ports 3 and 4/AD0–15

These pins have two functions. They are either bidirectional ports with open-drain outputs or System Bus pins which the memory controller uses when it is accessing off-chip memory. If the $\overline{EA}$ line is low, the pins always act as the System Bus. Otherwise they act as bus pins only during a memory access. If these pins are being used as ports and bus pins, ones must be written to them prior to bus operations.

Accessing Port 3 and 4 as I/O is easily done from internal registers. Since the LD and ST instructions require the use of internal registers, it may be necessary to first move the port information into an internal location before utilizing the data. If the data is already internal, the LD is unnecessary. For instance, to write a word value to Port 3 and 4 . . .

```
LD intreg, portdata   ; register ← data
                      ; not needed if already internal

ST intreg, 1FFEH      ; register → Port 3 and 4
```

To read Port 3 and 4 requires that "ones" be written to the port registers to first setup the input port configuration circuit. Note that the ports are reset to this input condition, but if zeroes have been written to the port, then ones must be re-written to any pins which are to be used as inputs. Reading Port 3 and 4 from a previously written zero condition is as follows . . .

```
LD intregA, #0FFFFH   ; setup port change mode pattern

ST intregA, 1FFEH     ; register → Port 3 and 4
                      ; LD & ST not needed if previously
                      ; written as ones

LD intregB, 1FFEH     ; register ← Port 3 and 4
```

Note that while the format of the LD and ST instructions are similar, the source and destination directions change.

When acting as the system bus the pins have strong drivers to both V_{CC} and V_{SS}. These drivers are used whenever data is being output on the system bus and are not used when data is being output by Ports 3 and 4. Only the pins and input buffers are shared between the bus and the ports. The ports use different output buffers which are configured as open-drain, and require pullup resistors. (open-drain is the MOS version of open-collector.) The port pins and their system bus functions are shown in Table 5.

Table 5. P3,4/AD0-15 Pins

Port Pin	System Bus Function
P3.0	AD0
P3.1	AD1
P3.2	AD2
P3.3	AD3
P3.4	AD4
P3.5	AD5
P3.6	AD6
P3.7	AD7
P4.0	AD8
P4.1	AD9
P4.2	AD10
P4.3	AD11
P4.4	AD12
P4.5	AD13
P4.6	AD14
P4.7	AD15

11.0 STATUS AND CONTROL REGISTERS

There are two I/O Control registers, IOC0 and IOC1. IOC0 controls Timer 2 and the HSI lines. IOC1 controls some pin functions, interrupt sources and 2 HSO pins.

Whenever input lines are switched between two sources, or enabled, it is possible to generate transitions on these lines. This could cause problems with respect to edge sensitive lines such as the HSI lines, Interrupt line, and Timer 2 control lines.

11.1 I/O Control Register 0 (IOC0)

IOC0 is located at 0015H. The four HSI lines can be enabled or disabled to the HSI unit by setting or clearing bits in IOC0. Timer 2 functions including clock and reset sources are also determined by IOC0. The control bit locations are shown in Figure 38.

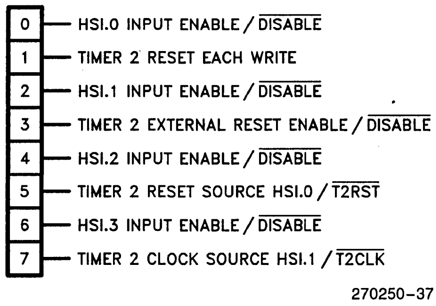

0 HSI.0 INPUT ENABLE / $\overline{\text{DISABLE}}$
1 TIMER 2 RESET EACH WRITE
2 HSI.1 INPUT ENABLE / $\overline{\text{DISABLE}}$
3 TIMER 2 EXTERNAL RESET ENABLE / $\overline{\text{DISABLE}}$
4 HSI.2 INPUT ENABLE / $\overline{\text{DISABLE}}$
5 TIMER 2 RESET SOURCE HSI.0 / $\overline{\text{T2RST}}$
6 HSI.3 INPUT ENABLE / $\overline{\text{DISABLE}}$
7 TIMER 2 CLOCK SOURCE HSI.1 / $\overline{\text{T2CLK}}$
270250-37

Figure 38. I/O Control Register 0 (IOC0)

11.2 I/O Control Register 1 (IOC1)

IOC1 is used to select some pin functions and enable or disable some interrupt sources. Its location is 0016H. Port pin P2.5 can be selected to be the PWM output instead of a standard output. The external interrupt source can be selected to be either EXTINT (same pin as P2.2) or Analog Channel 7 (ACH7, same pin as P0.7). Timer 1 and Timer 2 overflow interrupts can be individually enabled or disabled. The HSI interrupt can be selected to activate either when there is 1 FIFO entry or 7. Port pin P2.0 can be selected to be the TXD output. HSO.4 and HSO.5 can be enabled or disabled to the HSO unit. More information on interrupts is available in Section 4. The positions of the IOC1 control bits are shown in Figure 39.

11.3 I/O Status Register 0 (IOS0)

There are two I/O Status registers, IOS0 and IOS1. IOS0, located at 0015H, holds the current status of the HSO lines and CAM. The status bits of IOS0 are shown in Figure 40.

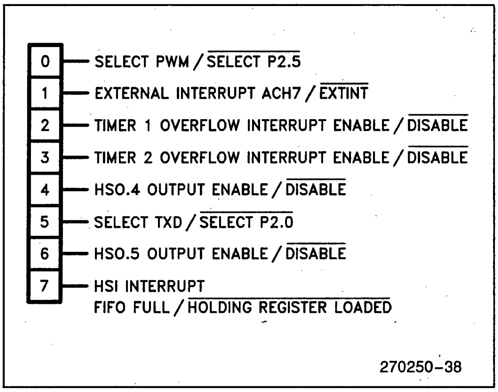

0 — SELECT PWM / SELECT P2.5
1 — EXTERNAL INTERRUPT ACH7 / EXTINT
2 — TIMER 1 OVERFLOW INTERRUPT ENABLE / DISABLE
3 — TIMER 2 OVERFLOW INTERRUPT ENABLE / DISABLE
4 — HSO.4 OUTPUT ENABLE / DISABLE
5 — SELECT TXD / SELECT P2.0
6 — HSO.5 OUTPUT ENABLE / DISABLE
7 — HSI INTERRUPT FIFO FULL / HOLDING REGISTER LOADED
270250-38

Figure 39. I/O Control Register 1 (IOC1)

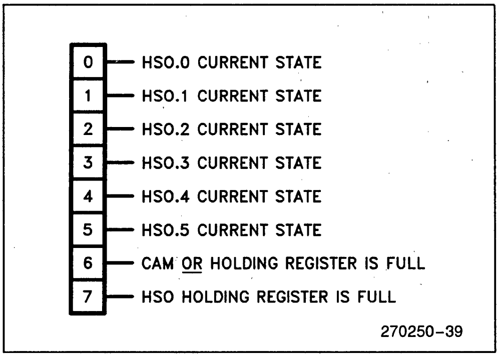


Figure 40. I/O Status Register 0 (IOS0)

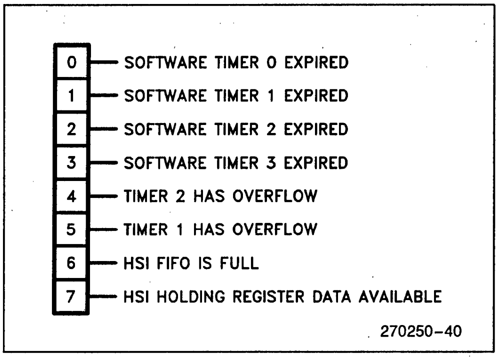

0 — SOFTWARE TIMER 0 EXPIRED
1 — SOFTWARE TIMER 1 EXPIRED
2 — SOFTWARE TIMER 2 EXPIRED
3 — SOFTWARE TIMER 3 EXPIRED
4 — TIMER 2 HAS OVERFLOW
5 — TIMER 1 HAS OVERFLOW
6 — HSI FIFO IS FULL
7 — HSI HOLDING REGISTER DATA AVAILABLE
270250-40

Figure 41. HSIO Status Register 1 (IOS1)

11.4 I/O Status Register 1 (IOS1)

IOS1 is located at 016H. It contains status bits for the timers and the HSI/O. The positions of these bits are shown in Figure 41.

Whenever the processor reads this register all of the time-related flags (bits 5 through 0) are cleared. This applies not only to explicit reads such as:

```
LDB    AL,IOS1
```

but also to implicit reads such as:

```
JB     IOS1.3,somewhere_else
```

which jumps to somewhere_else if bit 3 of IOS1 is set. In most cases this situation can best be handled by having a byte in the register file which is used to maintain an image of lower five bits of the register. Any time a hardware timer interrupt or a HSO software timer interrupt occurs the byte can be updated:

```
ORB    IOS1_image,IOS1
```

leaving IOS1_image containing all the flags that were set before plus all the new flags that were read and cleared from IOS1. Any other routine which needs to sample the flags can safely check IOS1_image. Note that if these routines need to clear the flags that they have acted on, then the modification of IOS1_image must be done from inside a critical region (see Section 4.4).

12.0 WATCHDOG TIMER

The WatchDog Timer (WDT) provides a means to recover gracefully from a software upset. When the watchdog is enabled it will initiate a hardware reset unless the software clears it every 64K state times.

The WDT is implemented as an 8-bit timer with an 8-bit prescaler. The prescaler is not synchronized, so the timer will overflow between 65280 and 65535 state times after being reset. When the timer overflows it pulls down the $\overline{\text{RESET}}$ pin for at least two state times, resetting the 8096BH and any other devices tied to the $\overline{\text{RESET}}$ line. If a large capacitor is connected to the line, the pin may take a long time to go low. This will effect the length of time the pin is low and the voltage on the pin when it is finished falling. Section 1.4 of the Hardware Design chapter contains more information about reset hardware connections.

The WDT is enabled the first time it is cleared. Once it is enabled, it can only be disabled by resetting the 8096BH. The internal bit which controls the watchdog can typically maintain its state through power glitches as low as V_{SS} and as high as 7.0V for up to one millisecond.

The 8X9X devices do not have the extra glitch protection on the WDT enable bit.

Enabling and clearing the WDT is done by writing a "01EH" followed by a "0E1H" to the WDT register at location 0AH. This double write is used to help prevent accidental clearing of the timer.

12.1 Software Protection Hints

Glitches and noise on the PC board can cause software upsets, typically by changing either memory locations or the program counter. These changes can be internal to the chip or be caused by bad data returning to the chip.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The software can be designed so that the watchdog times out if the program does not progress properly. The watchdog will also time-out if the software error was due to ESD (Electrostatic Discharge) or other hardware related problems. This prevents the controller from having a malfunction for longer than 16 milliseconds if a 12 MHz oscillator is used.

When using the WDT to protect software it is desirable to reset it from only one place in code. This will lessen the chance that an undesired WDT reset will occur. The section of code that resets the WDT should monitor the other code sections for proper operation. This can be done by checking variables to make sure they are within reasonable values. Simply using a software timer to reset the WDT every 15 milliseconds will not provide much protection against minor problems.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed undesired results will occur. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 8096 instruction has 7 bytes) and a RST or jump to error routine instruction. Since RST is a one-byte instruction, the NOPs are not needed if RSTs are used instead of jumps to an error routine. This will help to ensure a speedy recovery should the processor have a glitch in the program flow. Since RST instruction has an opcode of 0FFH, pulling the data lines high with resistors will cause an RST to be executed if unimplemented memory is addressed.

12.2 Disabling The Watchdog

The watchdog should be disabled by software not initializing it. If this is not possible, such as during program development, the watchdog can be disabled by holding the $\overline{\text{RESET}}$ pin at 2.0V to 2.5V. Voltages over 2.5V on the pin could quickly damage the part. Even at 2.5V, using this technique for other than debugging purposes is not recommended, as it may effect long term reliability. It is further recommended that any part used in this way for more than several seconds, not be used in production versions of products. Section 1.6 of the Hardware Design chapter has more information on disabling the Watchdog Timer.

13.0 RESET

13.1 Reset Signal

As with all processors, the 8096BH must be reset each time the power is turned on. This is done by holding the $\overline{\text{RESET}}$ pin low for at least 2 state times after the power supply is within tolerance and the oscillator has stabilized.

On 8X9X devices the $\overline{\text{RESET}}$ pin must be held low long enough for the power supply, oscillator and back-bias generator to stabilize. Typically, the back-bias generator requires one millisecond to stabilize.

After the $\overline{\text{RESET}}$ pin is brought high, a ten state reset sequence is executed. During this time, the Chip Configuration Byte (CCB) is read from location 2018H and written to the 8096BH Chip Configuration Register (CCR). If the voltage on the $\overline{\text{EA}}$ pin selects the internal/external execution mode the CCB is read from internal ROM/EPROM. If the voltage on the $\overline{\text{EA}}$ pin selects the external execution only mode the CCB is read from external memory.

The 8096BH can be reset using a capacitor, 1-shot, or any other method capable of providing a pulse of at least 2 state times longer than required for V_{CC} and the oscillator to stabilize.

For best functionality, it is suggested that the reset pin be pulled low with an open collector device. In this way, several reset sources can be wire ORed together. Remember, the $\overline{\text{RESET}}$ pin itself can be a reset source when the RST instruction is executed or when the Watchdog Timer overflows. Details of hardware suggestions for reset can be found in Section 1.4 of the Hardware Design chapter.

13.2 Reset Status

The I/O lines and control lines of the 8096BH will be in their reset state within 2 state times after reset is low, with V_{CC} and the oscillator stabilized. Prior to that time, the status of the I/O lines is indeterminate. After the 10 state time reset sequence, the Special Function Registers will be set as follows:

Register	Reset Value
Port 1	11111111B
Port 2	110XXXX1B
Port 3	11111111B
Port 4	11111111B
PWM Control	00H
Serial Port (Transmit)	undefined
Serial Port (Receive)	undefined
Baud Rate Register	undefined
Serial Port Control	XXXX0XXXB
Serial Port Status	X00XXXXXB
A/D Command	undefined
A/D Result	undefined
Interrupt Pending	undefined
Interrupt Mask	00000000B
Timer 1	0000H
Timer 2	0000H
Watchdog Timer	0000H
HSI Mode	11111111B
HSI Status	undefined
IOS0	00000000B
IOS1	00000000B
IOC0	X0X0X0X0B
IOC1	X0X0XXX1B
HSI FIFO	empty
HSO CAM	empty
HSO lines	000000B
PSW	0000H
Stack Pointer	undefined
Program Counter	2080H

Figure 42. Register Reset Status

Other conditions following a reset are:

Pin	Reset Value
$\overline{\text{RD}}$	high
$\overline{\text{WR}}/\overline{\text{WRL}}$	high
ALE/$\overline{\text{ADV}}$	high
$\overline{\text{BHE}}/\overline{\text{WRH}}$	low
INST	high
ALE (8X9X)	low

Figure 43. Bus Control Pins Reset Status

It is important to note that the Stack Pointer and Interrupt Pending Register are undefined, and need to be initialized in software. The Interrupts are disabled by both the mask register and PSW.9 after a reset.

13.3 Reset Sync Mode

The $\overline{\text{RESET}}$ line can be used to start the 8096BH at an exact state time to provide for synchronization of test equipment and multiple chip systems. $\overline{\text{RESET}}$ is active low. To synchronize parts, $\overline{\text{RESET}}$ is brought high on the rising edge of XTAL1. Complete details on synchronizing parts can be found in Section 1.5 of the Hardware Design chapter.

It is very possible that parts which start in sync may not stay that way. The best example of this would be when a "jump on I/O bit" is being used to hold the processor in a loop. If the line changes during the time it is being tested, one processor may see it as a one, while the other sees it as a zero. The result is that one processor will do an extra loop, thus putting it several states out of sync with the other.

14.0 QUICK REFERENCE

14.1 Pin Description

On the 48-pin parts the following pins are not bonded out: Port1, Port0 (Analog In) bits 0–3, T2CLK (P2.3), T2RST (P2.4), P2.6, P2.7, CLKOUT, INST, NMI, BUSWIDTH ($\overline{\text{TEST}}$ on 8X9X devices).

PIN DESCRIPTIONS

Symbol	Name and Function
V_{CC}	Main supply voltage (5V).
V_{SS}	Digital circuit ground (0V).
V_{PD}	RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e. V_{CC} drops to zero), if $\overline{\text{RESET}}$ is activated before V_{CC} drops below spec and V_{PD} continues to be held within spec., the top 16 bytes in the Register File will retain their contents. $\overline{\text{RESET}}$ must be held low during the Power Down and should not be brought high until V_{CC} is within spec and the oscillator has stabilized. See Section 2.3.
V_{REF}	Reference voltage for the A/D converter (5V). V_{REF} is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0. See Section 8.
ANGND	Reference ground for the A/D converter. Should be held at nominally the same potential as V_{SS}. See Section 8.
V_{PP} V_{BB}(8X9X)	Programming voltage for the EPROM parts. It should be +12.75V when programming and will float to 5V otherwise. It should not be above 5.5V on other than EPROM parts. This pin is V_{BB} on 8X9X parts. Systems that have this pin connected to ANGND through a capacitance (required on 8X9X parts) do not need to change.
XTAL1	Input of the oscillator inverter and of the internal clock generator. See Section 1.5.
XTAL2	Output of the oscillator inverter. See Section 1.5.
CLKOUT	Output of the internal clock generator. The frequency of CLKOUT is ⅓ the oscillator frequency. It has a 33% duty cycle. See Section 1.5
$\overline{\text{RESET}}$	Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to location 2080H is executed. Input high for normal operation. $\overline{\text{RESET}}$ has an internal pullup. (The read from 2018H is not done on 8X9X parts). See Section 13.
BUSWIDTH $\overline{\text{TEST}}$(8X9X)	Input for buswidth selection. If CCR bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is a 1, a 16-bit bus cycle occurs. If BUSWIDTH is a 0 an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. This pin is the $\overline{\text{TEST}}$ pin on 8X9X parts. Systems with $\overline{\text{TEST}}$ tied to V_{CC} do not need to change. If this pin is left unconnected, it will rise to V_{CC}. See Section 2.7.
NMI	A positive transition causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.
INST	Output high during an external memory read indicates the read is an instruction fetch. INST is valid throughout the bus cycle.
$\overline{\text{EA}}$	Input for memory select (External Access). $\overline{\text{EA}}$ equal to a TTL-high causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM/EPROM. $\overline{\text{EA}}$ equal to a TTL-low causes accesses to these locations to be directed to off-chip memory. $\overline{\text{EA}}$ = +12.5V causes execution to begin in the Programming mode on EPROM parts. $\overline{\text{EA}}$ has an internal pulldown, so it goes to 0 unless driven otherwise.
ALE/$\overline{\text{ADV}}$	Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is $\overline{\text{ADV}}$, it goes inactive high at the end of the bus cycle. $\overline{\text{ADV}}$ can be used as a chip select for external memory. ALE/$\overline{\text{ADV}}$ is activated only during external memory accesses. (The $\overline{\text{ADV}}$ function is not available on 8X9X parts). See Section 2.7.
$\overline{\text{RD}}$	Read signal output to external memory. $\overline{\text{RD}}$ is activated only during external memory reads.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
$\overline{\text{WR}}/\overline{\text{WRL}}$	Write and Write Low output to external memory, as selected by the CCR. $\overline{\text{WR}}$ will go low for every external write, while $\overline{\text{WRL}}$ will go low only for external writes where an even byte is being written. $\overline{\text{WR}}/\overline{\text{WRL}}$ is activated only during external memory writes. (The $\overline{\text{WRL}}$ function is not available on 8X9X parts). See Section 2.7.
$\overline{\text{BHE}}/\overline{\text{WRH}}$	Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{\text{BHE}}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{\text{BHE}}$ = 1), to the high byte only (A0 = 1, BHE # = 0), or both bytes (A0 = 0, $\overline{\text{BHE}}$ = 0). If the $\overline{\text{WRH}}$ function is selected, the pin will go low if the bus cycle is writing to an odd memory location. (The $\overline{\text{WRH}}$ function is not available on 8X9X parts). See Section 2.7.
READY	Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the Memory Controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. The bus cycle can be lengthened by up to 1 μs. When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle held not ready is available through configuration of CCR. READY has a weak internal pullup, so it goes to 1 unless externally pulled low. (Internal control of the number of wait states is not available on 8X9X parts). See Section 2.7.
HSI	Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as inputs by EPROM parts in Programming mode. See Section 6.
HSO	Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit. See Section 7.
Port 0	8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins are also a mode input to EPROM parts in the Programming mode. See Section 10.
Port 1	8-bit quasi-bidirectional I/O port. See Section 10.
Port 2	8-bit multi-functional port. Six of its pins are shared with other functions in the 8096BH, the remaining 2 are quasi-bidirectional. These pins are also used to input and output control signals on EPROM parts in Programming Mode. See Section 10.
Ports 3 and 4	8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups. Ports 3 and 4 are also used as a command, address and data path by EPROM parts operating in the programming mode. See Sections 2.7 and 10.

14.2 Pin List

The following is a list of pins in alphabetical order. Where a pin has two names it has been listed under both names, except for the system bus pins, AD0–AD15, which are listed under Port 3 and Port 4.

Name	68-Pin PLCC	68-Pin PGA	48-Pin DIP
ACH0/P0.0	6	4	—
ACH1/P0.1	5	5	—
ACH2/P0.2	7	3	—
ACH3/P0.3	4	6	—
ACH4/P0.4/MOD.0	11	67	43
ACH5/P0.5/MOD.1	10	68	42
ACH6/P0.6/MOD.2	8	2	40
ACH7/P0.7/MOD.3	9	1	41
ALE/$\overline{\text{ADV}}$	62	16	34
ANGND	12	66	44
$\overline{\text{BHE}}$/$\overline{\text{WRH}}$	41	37	15
BUSWIDTH ($\overline{\text{TEST}}$)	64	14	—
CLKOUT	65	13	—
$\overline{\text{EA}}$	2	8	39
EXTINT/P2.2/$\overline{\text{PROG}}$	15	63	47
HSI.0	24	54	3
HSI.1	25	53	4
HSI.2/HSO.4	26	52	5
HSI.3/HSO.5	27	51	6
HSO.0	28	50	7
HSO.1	29	49	8
HSO.2	34	44	9
HSO.3	35	43	10
HSO.4/HSI.2	26	52	5
HSO.5/HSI.3	27	51	6
INST	63	15	—
NMI	3	7	—
PWM/P2.5/$\overline{\text{PDO}}$	39	39	13
PALE/P2.1/RXD	17	61	1
$\overline{\text{PROG}}$/P2.2/EXTNT	15	63	47
PVER/P2.0/TXD	18	60	2
P0.0/ACH0	6	4	—
P0.1/ACH1	5	5	—
P0.2/ACH2	7	3	—
P0.3/ACH3	4	6	—
P0.4/ACH4/MOD.0	11	67	43
PO.5/ACH5/MOD.1	10	68	42
PO.6/ACH6/MOD.2	8	2	40
PO.7/ACH7/MOD.3	9	1	41
P1.0	19	59	—
P1.1	20	58	—
P1.2	21	57	—
P1.3	22	56	—
P1.4	23	55	—
P1.5	30	48	—

Name	68-Pin PLCC	68-Pin PGA	48-Pin DIP
P1.6	31	47	—
P1.7	32	46	—
P2.0/TXD/NER	18	60	2
P2.1/RXD/PALE	17	61	1
P2.2/EXTINT	15	63	47
P2.3/T2CLK	44	34	—
P2.4/T2RST	42	36	—
P2.5/PWM/$\overline{\text{PDO}}$	39	39	13
P2.6	33	45	—
P2.7	38	40	—
P3.0/AD0	60	18	32
P3.1/AD1	59	19	31
P3.2/AD2	58	20	30
P3.3/AD3	57	21	29
P3.4/AD4	56	22	28
P3.5/AD5	55	23	27
P3.6/AD6	54	24	26
P3.7/AD7	53	25	25
P4.0/AD8	52	26	24
P4.1/AD9	51	27	23
P4.2/AD10	50	28	22
P4.3/AD11	49	29	21
P4.4/AD12	48	30	20
P4.5/AD13	47	31	19
P4.6/AD14	46	32	18
P4.7/AD15	45	33	17
$\overline{\text{RD}}$	61	17	33
READY	43	35	16
$\overline{\text{RESET}}$	16	62	48
RXD/P2.1	17	61	1
SALE/PVER/P2.0	18	60	2
$\overline{\text{SPROG}}$/$\overline{\text{PDO}}$/P2.5	39	39	13
TXD/P2.0	18	60	2
T2CLK/P2.3	44	34	—
T2RST/P2.4	42	36	—
VBB	37	41	12
VCC	1	9	38
VPD	14	64	46
VREF	13	65	45
VSS	68	10	11
VSS	36	42	37
$\overline{\text{WR}}$/$\overline{\text{WRL}}$	40	38	14
$\overline{\text{WRH}}$/$\overline{\text{BHE}}$	41	37	15
XTAL1	67	11	36
XTAL2	66	12	35

The Following pins are not bonded out in the 48-pin package:

P1.0 through P1.7, P0.0 through P0.3, P2.3, P2.4, P2.6, P2.7 CLKOUT, INST, NMI, $\overline{\text{TEST}}$, T2CLK (P2.3), T2RST (P2.4).

14.3 Packaging

The MCS-96 products are available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM or EPROM. The MCS-96 numbering system is shown below. Section 14.4 shows the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin versions come in a Plastic Leaded Chip Carrier (PLCC), a Pin Grid Array (PGA) or a Type "B" Leadless Chip Carrier.

The MCS®-96 Family Nomenclature

		Without A/D	With A/D
ROMless 809XBH	**48 Pin**		C8095BH - Ceramic DIP P8095BH - Plastic DIP
	68 Pin	A8096BH - Ceramic PGA N8096BH - PLCC	A8097BH - Ceramic PGA N8097BH - PLCC
ROM 839XBH	**48 Pin**		C8395BH - Ceramic DIP P8395BH - Plastic DIP
	68 Pin	A8396BH - Ceramic PGA N8396BH - PLCC	A8397BH - Ceramic PGA N8397BH - PLCC
EPROM 879XBH	**48 Pin**		C8795BH - Ceramic DIP
	68 Pin	A8796BH - Ceramic PGA R8796BH - Ceramic LCC	A8797BH - Ceramic PGA R8797BH - Ceramic LCC
ROMless 8096	**48 Pin**		C8095-90 - Ceramic DIP P8095-90 - Plastic DIP
	68 Pin	A8096-90 - Ceramic PGA N8096-90 - PLCC	A8097-90 - Ceramic PGA N8097-90 - PLCC
ROM 8396	**48 Pin**		C8395-90 - Ceramic DIP P8395-90 - Plastic DIP
	68 Pin	A8396-90 - Ceramic PGA N8396-90 - PLCC	A8397-90 - Ceramic PGA N8397-90 - PLCC

Transistor Count

Device Type	#MOS Gates
839X/879X	120,000
809X	50,000

MTBF Calculations*

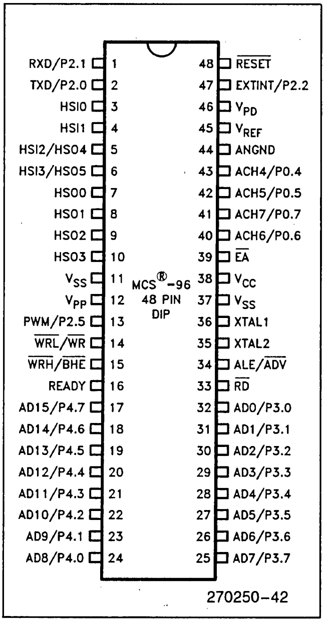
3.8 × 10^7 Device Hours @ 55°C
1.7 × 10^7 Device Hours @ 70°C

*MTBF data was obtained through calculations based upon the actual average junction temperatures under stress at 55°C and 70°C ambient.

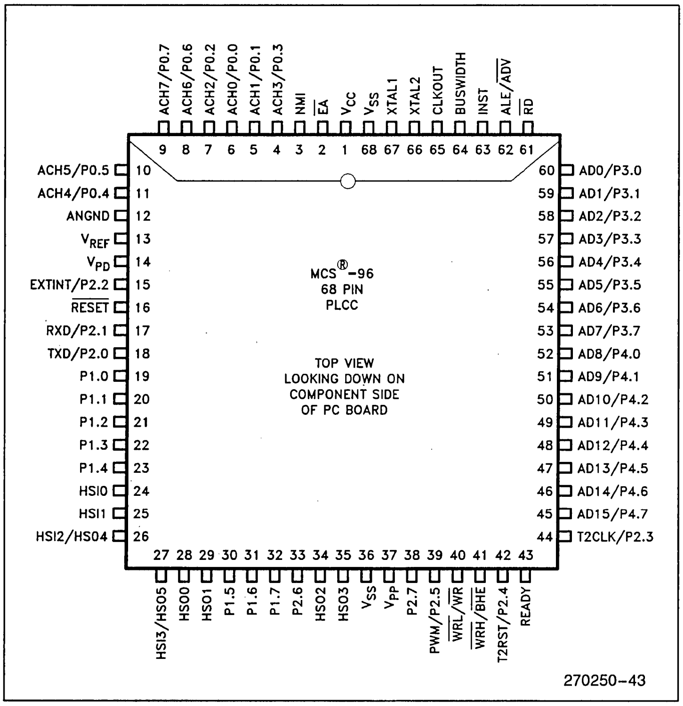
Thermal Characteristics

T_{CASE}		Package Type	θJa	θJc
COMM'L	EXPRESS			
85°C	100°C	PGA	35°C/W	10°C/W
85°C	100°C	PLCC	37°C/W	10°C/W
		LCC	28°C/W	—
		Plastic DIP	38°C/W	—
79.75°C	94.75°C	Ceramic DIP	26°C/W	6.5°C/W

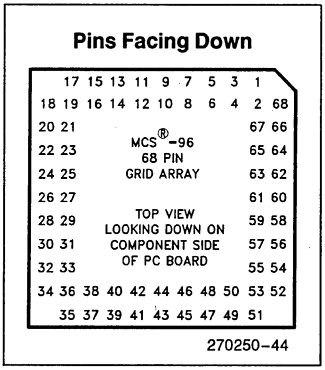
14.4 Package Diagrams



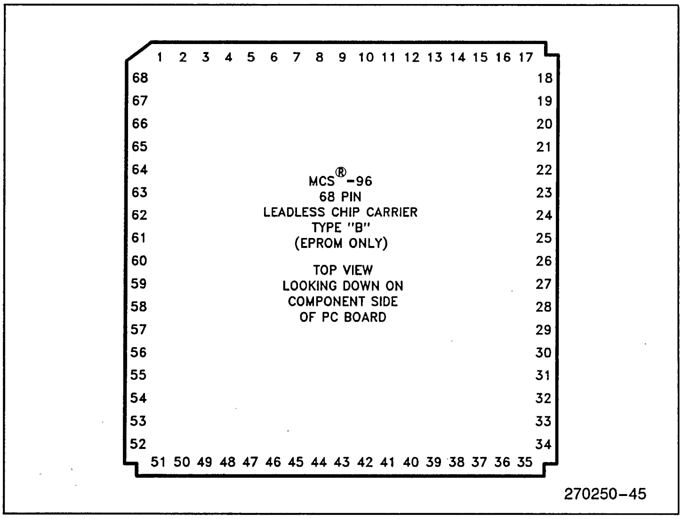
48-Pin Package



68-Pin Package (PLCC - Top View)



68-Pin Package (Pin Grid Array - Top View)

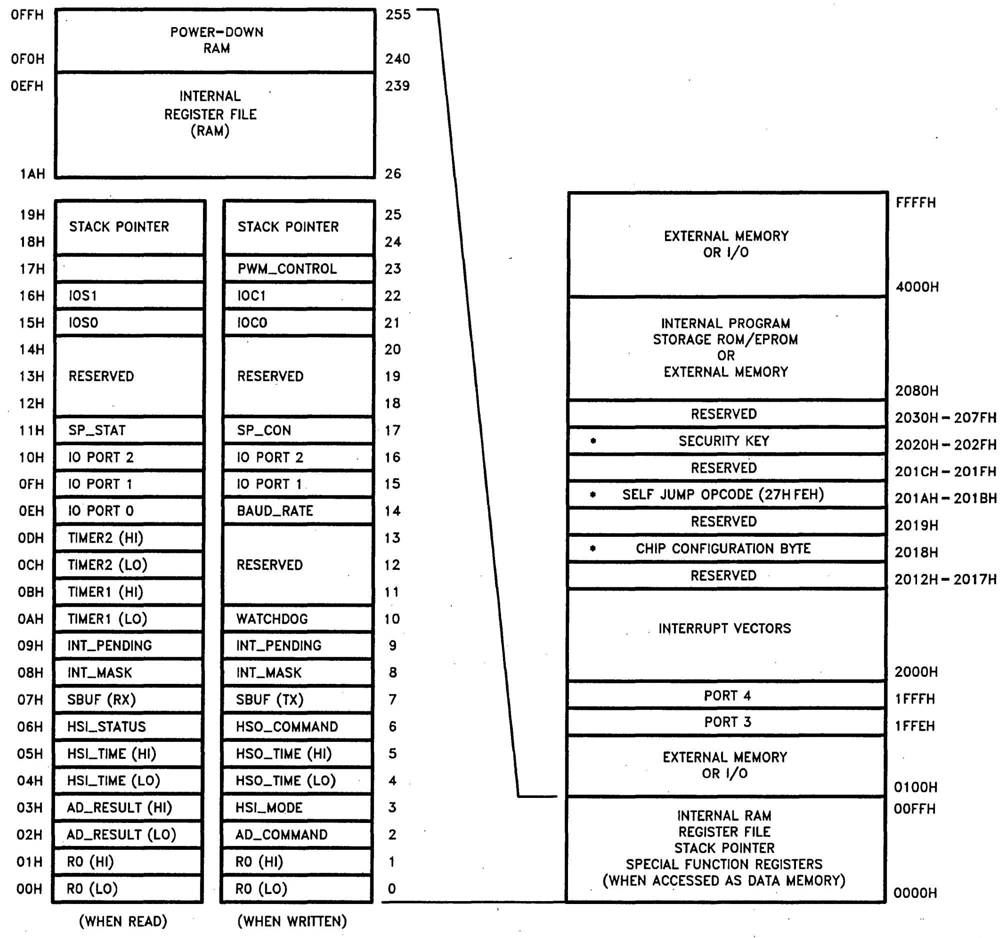


68-Pin Package (LCC - Top View)

14.5 PGA, PLCC and LCC Function Pinouts

PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description
1	9	ACH7/P0.7/PMOD.3	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6/PMOD.2	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{\text{EA}}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	VCC	32	46	AD14/P4.6	55	23	P1.4
10	68	VSS	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	BUSWIDTH	37	41	$\overline{\text{BHE}}$/$\overline{\text{WRH}}$	60	18	TXD/P2.0/PVER/SALE
15	63	INST	38	40	$\overline{\text{WR}}$/$\overline{\text{WRL}}$	61	17	RXD/P2.1/PALE
16	62	ALE/$\overline{\text{ADV}}$	39	39	PWM/P2.5/$\overline{\text{PDO}}$/$\overline{\text{SPROG}}$	62	16	$\overline{\text{RESET}}$
17	61	$\overline{\text{RD}}$	40	38	P2.7	63	15	EXTINT/P2.2/$\overline{\text{PROG}}$
18	60	AD0/P3.0	41	37	VPP	64	14	VPD
19	59	AD1/P3.1	42	36	VSS	65	13	VREF
20	58	AD2/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4/PMOD.0
22	56	AD4/P3.4	45	33	P2.6	68	10	ACH5/P0.5/PMOD.1
23	55	AD5/P3.5	46	32	P1.7			

14.6 Memory Map



0FFH	POWER-DOWN RAM		255
0F0H			240
0EFH	INTERNAL REGISTER FILE (RAM)		239
1AH			26
19H	STACK POINTER	STACK POINTER	25
18H			24
17H		PWM_CONTROL	23
16H	IOS1	IOC1	22
15H	IOS0	IOC0	21
14H	RESERVED	RESERVED	20
13H			19
12H			18
11H	SP_STAT	SP_CON	17
10H	IO PORT 2	IO PORT 2	16
0FH	IO PORT 1	IO PORT 1	15
0EH	IO PORT 0	BAUD_RATE	14
0DH	TIMER2 (HI)	RESERVED	13
0CH	TIMER2 (LO)		12
0BH	TIMER1 (HI)		11
0AH	TIMER1 (LO)	WATCHDOG	10
09H	INT_PENDING	INT_PENDING	9
08H	INT_MASK	INT_MASK	8
07H	SBUF (RX)	SBUF (TX)	7
06H	HSI_STATUS	HSO_COMMAND	6
05H	HSI_TIME (HI)	HSO_TIME (HI)	5
04H	HSI_TIME (LO)	HSO_TIME (LO)	4
03H	AD_RESULT (HI)	HSI_MODE	3
02H	AD_RESULT (LO)	AD_COMMAND	2
01H	R0 (HI)	R0 (HI)	1
00H	R0 (LO)	R0 (LO)	0
	(WHEN READ)	(WHEN WRITTEN)	

EXTERNAL MEMORY OR I/O	FFFFH 4000H
INTERNAL PROGRAM STORAGE ROM/EPROM OR EXTERNAL MEMORY	2080H
RESERVED	2030H – 207FH
* SECURITY KEY	2020H – 202FH
RESERVED	201CH – 201FH
* SELF JUMP OPCODE (27H FEH)	201AH – 201BH
RESERVED	2019H
* CHIP CONFIGURATION BYTE	2018H
RESERVED	2012H – 2017H
INTERRUPT VECTORS	2000H
PORT 4	1FFFH
PORT 3	1FFEH
EXTERNAL MEMORY OR I/O	0100H
INTERNAL RAM REGISTER FILE STACK POINTER SPECIAL FUNCTION REGISTERS (WHEN ACCESSED AS DATA MEMORY)	00FFH 0000H

270250–5

NOTE:
*Registers marked by an asterisk are not present on 8X9X devices

14.7 Instruction Summary

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	✓	✓	✓	✓	↑	—	
ADD/ADDB	3	D ← B + A	✓	✓	✓	✓	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	✓	✓	✓	↑	—	
SUB/SUBB	2	D ← D − A	✓	✓	✓	✓	↑	—	
SUB/SUBB	3	D ← B − A	✓	✓	✓	✓	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	✓	✓	✓	↑	—	
CMP/CMPB	2	D − A	✓	✓	✓	✓	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	✓	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	✓	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	
AND/ANDB	2	D ← D and A	✓	✓	0	0	—	—	
AND/ANDB	3	D ← B and A	✓	✓	0	0	—	—	
OR/ORB	2	D ← D or A	✓	✓	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	✓	✓	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ✓	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR [indirect]	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 − D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign(D)	✓	✓	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — — lsb ← 0	✓	?	✓	✓	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — — lsb → C	✓	?	✓	0	—	✓	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — — lsb → C	✓	✓	✓	0	—	✓	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left shift till msb = 1; D ← shift count	✓	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.

5. Offset is a 2's complement number.

6. Specified bit is one of the 2048 bits in the register file.

7. The "L" (Long) suffix indicates double-word operation.

8. Initiates a Reset by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.

9. The assembler will not accept this mnemonic.

14.8 Opcode and State Time Listing

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⊛					INDEXED⊛				
								NORMAL			AUTO-INC.		SHORT			LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES⑧	BYTES	STATE① TIMES⑧
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29

270250–46

NOTES:

*Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any Indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

⑧ State times shown for 16-bit bus.

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⊛					INDEXED⊛				
								NORMAL			AUTO-INC.		SHORT			LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES⑧	BYTES	STATE① TIMES⑧
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	——	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	——	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	——	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	——	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	**OPCODE**	**BYTES**	**STATES**	**MNEMONIC**	**OPCODE**	**BYTES**	**STATES**
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
BR[ ]	E3	2	8	RET	F0	1	12/16⑤
				TRAP③	F7	1	21/24

270250–47

NOTES:

① Number of state times shown for internal/external operands.
③ The assembler does not accept this mnemonic.
④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
⑤ State times for stack located internal/external.
⑧ State times shown for 16-bit bus.

CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.[8]

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.[8]

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

MNEMONIC	OPCODE	BYTES	STATE TIMES
DJNZ	EO	3	5/9 STATE TIME (NOT TAKEN/TAKEN)[8]

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES[8]	MNEMONIC	OPCODE	BYTES	STATES[8]
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR MNEMONIC	WORD		INSTR MNEMONIC	BYTE		INSTR MNEMONIC	DBL WD		STATE TIMES[8]
	OP	B		OP	B		OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT[7]
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT[7]
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT[7]

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES[8]	MNEMONIC	OPCODE	BYTES	STATES[8]
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST[6]	FF	1	166	SKIP	00	2	4

NORMALIZE

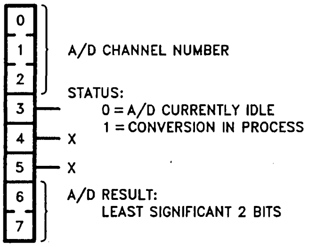
MNEMONIC	OPCODE	BYTES	STATE TIMES
NORML	0F	3	11 + 1 PER SHIFT

NOTES:

6. This instruction takes 2 states to pull $\overline{\text{RESET}}$ low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H. If a capacitor is tied to $\overline{\text{RESET}}$, the pin may take longer to go low and may never reach the V_{OL} specification.
7. Execution will take at least 8 states, even for 0 shift.
8. State times shown for 16-bit bus.

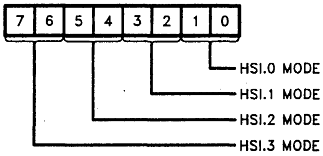
14.9 SFR Summary

A/D Result LO (02H)



270250–48

HSI_Mode (03H)

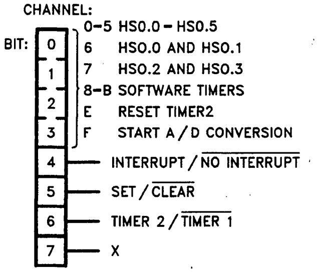


WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00	8 POSITIVE TRANSITIONS
01	EACH POSITIVE TRANSITION
10	EACH NEGATIVE TRANSITION
11	EVERY TRANSITION (POSITIVE AND NEGATIVE)

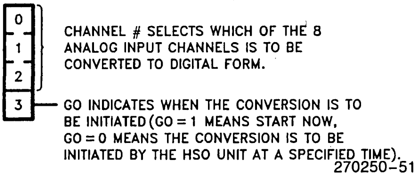
270250–49

HSO Command (06H)



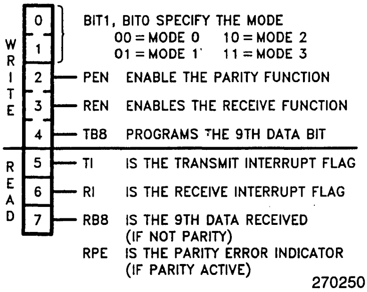
270250–50

A/D Command (02H)


0
1
2
3
CHANNEL # SELECTS WHICH OF THE 8 ANALOG INPUT CHANNELS IS TO BE CONVERTED TO DIGITAL FORM.
GO INDICATES WHEN THE CONVERSION IS TO BE INITIATED (GO = 1 MEANS START NOW, GO = 0 MEANS THE CONVERSION IS TO BE INITIATED BY THE HSO UNIT AT A SPECIFIED TIME).

270250–51

SPCON/SPSTAT (11H)


WRITE
READ
0
1
2
3
4
5
6
7
BIT1, BIT0 SPECIFY THE MODE
00 = MODE 0 10 = MODE 2
01 = MODE 1 11 = MODE 3
PEN ENABLE THE PARITY FUNCTION
REN ENABLES THE RECEIVE FUNCTION
TB8 PROGRAMS THE 9TH DATA BIT
TI IS THE TRANSMIT INTERRUPT FLAG
RI IS THE RECEIVE INTERRUPT FLAG
RB8 IS THE 9TH DATA RECEIVED (IF NOT PARITY)
RPE IS THE PARITY ERROR INDICATOR (IF PARITY ACTIVE)

270250–52

Baud Rate Calculations

Using XTAL1:

Mode 0: $\text{Baud Rate} = \dfrac{\text{XTAL1 frequency}}{4^*(B + 1)}$; $B \neq 0$

Others: $\text{Baud Rate} = \dfrac{\text{XTAL1 frequency}}{64^*(B + 1)}$

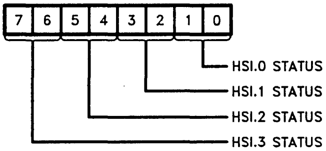
Using T2CLK:

Mode 0: $\text{Baud Rate} = \dfrac{\text{T2CLK frequency}}{B}$; $B \neq 0$

Others: $\text{Baud Rate} = \dfrac{\text{T2CLK frequency}}{16^*B}$; $B \neq 0$

Note that B cannot equal 0, except when using XTAL1 in other than Mode 0.

HSI_Status (06H)



WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

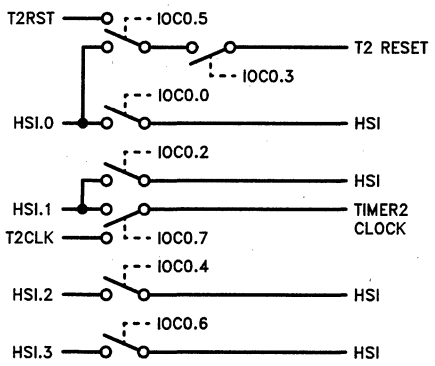
270250–53

IOC0 (15H)

0	HSI.0 INPUT ENABLE / $\overline{\text{DISABLE}}$
1	TIMER 2 RESET EACH WRITE
2	HSI.1 INPUT ENABLE / $\overline{\text{DISABLE}}$
3	TIMER 2 EXTERNAL RESET ENABLE / $\overline{\text{DISABLE}}$
4	HSI.2 INPUT ENABLE / $\overline{\text{DISABLE}}$
5	TIMER 2 RESET SOURCE HSI.0 / $\overline{\text{T2RST}}$
6	HSI.3 INPUT ENABLE / $\overline{\text{DISABLE}}$
7	TIMER 2 CLOCK SOURCE HSI.1 / $\overline{\text{T2CLK}}$

270250-54

IOC0 (15H)



270250-55

IOS0 (15H)

0	HSO.0 CURRENT STATE
1	HSO.1 CURRENT STATE
2	HSO.2 CURRENT STATE
3	HSO.3 CURRENT STATE
4	HSO.4 CURRENT STATE
5	HSO.5 CURRENT STATE
6	CAM <u>OR</u> HOLDING REGISTER IS FULL
7	HSO HOLDING REGISTER IS FULL

270250-56

IOC1 (16H)

0	SELECT PWM / $\overline{\text{SELECT P2.5}}$
1	EXTERNAL INTERRUPT ACH7 / $\overline{\text{EXTINT}}$
2	TIMER 1 OVERFLOW INTERRUPT ENABLE / $\overline{\text{DISABLE}}$
3	TIMER 2 OVERFLOW INTERRUPT ENABLE / $\overline{\text{DISABLE}}$
4	HSO.4 OUTPUT ENABLE / $\overline{\text{DISABLE}}$
5	SELECT TXD / $\overline{\text{SELECT P2.0}}$
6	HSO.5 OUTPUT ENABLE / $\overline{\text{DISABLE}}$
7	HSI INTERRUPT FIFO FULL / $\overline{\text{HOLDING REGISTER LOADED}}$

270250-57

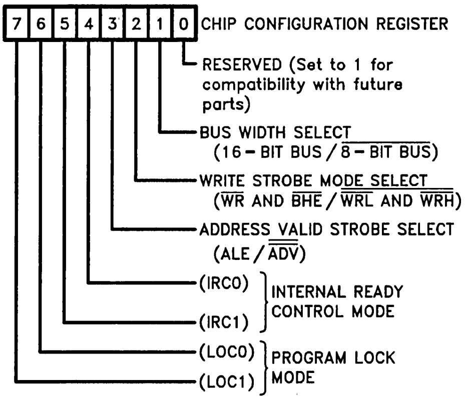
Vector	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

IOS1 (16H)

0	SOFTWARE TIMER 0 EXPIRED
1	SOFTWARE TIMER 1 EXPIRED
2	SOFTWARE TIMER 2 EXPIRED
3	SOFTWARE TIMER 3 EXPIRED
4	TIMER 2 HAS OVERFLOW
5	TIMER 1 HAS OVERFLOW
6	HSI FIFO IS FULL
7	HSI HOLDING REGISTER DATA AVAILABLE

270250-58

Chip Configuration


7 6 5 4 3 2 1 0
CHIP CONFIGURATION REGISTER
RESERVED (Set to 1 for compatibility with future parts)
BUS WIDTH SELECT (16-BIT BUS / $\overline{\text{8-BIT BUS}}$)
WRITE STROBE MODE SELECT ($\overline{\text{WR}}$ AND $\overline{\text{BHE}}$ / $\overline{\text{WRL}}$ AND $\overline{\text{WRH}}$)
ADDRESS VALID STROBE SELECT (ALE / $\overline{\text{ADV}}$)
(IRC0)
(IRC1)
INTERNAL READY CONTROL MODE
(LOC0)
(LOC1)
PROGRAM LOCK MODE

270250-59

Internal Ready Control

IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

Program Lock Modes

LOC1	LOC0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

Programming Function PMODE Values

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming Mode
0DH	Program Configuration Byte
0EH–0FH	Reserved

Slave Programming Mode Commands

P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

8X9XBH Signature Word

Device	Signature Word
879XBH	896FH
839XBH	896EH
809XBH	Undefined

Port 2 Pin Functions

Port	Function	Alternate Function
P2.0	Output	TXD (Serial Port Transmit)
P2.1	Input	RXD (Serial Port Receive)
P2.2	Input	EXTINT (External Interrupt)
P2.3	Input	T2CLK (Timer 2 Clock)
P2.4	Input	T2RST (Timer 2 Reset)
P2.5	Output	PWM (Pulse Width Modulation)

MCS®-96 Instruction Set

18

MCS®-96 INSTRUCTION SET

OVERVIEW

This chapter of the manual gives a description of each instruction recognized by the 8096. The instructions are sorted alphabetically by the mnemonic used in the assembly language for the 8096. A summary of the instruction set is included in Section 14 of the MCS®-96 Architecture chapter.

The instruction set descriptions in the following sections do not always show the effect on the program counter (PC). Unless otherwise specified, all instructions increment the PC by the number of bytes in the instruction.

A set of acronyms are used to make the instruction set descriptions easier to read, their definitions are listed below:

aa. A two bit field within an opcode which selects the basic addressing mode user. This field is only present in those opcodes which allow address mode options. The encoding of the field is as follows:

aa	Addressing mode
00	Register direct
01	Immediate
10	Indirect
11	Indexed

The selection between indirect and indirect with auto-increment or between short and long indexing is done based on the least significant bit of the instruction byte which follows the opcode. This type selects the 16-bit register which is to take part in the address calculation. Since the 8096 requires that words be aligned on even byte boundaries this bit would be otherwise unused.

breg. A byte register in the internal register file. When confusion could exist as to whether this field refers to a source or a destination register it will be prefixed with an "S" or a "D".

baop. A byte operand which is addressed by any of the address modes discussed in Section 3.2 of the MCS-96 Architecture chapter.

bitno. A three bit field within an instruction op-code which selects one of the eight bits in a byte.

wreg. A word register in the internal register file. When confusion could exist as to whether this field refers to a source register or a destination register it will be prefixed with an "S" or a "D".

waop. A word operand which is addressed by any of the address modes discussed in Section 3.2 of the MCS-96 Architecture chapter.

Lreg. A 32-bit register in the internal register file.

BEA. Extra bytes of code required for the address mode selected.

CEA. Extra state times (cycles) required for the address mode selected.

cadd. An address in the program code.

Flag Settings. The modification to the flag setting is shown for each instruction. A checkmark (✓) means that the flag is set or cleared as appropriate. A hyphen means that the flag is not modified. A one or zero (1) or (0) indicates that the flag will be in that state after the instruction. An up arrow (↑) indicates that the instruction may set the flag if it is appropriate but will not clear the flag. A down arrow (↓) indicates that the flag can be cleared but not set by the instruction. A question mark (?) indicates that the flag will be left in an indeterminant state after the operation.

Generic Jumps and Calls. The assembler for the MCS-96 family provides for generic jumps and calls. For all of the conditional jump instructions a "B" can be substituted for the "J" and the assembler will generate a code sequence which is logically equivalent but can reach anywhere in the memory. A JH can only jump about 128 locations from the current program counter; a BH can jump anywhere in memory. In a like manner a BR will cause a SJMP or LJMP to be generated as appropriate and a CALL will cause a SCALL or LCALL to be generated. The assembler user's guide should be consulted for the algorithms used by the assembler to convert these generic instructions into actual machine instructions.

Indirect Shifts. The indirect shift operations use registers 24 through 255 (18H–0FFH), since 0–15 are direct operators and registers 16 through 23 are Special Function Registers. Note that indirect shifts through SFRs are illegal operations.

The maximum shift count is 31 (1FH). Count values above this will be truncated to the 5 least significant bits.

1. ADD (Two Operands) — ADD WORDS

Operation: The sum of the two word operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

```
        DST     SRC
ADD     wreg,   waop
```

Object Code Format: [011001aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

2. ADD (Three Operands) — ADD WORDS

Operation: The Sum of the second and third word operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
ADD     Dwreg,  Swreg,  waop
```

Object Code Format: [010001aa] [waop] [Swreg] [Dwreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

3. ADDB (Two Operands) — ADD BYTES

Operation: The sum of the two byte operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

	DST	SRC
ADDB	breg,	baop

Object Code Format: [011101aa] [baop] [breg]

Bytes 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

4. ADDB (Three Operands) — ADD BYTES

Operation: The sum of the second and third byte operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
ADDB	Dbreg,	Sbreg,	baop

Object Code Format: [010101aa] [baop] [Sbreg] [Dbreg]

Bytes: 3 + BEA
States 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

5. ADDC — ADD WORDS WITH CARRY

Operation: The sum of the two word operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

	DST	SRC
ADDC	wreg,	waop

Object Code Format: [101001aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + BEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

6. ADDCB — ADD BYTES WITH CARRY

Operation: The sum of the two byte operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

	DST	SRC
ADDCB	breg,	baop

Object Code Format: [101101aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

7. AND (Two Operands) — LOGICAL AND WORDS

Operation: The two word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
       DST    SRC
AND    wreg,  waop
```

Object Code Format: [011000aa] [waop] [wreg]

Bytes: 2 + BEA
States 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

8. AND (Three Operands) — LOGICAL AND WORDS

Operation: The second and third word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
       DST     SRC1    SRC2
AND    Dwreg,  Swreg,  waop
```

Object Code Format: [010000aa] [waop] [Swreg] [Dwreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

9. ANDB (Two Operands) — LOGICAL AND BYTES

Operation: The two byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
        DST     SRC
ANDB    breg,   baop
```

Object Code Format: [011100aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

10. ANDB (Three Operands) — LOGICAL AND BYTES

Operation: The second and third byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
ANDB    Dbreg,  Sbreg,  baop
```

Object Code Format: [010100aa] [baop] [Sbreg] [Dbreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

11. BR (Indirect) — BRANCH INDIRECT

Operation: The execution continues at the address specified in the operand word register.

PC ← (DEST)

Assembly Language Format: BR [wreg]

Object Code Format: [11100011] [wreg]

Bytes: 2
States: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

12. CLR — CLEAR WORD

Operation: The value of the word operand is set to zero.

(DEST) ← 0

Assembly Language Format: CLR wreg

Object Code Format: [00000001] [wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
1	0	0	0	—	—

13. CLRB — CLEAR BYTE

Operation: The value of the byte operand is set to zero.
(DEST) ← 0

Assembly Language Format: CLRB breg

Object Code Format: [00010001] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
1	0	0	0	—	—

14. CLRC — CLEAR CARRY FLAG

Operation: The value of the carry flag is set to zero.
C ← 0

Assembly Language Format: CLRC

Object Code Format: [11111000]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	0	—	—	—

15. CLRVT — CLEAR OVERFLOW TRAP

Operation: The value of the overflow-trap flag is set to zero.

VT ← 0

Assembly Language Format: CLRVT

Object Code Format: [11111100]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

16. CMP — COMPARE WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

```
        DST    SRC
CMP     wreg,  waop
```

Object Code Format: [100010aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

17. CMPB — COMPARE BYTES

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
CMPB    breg,   baop
```

Object Code Format: [100110aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

18. DEC — DECREMENT WORD

Operation: The value of the word operand is decremented by one.

(DEST) ← (DEST) — 1

Assembly Language Format: DEC wreg

Object Code Format: [00000101] [wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

19. DECB — DECREMENT BYTE

Operation: The value of the byte operand is decremented by one.
(DEST) ← (DEST) — 1

Assembly Language Format: DECB breg

Object Code Format: [00010101] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

20. DI — DISABLE INTERRUPTS

Operation: Interrupts are disabled. Interrupt-calls will not occur after this instruction.
Interrupt Enable (PSW.9) ← 0

Assembly Language Format: DI

Object Code Format: [11111010]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

21. DIV — DIVIDE INTEGERS

Operation: This instruction divides the contents of the destination LONG-INTEGER operand by the contents of the INTEGER word operand, using signed arithmetic. The low order word of the destination (i.e., the word with the lower address) will contain the quotient; the high order word will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
        DST     SRC
DIV     lreg,   waop
```

Object Code Format: [11111110] [100011aa] [waop] [lreq]

Bytes: 2 + BEA
States: 29 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	?	↑	—

22. DIVB — DIVIDE SHORT-INTEGERS

Operation: This instruction divides the contents of the destination INTEGER operand by the contents of the source SHORT-INTEGER operand, using signed arithmetic. The low order byte of the destination (i.e. the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
        DST     SRC
DIVB    wreg,   baop
```

Object Code Format: [11111110] [100111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 21 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	?	↑	—

23. DIVU — DIVIDE WORDS

Operation: This instruction divides the content of the destination DOUBLE-WORD operand by the contents of the source WORD operand, using unsigned arithmetic. The low order word will contain the quotient; the high order WORD will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
        DST     SRC
DIVU    lreg,   waop
```

Object Code Format: [100011aa] [waop] [lreq]

Bytes: 2 + BEA
States: 25 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	✓	↑	—

24. DIVUB — DIVIDE BYTES

Operation: This instruction divides the contents of the destination WORD operand by the contents of the source BYTE operand, using unsigned arithmetic. The low order byte of the destination, (i.e., the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
        DST     SRC
DIVUB   wreg,   baop
```

Object Code Format: [100111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 17 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	✓	↑	—

25. DJNZ — DECREMENT AND JUMP IF NOT ZERO

Operation: The value of the byte operand is decremented by 1. If the result is not equal to 0, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the result of the decrement is zero then control passes to the next sequential instruction.

```
(COUNT) ← (COUNT) — 1
if (COUNT) <> 0 then
  PC ← PC + disp (sign-extended to 16 bits)
end_if
```

Assembly Language Format: DJNZ breg,cadd

Object Code Format: [11100000] [breg] [disp]

Bytes:		3
States:	Jump Not Taken:	5
	Jump Taken:	9

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

26. EI — ENABLE INTERRUPTS

Operation: Interrupts are enabled following the execution of the next statement. Interrupt-calls cannot occur immediately following this instruction.

Interrupt Enable (PSW.9) ← 1

Assembly Language Format: EI

Object Code Format: [11111011]

Bytes:	1
States:	4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

27. EXT — SIGN EXTEND INTEGER INTO LONG-INTEGER

Operation: The low order word of the operand is sign-extended throughout the high order word of the operand.

```
if (low word DEST) < 8000H then
  (high word DEST) ← 0
else
  (high word DEST) ← 0FFFFH
end_if
```

Assembly Language Format: EXT lreg

Object Code Format: [00000110] [lreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

28. EXTB — SIGN EXTEND SHORT-INTEGER INTO INTEGER

Operation: The low order byte of the operand is sign-extended throughout the high order byte of the operand.

```
if (low byte DEST) < 80H then
  (high byte DEST) ← 0
else
  (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format: EXTB wreg

Object Code Format: [00010110] [wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

29. INC — INCREMENT WORD

Operation: The value of the word operand is incremented by 1.
(DEST) ← (DEST) + 1

Assembly Language Format: INC wreg

Object Code Format: [00000111] [wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

30. INCB — INCREMENT BYTE

Operation: The value of the byte operand is incremented by 1.
(DEST) ← (DEST) + 1

Assembly Language Format: INCB breg

Object Code Format: [00010111] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

31. JBC — JUMP IF BIT CLEAR

Operation: The specified bit is tested. If it is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is set (i.e., 1), control passes to the next sequential instruction.

```
if (specified bit) = 0 then
  PC ← PC + disp (sign-extened to 16 bits)
```

Assembly Language Format: JBC breg,bitno,cadd

Object Code Format: [00110bbb] [breg] [disp]
where bbb is the bit number within the specified register.

Bytes:		3
States:	Jump Not Taken:	5
	Jump Taken:	9

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

32. JBS — JUMP IF BIT SET

Operation: The specified bit is tested. If it is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is clear (i.e., 0), control passes to the next sequential instruction.

```
if (specified bit) = 1 then
  PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JBS breg,bitno,cadd

Object Code Format: [00111bbb][breg] [disp]
where bbb is the bit number within the specified register.

Bytes:		3
States:	Jump Not Taken:	5
	Jump Taken:	9

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

33. JC — JUMP IF CARRY FLAG IS SET

Operation: If the carry flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if C = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JC cadd

Object Code Format: [11011011] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

34. JE — JUMP IF EQUAL

Operation: If the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JE cadd

Object Code Format: [11011111] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

35. JGE — JUMP IF SIGNED GREATER THAN OR EQUAL

Operation: If the negative flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is set (i.e., 1), control passes to the next sequential instruction.

```
if N = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGE cadd

Object Code Format: [11010110] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

36. JGT — JUMP IF SIGNED GREATER THAN

Operation: If both the negative flag and the zero flag are clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the negative flag or the zero flag are set (i.e., 1,) control passes to the next sequential instruction.

```
if N = 0 AND Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGT cadd

Object Code Format: [11010010] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

37. JH — JUMP IF HIGHER (UNSIGNED)

Operation: If the carry flag is set (i.e., 1), but the zero flag is not, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction.

```
if C = 1 AND Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JH cadd

Object Code Format: [11011001] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

38. JLE — JUMP IF SIGNED LESS THAN OR EQUAL

Operation: If either the negative flag or the zero flag are set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to + 127. If both the negative flag and the zero flag are clear (i.e., 0), control passes to the next sequential instruction.

```
if N = 1 OR Z = 1 then
    PC <— PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLE cadd

Object Code Format: [11011010] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

39. JLT — JUMP IF SIGNED LESS THAN

Operation: If the negative flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if N = 1 then
    PC <— PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLT cadd

Object Code Format: [11011110] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

40. JNC — JUMP IF CARRY FLAG IS CLEAR

Operation: If the carry flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is set (i.e., 1), control passes to the next sequential instruction.

```
if C = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNC cadd

Object Code Format: [11010011] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

41. JNE — JUMP IF NOT EQUAL

Operation: If the zero flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is set (i.e., 1), control passes to the next sequential instruction.

```
if Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNE cadd

Object Code Format: [11010111 [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

42. JNH — JUMP IF NOT HIGHER (UNSIGNED)

Operation: If either the carry flag is clear (i.e., 0), or the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to + 127. If the carry flag is set (i.e., 1) and the zero flag is not, control passes to the next sequential instruction.

```
if C = 0 OR Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNH cadd

Object Code Format: [11010001] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

43. JNST — JUMP IF STICKY BIT IS CLEAR

Operation: If the sticky bit flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is set (i.e., 1), control passes to the next sequential instruction.

```
if ST = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNST cadd

Object Code Format: [11010000] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

44. JNV — JUMP IF OVERFLOW FLAG IS CLEAR

Operation: If the overflow flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is set (i.e., 1), control passes to next sequential instruction.

```
if V = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNV cadd

Object Code Format: [11010101] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

45. JNVT — JUMP IF OVERFLOW TRAP IS CLEAR

Operation: If the overflow trap flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is set (i.e., 1), control passes to the next sequential instruction. The VT flag is cleared.

```
if VT = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNVT cadd

Object Code Format: [11010100] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jumps Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

46. JST — JUMP IF STICKY BIT IS SET

Operation: If the sticky bit flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if ST = 1 then
    PC <— PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JST cadd

Object Code Format: [11011000] [disp]

Bytes:		2
State:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

47. JV — JUMP IF OVERFLOW FLAG IS SET

Operation: If the overflow is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if V = 1 then
    PC <— PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JV cadd

Object Code Format: [11011101] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

48. JVT — JUMP IF OVERFLOW TRAP IS SET

Operation: If the overflow trap flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is clear (i.e., 0), control passes to the next sequential instruction. The VT flag is cleared.

```
if VT = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JVT cadd

Object Code Format: [11011100] [disp]

Bytes:		2
States:	Jump Not Taken:	4
	Jump Taken:	8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

49. LCALL — LONG CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The operand may be any address in the entire address space.

```
SP ← SP − 2
(SP) ← PC
PC ← PC + disp
```

Assembly Language Format: LCALL cadd

Object Code Format: [11101111] [disp-low] [disp-hi]

Bytes:		3
States:	Onchip stack:	13
	Offchip stack:	16

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

50. LD — LOAD WORD

Operation: The value of the source (rightmost) word operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

	DST	SRC
LD	wreg,	waop

Object Code Format: [101000aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

51. LDB — LOAD BYTE

Operation: The value of the source (rightmost) byte operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

	DST	SRC
LDB	breg,	baop

Object Code Format: [101100aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

52. LDBSE — LOAD INTEGER WITH SHORT-INTEGER

Operation: The value of the source (rightmost) byte operand is sign-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
if (SRC) < 80H then
    (high byte DEST) ← 0
else
    (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format:

```
        DST     SRC
LDBSE   wreg,   baop
```

Object Code Format: [101111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

53. LDBZE — LOAD WORD WITH BYTE

Operation: The value of the source (rightmost) byte operand is zero-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
(high byte DEST) ← 0
```

Assembly Language Format:

```
        DST     SRC
LDBZE   wreg,   baop
```

Object Code Format: [101011aa] [baop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

54. LJMP — LONG JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The operand may be any address in the entire address space.

PC ← PC + disp

Assembly Language Format: LJMP cadd

Object Code Format: [11100111] [disp-low] [disp-hi]

Bytes: 3
States: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

55. MUL (Two Operands) — MULTIPLY INTEGERS

Operation: The two INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG-INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
        DST     SRC
MUL     lreg,   waop
```

Object Code Format: [11111110] [011011aa] [waop] [lreg]

Bytes: 3 + BEA
States 29 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

56. MUL (Three Operands) — MULTIPLY INTEGERS

Operation: The second and third INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
MUL     lreg,   wreg,   waop
```

Object Code Format: [11111110] [010011aa] [waop] [wreg] [lreg]

Bytes: 4 + BEA
States: 30 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

57. MULB (Two Operands) — MULTIPLY SHORT-INTEGERS

Operation: The two SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
        DST     SRC
MULB    wreg,   baop
```

Object Code Format: [11111110] [011111aa] [baop] [wreg]

Bytes: 3 + BEA
States: 21 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

58. MULB (Three Operands) — MULTIPLY SHORT-INTEGERS

Operation: The second and third SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MULB	wreg,	breg	baop

Object Code Format: [11111110] [010111aa] [baop] [breg] [wreg]

Bytes: 4 + BEA
States: 22 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

59. MULU (Two Operands) — MULTIPLY WORDS

Operation: The two WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

	DST	SRC
MULU	lreg,	waop

Object Code Format: [011011aa] [waop] [lreg]

Bytes: 2 + BEA
States: 25 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

60. MULU (Three Operands) — MULTIPLY WORDS

Operation: The second and third WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MULU	lreg,	wreg,	waop

Object Code Format: [010011aa] [waop] [wreg] [lreg]

Bytes: 3 + BEA
States: 26 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

61. MULUB (Two Operands) — MULTIPLY BYTES

Operation: The two BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

	DST	SRC
MULUB	wreg,	baop

Object Code Format: [011111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 17 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

62. MULUB (Three Operands) — MULTIPLY BYTES

Operation: The second and third BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MULUB	wreg,	breg,	baop

Object Code Format: [010111aa] [baop] [breg] [wreg]

Bytes: 3 + BEA
States: 18 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

63. NEG — NEGATE INTEGER

Operation: The value of the INTEGER operand is negated.

(DEST) ← −(DEST)

Assembly Language Format: NEG wreg

Object Code Format: [00000011] [wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

64. NEGB — NEGATE SHORT-INTEGER

Operation: The value of the SHORT-INTEGER operand is negated.
(DEST) ← −(DEST)

Assembly Language Format: NEGB breg

Object Code Format: [00010011] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

65. NOP — NO OPERATION

Operation: Nothing is done. Control passes to the next sequential instruction.

Assembly Language Format: NOP

Object Code Format: [11111101]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

66. NORML — NORMALIZE LONG-INTEGER

Operation: The LONG-INTEGER operand is normalized; i.e., it is shifted to the left until its most significant bit is 1. If the most significant bit is still 0 after 31 shifts, the process stops and the zero flag is set. The number of shifts actually performed is stored in the second operand.

```
(COUNT) ← 0
do while (MSB(DEST) = 0) AND ((COUNT) < 31)
    (DEST) ← (DEST) * 2
    (COUNT) ← (COUNT) + 1
end_while
```

Assembly Language Format: NORML lreg,breg

Object Code Format: [00001111] [breg] [lreg]

Bytes: 3
States: 11 + No. of shifts performed

Flags Affected					
Z	N	C	V	VT	ST
✓	?	0	—	—	—

67. NOT — COMPLEMENT WORD

Operation: The value of the WORD operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

(DEST) ← NOT (DEST)

Assembly Language Format: NOT wreg

Object Code Format: [00000010] [wreg]

Bytes 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

68. NOTB — COMPLEMENT BYTE

Operation: The vaule of the BYTE operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

(DEST) ← NOT (DEST)

Assembly Language Format: NOTB breg

Object Code Format: [00010010] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

69. OR — LOGICAL OR WORDS

Operation: The source (rightmost) WORD is ORed with the destination (leftmost) WORD operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand is 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format:

```
        DST     SRC
OR      wreg,   waop
```

Object Code Format: [100000aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

70. ORB — LOGICAL OR BYTES

Operation: The source (rightmost) BYTE operand is ORed with the destination (leftmost) BYTE operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format: ORB breg,baop

Object Code Format: [100100aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

71. POP — POP WORD

Operation: The word on top of the stack is popped and placed at the destination operand.

(DEST) ← (SP)
SP ← SP + 2

Assembly Language Format: POP waop

Object Code Format: [110011aa] [waop]

Bytes		1 + BEA
States:	Onchip Stack:	12 + CEA
	Offchip Stack:	14 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

72. POPF — POP FLAGS

Operation: The word on top of the stack is popped and placed in the PSW. Interrupt calls cannot occur immediately following this instruction.

(PSW) ← (SP)
SP ← SP + 2

Assembly Language Format: POPF

Object Code Format: [11110011]

Bytes:		1
States:	Onchip Stack:	9
	Offchip Stack:	13

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	✓	✓

73. PUSH — PUSH WORD

Operation: The specified operand is pushed onto the stack.

SP ← SP − 2
(SP) ← (DEST)

Assembly Language Format: PUSH waop

Object Code Format: [110010aa] [waop]

Bytes:		1 + BEA
States:	Onchip Stack:	8 + CEA
	Offchip Stack:	12 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

74. PUSHF — PUSH FLAGS

Operation: The PSW is pushed on top of the stack, and then set to all zeroes. This implies that all interrupts are disabled. Interrupt-calls cannot occur immediately following this instruction.

SP ← SP − 2
(SP) ← PSW
PSW ← 0

Assembly Language Format: PUSHF

Object Code Format: [11110010]

Bytes:		1
States:	Onchip Stack:	8
	Offchip Stack:	12

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

75. RET — RETURN FROM SUBROUTINE

Operation: The PC is popped off the top of the stack.

PC ← (SP)
SP ← SP + 2

Assembly Language Format: RET

Object Code Format: [11110000]

Bytes:		1
States:	Onchip Stack:	12
	Offchip Stack:	16

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

76. RST — RESET SYSTEM

Operation: The PSW is initialized to zero, and the PC is initialized to 2080H. The I/O registers are set to their initial value. Executing this instruction will cause a pulse to appear on the reset pin of the 8096.

PSW ← 0
PC ← 2080H

Assembly Language Format: RST

Object Code Format: [11111111]

Bytes: 1
States: 16

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

77. SCALL — SHORT CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The offset from the end of this instruction to the target label must be in the range of −1024 to +1023 inclusive.

SP ← SP − 2
(SP) ← PC
PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SCALL cadd

Object Code Format: [00101xxx] [disp-low]
where xxx holds the three high-order bits of displacement.

Bytes:		2
States	Onchip Stack:	13
	Offchip Stack:	16

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

78. SETC — SET CARRY FLAG

Operation: The carry flag is set.

```
C ← 1
```

Assembly Language Format: SETC

Object Code Format: [11111001]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	1	—	—	—

79. SHL — SHIFT WORD LEFT

Operation: The destination (leftmost) word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    Temp ← Temp — 1
end_while
```

Assembly Language Format:

```
        SHL     wreg,#count
or
        SHL     wreg,breg
```

Object Code Format: [00001001] [cnt/breg] [wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	—

80. SHLB — SHIFT BYTE LEFT

Operation: The destination (leftmost) byte operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    TEMP ← Temp − 1
end_while
```

Assembly Language Format:

```
          SHLB     breg,#count
or
          SHLB     breg,breg
```

Object Code Format: [00011001] [cnt/breg] [breg]

Bytes 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	—

81. SHLL — SHIFT DOUBLE-WORD LEFT

Operation: The destination (leftmost) double-word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
          SHLL      lreg,#count
or
          SHLL      lreg, breg
```

Object Code Format: [00001101] [cnt/breg] [lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	—

82. SHR — LOGICAL RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved to the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHR     wreg,#count
or
        SHR     wreg,breg
```

Object Code Format: [00001000] [cnt/breg] [wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

83. SHRA — ARITHMETIC RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRA    wreg,#count
or
        SHRA    wreg,breg
```

Object Code Format: [00001010] [cnt/breg] [wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

84. SHRAB — ARITHMETIC RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If that value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C, = Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
          SHRAB    breg,#count
or
          SHRAB    breg,breg
```

Object Code Format: [00011010] [cnt/breg] [breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

85. SHRAL — ARITHMETIC RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The sticky bit is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRAL    lreg,#count
or
        SHRAL    lreg,breg
```

Object Code Format: [00001110] [cnt/breg] [lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

86. SHRB — LOGICAL RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRB    breg,#count
or
        SHRB    breg,breg
```

Object Code Format: [00011000] [cnt/breg] [breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

87. SHRL — LOGICAL RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DSET) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRL    lreg,#count
or
        SHRL    lreg,breg
```

Object Code Format: [00001100] [cnt/breg] [lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

88. SJMP — SHORT JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the label must be in the range of −1024 to +1023 inclusive.

PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SJMP cadd

Object Code Format: [00100xxx] [disp-low]
where xxx holds the three high order bits of the displacement.

Bytes: 2
States: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

89. SKIP — TWO BYTE NO-OPERATION

Operation: Nothing is done. This is actually a two-byte NOP where the second byte can be any value, and is simply ignored. Control passes to the next sequential instruction.

Assembly Language Format: SKIP breg

Object Code Format: [00000000] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

90. ST — STORE WORD

Operation: The value of the leftmost word operand is stored into the rightmost operand.

(DEST) ← (SRC)

Assembly Language Format:

```
       SRC     DST
ST     wreg,   waop
```

Object Code Format: [110000aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

91. STB — STORE BYTE

Operation: The value of the leftmost byte operand is stored into the rightmost operand.

(DEST) ← (SRC)

Assembly Language Format:

```
       SRC     DST
STB    breg,   baop
```

Object Code Format: [110001aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

92. SUB (Two Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
SUB     wreg,   waop
```

Object Code Format: [011010aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

93. SUB (Three Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the second word operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

(DEST) ← (SRC1) — (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2,
SUB     wreg,   wreg,   waop
```

Object Code Format: [010010aa] [waop] [Sweg] [Dwreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

94. SUBB (Two Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

	DST	SRC
SUBB	breg,	baop

Object Code Format: [011110aa] [baop] [breg]

Bytes: 2 + BEA
States 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

95. SUBB (Three Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte operand is subtracted from the second byte operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

(DEST) ← (SRC1) — (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
SUBB	breg,	Sbreg	baop

Object Code Format: [010110aa] [baop] [Sbreg] [Dbreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

96. SUBC — SUBTRACT WORDS WITH BORROW

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the orignal destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

	DST	SRC
SUBC	wreg,	waop

Object Code Format: [101010aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

97. SUBCB — SUBTRACT BYTES WITH BORROW

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

	DST	SRC
SUBCB	breg,	baop

Object Code Format: [101110aa] [baop] [breg]

Bytes: 2 + BEA
States 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

98. TRAP — SOFTWARE TRAP

Operation: This instruction causes an interrupt-call which is vectored through location 2010H. The operation of this instruction is not effected by the state of the interrupt enable flag in the PSW (I). Interrupt-calls cannot occur immediately following this instruction. This instruction is intended for use by Intel provided development tools. These tools will not support user-application of this instruction.

SP ← SP − 2
(SP) ← PC
PC ← (2010H)

Assembly Language Format: This instruction is not supported by revision 1.0 of the 8096 assembly language.

Object Code Format: [11110111]

Bytes:		1
States	Onchip Stack:	21
	Offchip Stack:	24

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

99. XOR — LOGICAL EXCLUSIVE-OR WORDS

Operation: The source (rightmost) word operand is XORed with the destination (leftmost) word operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

Assembly Language Format:

```
        DST     SRC
XOR     wreg,   waop
```

Object Code Format: [100001aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

100. XORB — LOGICAL EXCLUSIVE-OR BYTES

Operation: The source (rightmost) byte operand is XORed with the destination (leftmost) byte operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

Assembly Language Format:

	DST	SRC
XORB	breg,	baop

Object Code Format: [100101aa] [baop] [breg]

Bytes:	2 + BEA
States:	4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

MCS®-96 Hardware Design Information

19

MCS®-96 HARDWARE DESIGN INFORMATION

OVERVIEW

This Chapter of the manual is devoted to the hardware engineer. All of the information you need to connect the correct pin to the correct external circuit is provided. Many of the special function pins have different characteristics which are under software control, therefore, it is necessary to define the system completely before the hardware is wired-up.

Frequently within this chapter a specification for a current, voltage, or time period is referred to; the values provided are to be used as an approximation only. The exact specification can be found in the latest data sheet for the particular part and temperature range that is being used.

1.0 REQUIRED HARDWARE CONNECTIONS

Although the 8096BH is a single-chip microcontroller, it still requires several external connections to make it work. Power must be applied, a clock source provided, and some form of reset circuitry must be present. We will look at each of these areas of circuitry separately. Figure 6 shows the connections that are needed for a single-chip system.

1.1 Power Supply Information

Power for the 8096BH flows through six pins; they are: three positive voltage pins—V_{CC} (digital), V_{REF} (Port 0 digital I/O and A/D power), V_{PD} (power down mode); and three common returns—two V_{SS} pins and one ANGND pin. All six of these pins **must** be connected on the 8096BH for normal operation. The V_{CC} pin, V_{REF} pin and V_{PD} pin should be tied to 5 volts. The two V_{SS} pins and the ANGND pin must be grounded. When the analog to digital converter is being used it may be desirable to connect the VREF pin to a separate power supply, or at least a separate power supply line.

The three common return pins should be connected at the chip with as short a lead as possible to avoid problems due to voltage drops across the wiring. There should be no measurable voltage difference between V_{SS1} and V_{SS2}. The two V_{SS} pins and the ANGND pin must all be nominally at 0 volts. The maximum current drain of the 8096BH is around 180 mA, with all lines unloaded.

When the analog converter is being used, clean, stable power must be provided to the analog section of the chip to assure highest accuracy. To achieve this, it may be desirable to separate the analog power supply from the digital power supply. The V_{REF} pin supplies the digital circuitry in the A/D converter and provides the 5 volt reference to the analog portion of the converter. V_{REF} and ANGND must be connected even if the A/D converter is not used. More information on the analog power supply is in Section 3.1.

1.2 Other Needed Connections

Several other connections are needed to configure the 8096BH. In normal operation the following pins should be connected to the indicated power supply.

Pin		Power Supply
8X9XBH	**8X9X**	
NMI	NMI	V_{CC}
	$\overline{\text{TEST}}$	V_{CC}
$\overline{\text{EA}}$	$\overline{\text{EA}}$	V_{CC} (to allow internal execution)
		V_{SS} (to force external execution)

Although the $\overline{\text{EA}}$ pin has an internal pulldown, it is best to tie this pin to the desired level. This will prevent induced noise from disturbing the system. With the exception of 8X9X devices, raising $\overline{\text{EA}}$ to +12.75 volts will place an 8096BH in a special operating mode designed for programming and program memory verification (see Section 10).

1.3 Oscillator Information

The 8096BH requires a clock source to operate. This clock can be provided to the chip through the XTAL1 input or the on-chip oscillator can be used. The frequency of operation is from 6 MHz to 12 MHz.

The on-chip circuitry for the 8096BH oscillator is a single stage linear inverter as shown in Figure 1. It is intended for use as a crystal-controlled, positive reactance oscillator with external connections as shown in Figure 2. In this application, the crystal is being operated in its fundamental response mode as an inductive reactance in parallel resonance with shunt capacitance external to the crystal.

The crystal specifications and capacitance values (C1 and C2 in Figure 2) are not critical. Thirty pF can be used in these positions at any frequency with good quality crystals. For 0.5% frequency accuracy, the crystal frequency can be specified at series resonance or for parallel resonance with any load capacitance. (In other words, for that degree of frequency accuracy, the load capacitance simply doesn't matter.) For 0.05% frequency accuracy the crystal frequency should be specified for parallel resonance with 25 pF load capacitance, if C1 and C2 are 30 pF.

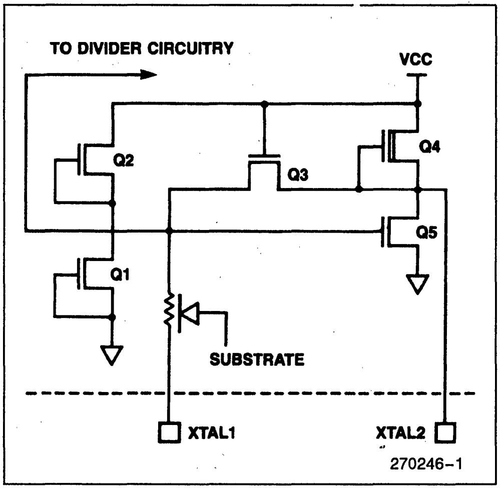


Figure 1. 8096BH Oscillator Circuit

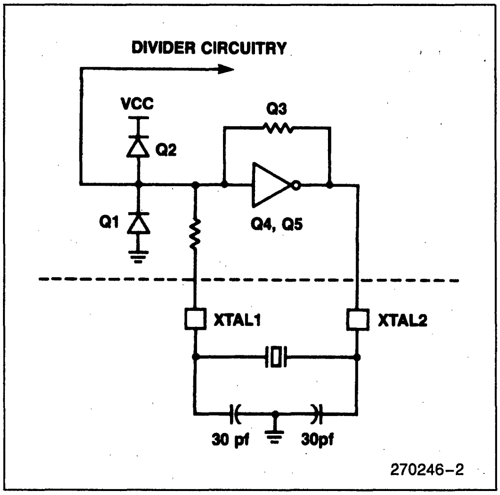


Figure 2. Crystal Oscillator Circuit

A more in-depth discussion of crystal specifications and the selection of values for C1 and C2 can be found in the Intel Application Note, AP-155, "Oscillators for Microcontrollers."

To drive the 8096BH with an external clock source, apply the external clock signal to XTAL1 and let XTAL2 float. An example of this circuit is shown in Figure 3. The required voltage levels on XTAL1 are specified in the data sheet. The signal on XTAL1 must be clean with good solid levels.

It is important that the minimum high and low times are met to avoid having the XTAL1 pin in the transition range for long periods of time. The longer the signal is in the transition region, the higher the probability that an external noise glitch could be seen by the clock generator circuitry. Noise glitches on the 8096BH internal clock lines will cause unreliable operation.

The clock generator provides a 3 phase clock output from the XTAL1 pin input. Figure 4 shows the waveforms of the major internal timing signals.

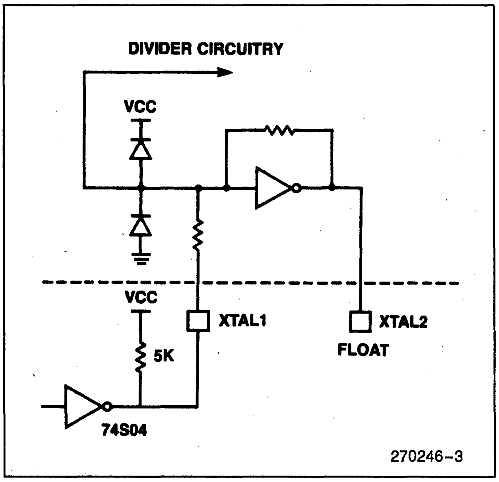


Figure 3. External Clock Drive

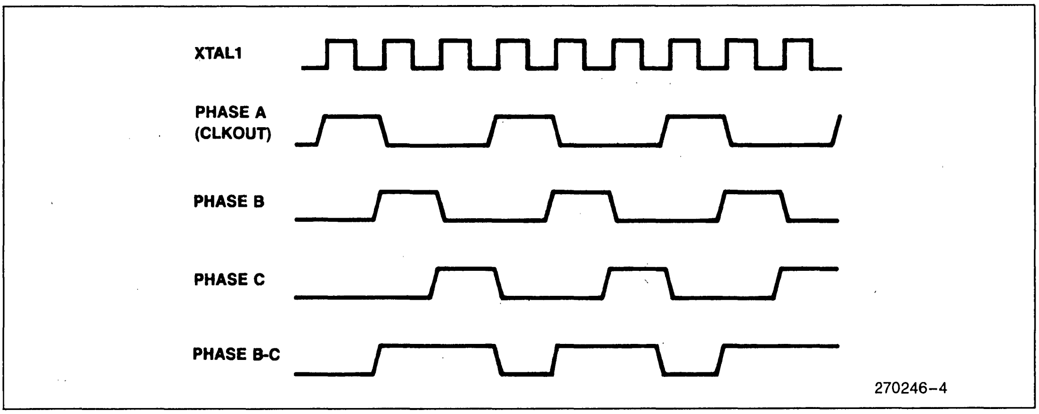


Figure 4. Internal Timings

1.4 Reset Information

In order for the 8096BH to function properly it must be reset. This is done by holding the $\overline{\text{RESET}}$ pin low for at least 2 state times after the power supply is within tolerance and the oscillator has stabilized.

On 8X9X devices the $\overline{\text{RESET}}$ pin must be held low long enough for the power supply, oscillator and back-bias generator to stabilize. Typically, the back-bias generator requires one millisecond to stabilize.

After the $\overline{\text{RESET}}$ pin is brought high, a ten state reset sequence is executed. During this time, the Chip Configuration Byte (CCB) is read from location 2018H and written to the 8096BH Chip Configuration Register (CCR). If the voltage on the $\overline{\text{EA}}$ pin selects the internal/external execution mode the CCB is read from internal ROM/EPROM. If the voltage on the $\overline{\text{EA}}$ pin selects the external execution only mode the CCB is read from external memory. See Figure 5.

On 8X9X devices, the CCB read does not occur, and ALE is high while $\overline{\text{RESET}}$ is held low.

There are several ways to provide a good reset to an 8096BH, the simplest being just to connect a capacitor from the reset pin to ground. The capacitor should be on the order of 2 microfarads for every millisecond of reset time required. This method will only work if the rise time of V_{CC} is fast and the total reset time is less than around 50 milliseconds. It also may not work if the $\overline{\text{RESET}}$ pin is to be used to reset other parts on the board. An 8096BH with the minimum required connections is shown in Figure 6.

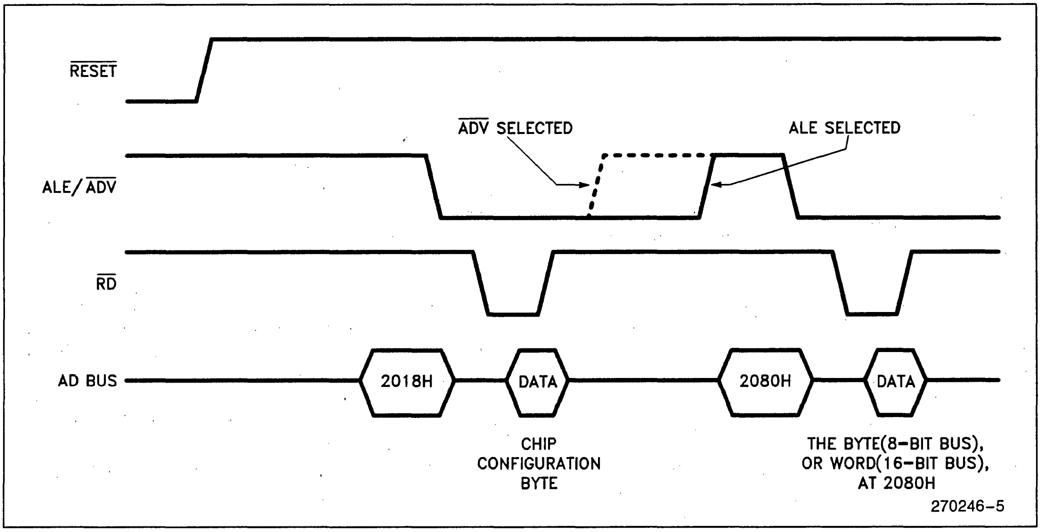
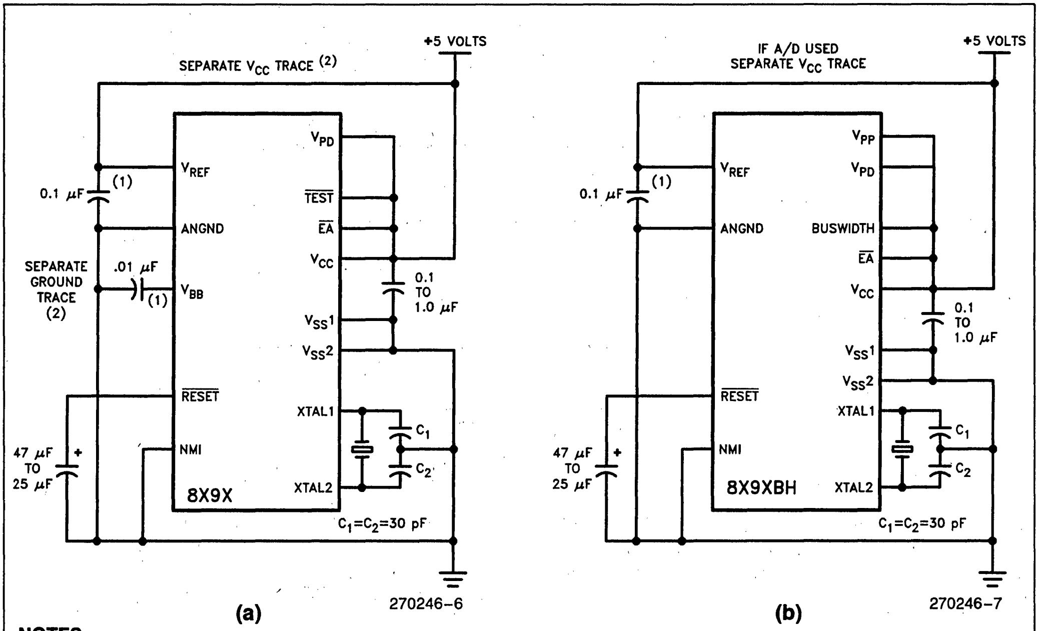


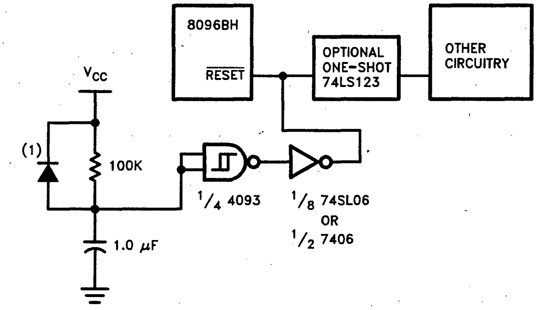
Figure 5. Reset Sequence



NOTES:

1. These capacitors are needed only if A to D is used.
2. VREF & ANGND may be connected to the same traces as the digital power supply if the A to D is not used.

Figure 6. Minimum Hardware Connections



NOTE:

1. The diode will provide a faster cycle time repetitive power-on-resets.

Figure 7. Multiple Chip Reset Circuit

The 8096BH $\overline{\text{RESET}}$ pin can be used to allow other chips on the board to make use of the Watchdog Timer or the RST instruction. When this is done the reset hardware should be a one-shot with an open collector output. The reset pulse going to the other parts may have to be buffered and lengthened with a one-shot, since the $\overline{\text{RESET}}$ low duration is only two state times. If this is done, it is possible that the 8096BH will be reset and start running before the other parts on the board are out of reset. The software must account for this possible problem.

A capacitor directly connected to $\overline{\text{RESET}}$ cannot be used to reset the part if the pin is to be used as an output. If a large capacitor is used, the pin will pull-down more slowly than normal. It will continue to pull-down until the 8096BH is reset. It could fall so slowly that it never goes below the internal switch point of the reset signal (1 to 1.5 volts), a voltage which may be above the guaranteed switch point of external circuitry connected to the pin. A circuit example is shown in Figure 7.

1.5 Sync Mode

If $\overline{\text{RESET}}$ is brought high at the same time as or just after the rising edge of XTAL1, the part will start executing the 10 state time RST instruction exactly 6½ XTAL1 cycles later. This feature can be used to synchronize several MCS-96 devices. A diagram of a typical connection is shown in Figure 8. It should be noted that parts that start in sync may not stay that way, due to propagation delays which may cause the synchronized parts to receive signals at slightly different times.

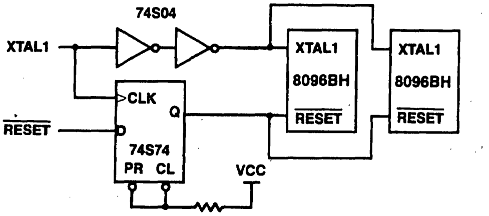
1.6 Disabling the Watchdog Timer

The Watchdog Timer will pull the $\overline{\text{RESET}}$ pin low when it overflows. See Figure 9. If the pin is being externally held above the low going threshold, the pulldown transistor will remain on indefinitely. This means that once the watchdog overflows, the part must be reset or $\overline{\text{RESET}}$ must be held high indefinitely. Just resetting the Watchdog Timer in software will not clear the flip-flop which keeps the $\overline{\text{RESET}}$ pulldown on.

The pulldown is capable of sinking on the order of 30 milliamps if it is held at 2.0 volts. This amount of current may cause some long term reliability problems due to localized chip heating. For this reason, parts that will be used in production should never have had the Watchdog Timer over-ridden for more than a second or two.

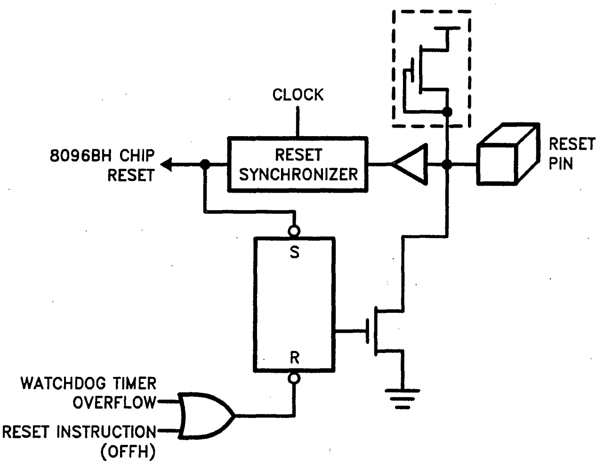
Whenever the reset pin is being pulled high while the pulldown is on, it should be through a resistor that will limit the voltage on $\overline{\text{RESET}}$ to 2.5 volts and the current through the pin to 40 milliamps.

If it is necessary to disable the Watchdog Timer for more than a brief test the software solution of never initiating the timer should be used. See Section 14 in the Architecture Chapter.


74S04
XTAL1
$\overline{\text{RESET}}$
>CLK
Q
D
74S74
PR CL
VCC
XTAL1
8096BH
$\overline{\text{RESET}}$
XTAL1
8096BH
$\overline{\text{RESET}}$

270246-9

Figure 8. Reset Sync Mode



270246-10

Figure 9. Reset Logic

1.7 Power Down Circuitry

Battery backup can be provided on the 8096BH with a 1 mA current drain at 5 volts. This mode will hold locations 0F0H through 0FFH valid as long as the power to the V_{PD} pin remains on. The required timings to put the part into power-down and an overview of this mode are given in Section 4.2 in the MCS-96 Architecture Chapter.

A 'key' can be written into power-down RAM while the part is running. This key can be checked on reset to determine if it is a start-up from power-down or a complete cold start. In this way the validity of the power-down RAM can be verified. The length of this key determines the probability that this procedure will work, however, there is always a statistical chance that the RAM will power up with a replica of the key.

Under most circumstances, the power-fail indicator which is used to initiate a power-down condition must come from the unfiltered, unregulated section of the power supply. The power supply must have sufficient storage capacity to operate the 8096BH until it has completed its reset operation.

2.0 DRIVE AND INTERFACE LEVELS

There are five types of I/O lines on the 8096BH. Of these, two are inputs and three are outputs. All of the pins of the same type have the same current/voltage characteristics. Some of the control input pins, such as XTAL1 and $\overline{\text{RESET}}$, may have slightly different characteristics. These pins are discussed in Section 1.

While discussing the characteristics of the I/O pins some approximate current or voltage specifications will be given. The exact specifications are available in the lastest version of the data sheet that corresponds to the part being used.

2.1 Quasi-Bidirectional Ports

The Quasi-Bidirectional pins of Port 1, Port 2.6, and Port 2.7 have both input and output port configurations. They have three distinct states; low impedance current sink (Q2), low impedance current source (Q1), and high impedance current source (Q3). As a low impedance current sink, the pin has specification of sinking up to around 0.5 mA, while staying below 0.45 volts. The pin is placed in this condition by writing a '0' to the SFR (Special Function Register) controlling the pin.

Examine Figure 10. When a '1' is written to the SFR location controlling the pin, Q1 (a low impedance MOSFET pullup) is turned on for one state time, then it is turned off and the depletion pullup holds the line at a logical '1' state. The low-impedance pullup is used to shorten the rise time of the pin, and has current source capability on the order of 100 times that of the depletion pullup.

While the depletion mode pullup is the only device on, the pin may be used as an input with a leakage of around 100 microamps from 0.45 volts to VCC. It is ideal for use with TTL or CMOS chips and may even be used directly with switches. However if the switch option is used, certain precautions should be taken. It is important to note that any time the pin is read, the value returned will be the value on the pin, not the value placed in the control register. This could cause logical operations made directly on these pins to indavertently write a 0 to pins being used as inputs. In order to perform logical operations on a port where a quasi-bidirectional pin is an input, it is necessary to guarantee that the bit associated with the input pin is always a one when writing to the port.

2.2 Quasi-Bidirectional Hardware Connections

When using the quasi-bidirectional ports as inputs tied to switches, series resistors may be needed if the ports will be written to internally after the part is initialized. The amount of current sourced to ground from each pin is tyically 20 mA or more. Therefore, if all 8 pins are tied to ground, 160 mA will be sourced. This is equivalent to instantaneously doubling the power used by the chip and may cause noise in some applications.

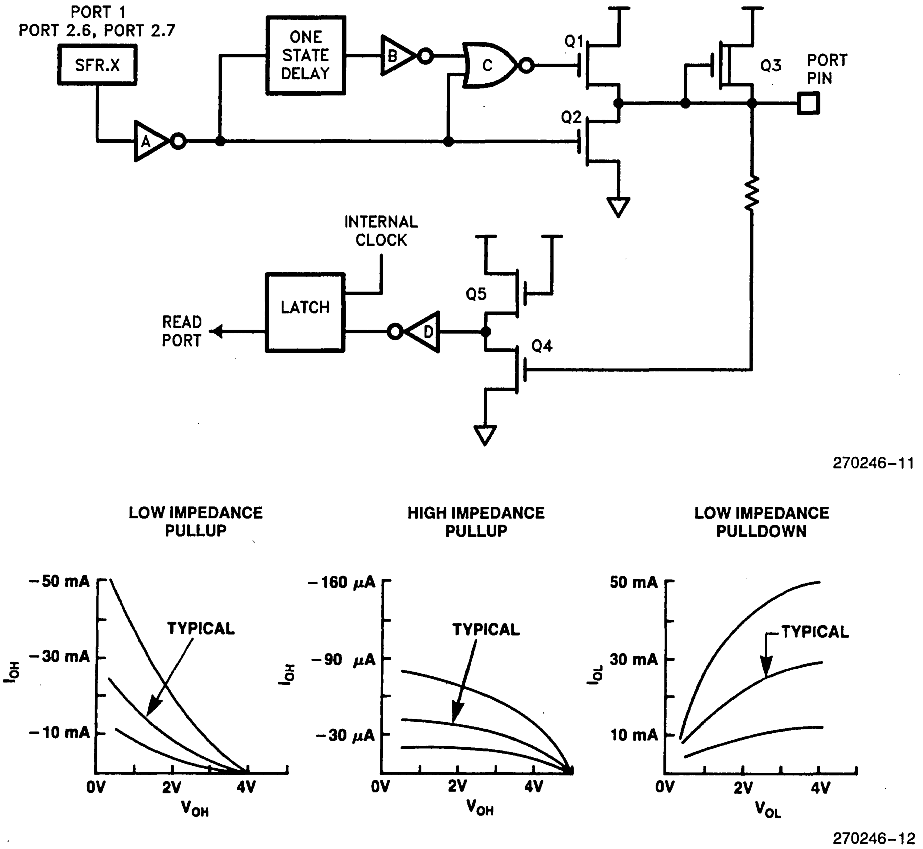
This potential problem can be solved in hardware or software. In software, never write a zero to a pin being used as an input.

In hardware, a 1K resistor in series with each pin will limit current to a reasonable value without impeding the ability to override the high impedance pullup. If all 8 pins are tied together a 120Ω resistor would be reasonable. The problem is not quite as severe when the inputs are tied to electronic devices instead of switches, as most external pulldowns will not hold 20 mA to 0.0 volts.

Writing to a Quasi-Bidirectional Port with electronic devices attached to the pins requires special attention. Consider using P1.0 as an input and trying to toggle P1.1 as an output:

```
ORB  IOPORT1, #00000001B ; Set P1.0
                         ; for input
XORB IOPORT1, #00000010B ; Complement
                         ; P1.1
```

The first instruction will work as expected but two problems can occur when the second instruction executes. The first is that even though P1.1 is being driven



NOTE:
These graphs show typical pin capabilities, they are not guaranteed specifications

Figure 10. Quasi-Bidirectional Port

high by the 8096 it is possible that it is being held low externally. This typically happens when the port pin is used to drive the base of an NPN transistor which in turn drives whatever there is in the outside world which needs to be toggled. The base of the transistor will clamp the port pin to the transistor's Vbe above ground, typically 0.7V. The 8096 will input this value as a zero even if a one has been written to the port pin. When this happens the XORB instruction will always write a one to the port pin's SFR and the pin will not toggle.

The second problem, which is related to the first, is that if P1.0 happens to be driven to a zero when Port 1 is read by the XORB instruction, then the XORB will write a zero to P1.0 and it will no longer be useable as an input.

The first situation can best be solved by the external driver design. A series resistor between the port pin and the base of the transistor often works by bringing up the voltage present on the port pin. The second case can be taken care of in the software fairly easily:

```
LDB  AL, IOPORT1
XORB AL, #010B
ORB  AL, #001B
STB  AL, IOPORT1
```

A software solution to both cases is to keep a byte in RAM as an image of the data to be output to the port; any time the software wants to modify the data on the port it can then modify the image byte and copy it to the port.

If a switch is used on a long line connected to a quasi-bidirectional pin, a pullup resistor is recommended to reduce the possibility of noise glitches and to decrease the rise time of the line. On extremely long lines that are handling slow signals, a capacitor may be helpful in addition to the resistor to reduce noise.

2.3 Input Only Ports

The high impedance input pins on the 8096BH have an input leakage of a few microamps and are predominantly capacitive loads on the order of 10 pF.

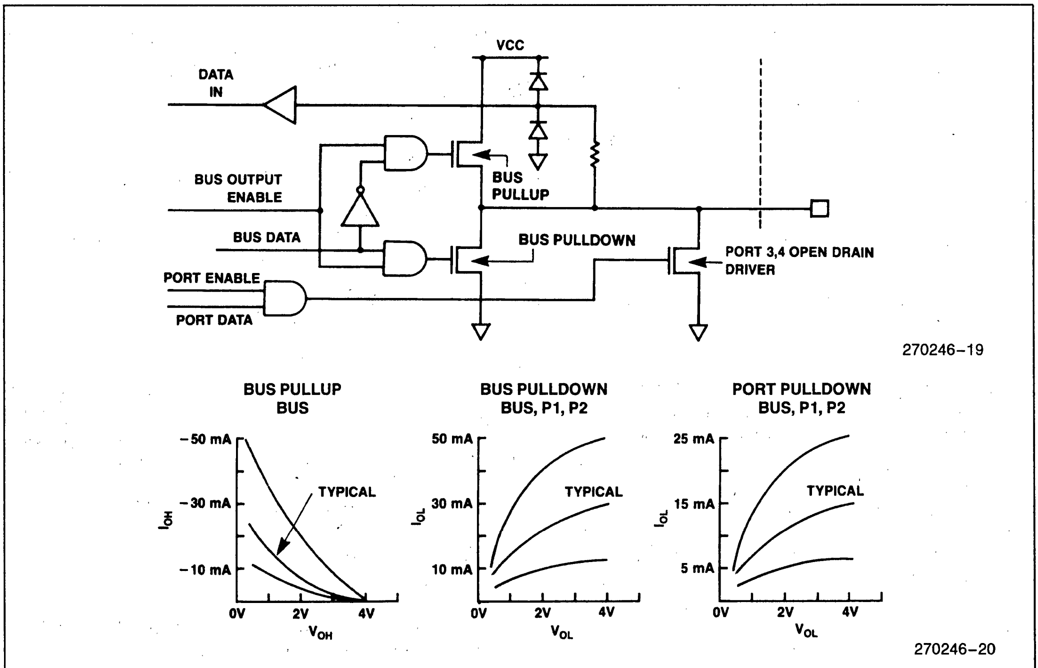
Port 0 pins are special in that they may individually be used as digital inputs, or as analog inputs. A Port 0 pin being used as a digital input acts as the high impedance input ports just described. However, Port 0 pins being used as analog inputs are required to provide current to the internal sample capacitor when a conversion begins. This means that the input characteristics of a pin will change if a conversion is being done on that pin. See Section 3. In either case, if Port 0 is to be used as analog or digital I/O, it will be necessary to provide power to this port through the V_{REF} pin.

Port 0 pins on 8X9X devices being used as analog inputs are required to provide current to an internal capacitor multiple times while a conversion is in progress. This means that the input characteristics of a Port 0 pin will change if a conversion is being done on that pin. See Section 3.

2.4 Open Drain Ports

Ports 3 and 4 on the 8096BH are open drain ports. There is no pullup when these pins are used as I/O ports. These pins have different characteristics when used as bus pins as described in the next section. A diagram of the output buffers connected to Ports 3 and 4 and the bus pins is shown in Figure 11.

When Ports 3 and 4 are to be used as inputs, or as bus pins, they must first be written with a '1'. This will put the ports in a high impedance mode. When they are used as outputs, a pullup resistor must be used externally. The sink capability of these pins is on the order of 0.4 milliamps so the total pullup current to the pin must be less than this. A 15K pullup resistor will source a maximum of 0.33 milliamps, so it would be a reasonable value to choose if no other circuits with pullups were connected to the pin.



NOTE:
These graphs show typical pin capabilities, they are not guaranteed specifications.

Figure 11. Bus and Port 3 and 4 Pins

2.5 HSO Pins, Control Outputs and Bus Pins

The control outputs and HSO pins have output buffers with the same output characteristics as those of the bus pins. Included in the category of control outputs are: TXD, RXD (in Mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, and $\overline{WR}$. The bus pins have 3 states: output high, output low, and high impedance input. As a high output, the pins are specified to source around 200 μA to 2.4 volts, but the pins can source on the order of ten times that value in order to provide the fast rise times. When used as a low output, the pins can sink around 2 mA at 0.45 volts, and considerably more as the voltage increases. When in the high impedance state, the pin acts as a capacitive load with a few microamps of leakage. Figure 11 shows the internal configuration of a bus pin.

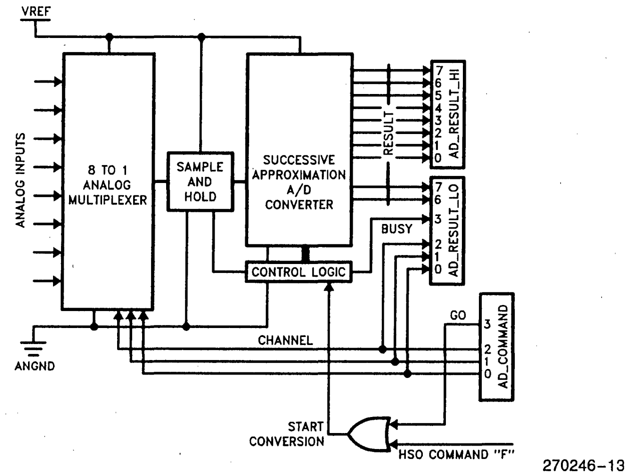
3.0 ANALOG INPUTS

The on-chip A/D converter of the 8096BH can be used to digitize analog inputs while analog outputs can be generated with either the chip's PWM output or HSO unit. This section describes the analog input suggestions. See Section 4 for analog output.

The 8096BH's Integrated A/D converter includes an eight channel analog multiplexer, sample-and-hold circuit and 10-bit analog to digital converter (Figure 12). The 8096BH can therefore select one of eight analog inputs to convert, sample-and-hold the input voltage and convert the voltage into a digital value. Each conversion takes 22 microseconds, including the time required for the sample-hold (with XTAL1 = 12 MHz). The method of conversion is successive approximation.

Section 3.6 contains the definitions of numerous terms used in connection with the A/D converter.

The A/D converter of 8X9X devices does not contain a Sample-and-Hold and has a conversion time of 42 μs (12 MHz on XTAL1). Section 3.5 discusses the differences.



NOTE:
1. Sample and hold not on 8X9X devices.

Figure 12. A/D Converter Block Diagram

3.1 A/D Overview

The conversion process is initiated by the execution of HSO command 0FH, or by writing a one to the GO Bit in the A/D Control Register. Either activity causes a start conversion signal to be sent to the A/D converter control logic. If an HSO command was used, the conversion process will begin when Timer 1 increments. This aids applications attempting to approach spectrally pure sampling, since successive samples spaced by equal Timer 1 delays will occur with a variance of about ±50 ns (assuming a stable clock on XTAL1). However, conversions initiated by writing a one to the ADCON register GO Bit will start within three state times after the instruction has completed execution resulting in a variance of about 0.75 μs (XTAL1 = 12 MHz).

Once the A/D unit receives a start conversion signal, there is a one state time delay before sampling (sample delay) while the successive approximation register is reset and the proper multiplexer channel is selected. After the sample delay, the multiplexer output is connected to the sample capacitor and remains connected for four state times (sample time). After this four state time "sample window" closes, the input to the sample capacitor is disconnected from the multiplexer so that changes on the input pin will not alter the stored charge while the conversion is in progress. The comparator is then auto-zeroed and the conversion begins. The sample delay and sample time uncertainties are each approximately ±50 ns, independent of clock speed.

To perform the actual analog-to-digital conversion the 8096BH implements a successive approximation algorithm. The converter hardware consists of a 256-resistor ladder, a comparator, coupling capacitors and a 10-bit successive approximation register (SAR) with logic that guides the process. The resistor ladder provides 20 mV steps (V_{REF} = 5.12V), while capacitive coupling is used to create 5 mV steps within the 20 mV ladder voltages. Therefore, 1024 internal reference voltages are available for comparison against the analog input to generate a 10-bit conversion result.

A successive approximation conversion is performed by comparing a sequence of reference voltages, to the analog input, in a binary search for the reference voltage that most closely matches the input. The ½ full scale reference voltage is the first tested. This corresponds to a 10-bit result where the most significant bit is zero, and all other bits are ones (0111.1111.11b). If the analog input was less than the test voltage, bit 10 of the SAR is left a zero, and a new test voltage of ¼ full scale (0011.1111.11b) is tried. If this test voltage was lower than the analog input, bit 9 of the SAR is set and bit 8 is cleared for the next test (0101.1111.11b). This binary search continues until 10 tests have occurred, at which time the valid 10-bit conversion result resides in the SAR where it can be read by software.

The total number of state times required is 88 for a 10-bit conversion. Attempting to short-cycle the 10-bit conversion process by reading A/D results before the done bit is set is not recommended.

3.2 A/D Interface Suggestions

The external interface circuitry to an analog input is highly dependent upon the application, and can impact converter characteristics. In the external circuit's design, important factors such as input pin leakage, sample capacitor size and multiplexer series resistance from the input pin to the sample capacitor must be considered.

For the 8096BH, these factors are idealized in Figure 13. The external input circuit must be able to charge a sample capacitor (C_S) through a series resistance (R_I) to an accurate voltage given a D.C. leakage (I_L). On the 8096BH, C_S is around 2 pF, R_I is around 5 KΩ and I_L is specified as 3 μA maximum. In determining the necessary source impedance R_S, the value of V_{BIAS} is not important.

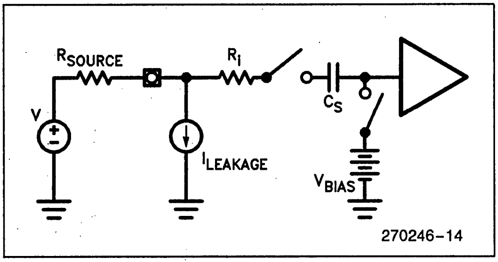


Figure 13. Idealized A/D Sampling Circuitry

External circuits with source impedances of 1 KΩ or less will be able to maintain an input voltage within a tolerance of about ±0.61 LSB (1.0 KΩ × 3.0 μA = 3.0 mV) given the D.C. leakage. Source impedances above 2 KΩ can result in an external error of at least one LSB due to the voltage drop caused by the 1 μA leakage. In addition, source impedances above 25 KΩ may degrade converter accuracy as a result of the internal sample capacitor not being fully charged during the 1 μs (12 MHz clock) sample window.

It is important to note that source impedance requirements relax if an external capacitor of sufficient size is attached directly to the analog input pin. Since the internal sample capacitor is around 2.0 pF, an external 0.005 μF capacitor (2048 × 2.0 pF) should provide an accurate input voltage to ±0.5 LSB. If there is leakage on the capacitor, the value of the capacitor must be increased to compensate for the leakage. For example, assuming just the 3 μA D.C. leakage caused by the 8096BH, 0.6 mV (less than 0.15 LSB) will be lost from a 0.005 μF capacitor in 1 μs. Therefore, the capacitor connected externally to the pin should be at least 0.005 μF if the source impedance is too large to provide the needed accuracy on its own. However, if the external signal changes slowly, it is recommended that the largest acceptable capacitance be used, given the input signal frequency.

Placing an external capacitor on each analog input will also reduce the sensitivity to noise, as the capacitor combines with series resistance in the external circuit to form a low-pass filter. In practice, one should include a small series resistance prior to the external capacitor on the analog input pin and choose the largest capacitor value practical, given the frequency of the signal being converted. This provides a low-pass filter on the input, while the resistor will also limit input current during over-voltage conditions.

Figure 14 shows a simple analog interface circuit based upon the discussion above. The circuit in the figure also provides limited protection against over-voltage conditions on the analog input. Should the input voltage inappropriately drop significantly below ground, diode D2 will forward bias at about 0.8 DCV. Since the specification of the pin has an absolute maximum low voltage rating of −0.3 DCV, this will leave about 0.5 DCV across the 270Ω resistor, or about 2.0 mA of current. This should limit current to a safe amount. *However, before any circuit is used in an actual application, it should be thoroughly analyzed for applicability to the specific problem at hand.*

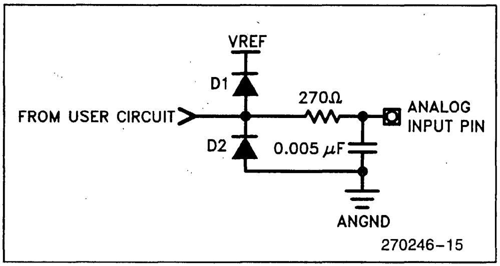


Figure 14. Suggested A/D Input Circuit

3.3 Analog References

Reference supply levels strongly influence the absolute accuracy of the conversion. For this reason, it is recommended that the ANGND pin be tied to the two V_{SS} pins as close to the chip as possible with minimum trace length. Bypass capacitors should also be used between V_{REF} and ANGND. ANGND should be within about a tenth of a volt V_{SS}. V_{REF} should be well regulated and used only for the A/D converter. The V_{REF} supply can be between 4.5V and 5.5V and needs to be able to source around 5 mA. Figure 6 shows all of these connections.

Note that if only ratiometric information is desired, V_{REF} can be connected to V_{CC}. In addition, V_{REF} and ANGND must be connected even if the A/D converter is not being used. Remember that Port 0 receives its power from the V_{REF} and ANGND pins even when it is used as digital I/O.

3.4 The A/D Transfer Function

The conversion result is a 10-bit ratiometric representation of the input voltage, so the numerical value obtained from the conversion will be:

$$\text{INT}\ [1023 \times (V_{IN} - \text{ANGND})/(V_{REF} - \text{ANGND})].$$

This produces a stair-stepped transfer function when the output code is plotted versus input voltage (see Figure 15). The resulting digital codes can be taken as simple ratiometric information, or they can be used to provide information about absolute voltages or relative voltage changes on the inputs. The more demanding the application is on the A/D converter, the more important it is to fully understand the converter's operation. For simple applications, knowing the absolute error of the converter is sufficient. However, closing a servo-loop with analog inputs necessitates a detailed understanding of an A/D converter's operation and errors.

The errors inherent in an analog-to-digital conversion process are many: quantizing error; zero offset; full-scale error; differential non-linearity; and non-linearity. These are "transfer function" errors related to the A/D converter. In addition, converter temperature drift, V_{CC} rejection, sample-hold feedthrough, multiplexer off-isolation, channel-to-channel matching and random noise should be considered. Fortunately, one "Absolute Error" specification is available which describes the sum total of all deviations between the actual conversion process and an ideal converter. However, the various sub-components of error are important in many applications. These error components are described in Section 3.5 and in the text below where ideal and actual converters are compared.

An unavoidable error simply results from the conversion of a continuous voltage to an integer digital representation. This error is called quantizing error, and is always ±0.5 LSB. Quantizing error is the only error seen in a perfect A/D converter, and is obviously present in actual converters. Figure 15 shows the transfer function for an ideal 3-bit A/D converter (i.e. the Ideal Characteristic).

Note that in Figure 15 the Ideal Characteristic possesses unique qualities: it's first code transition occurs when the input voltage is 0.5 LSB; it's full-scale code transition occurs when the input voltage equals the full-scale reference minus 1.5 LSB; and it's code widths are all exactly one LSB. These qualities result in a digitization without offset, full-scale or linearity errors. In other words, a perfect conversion.

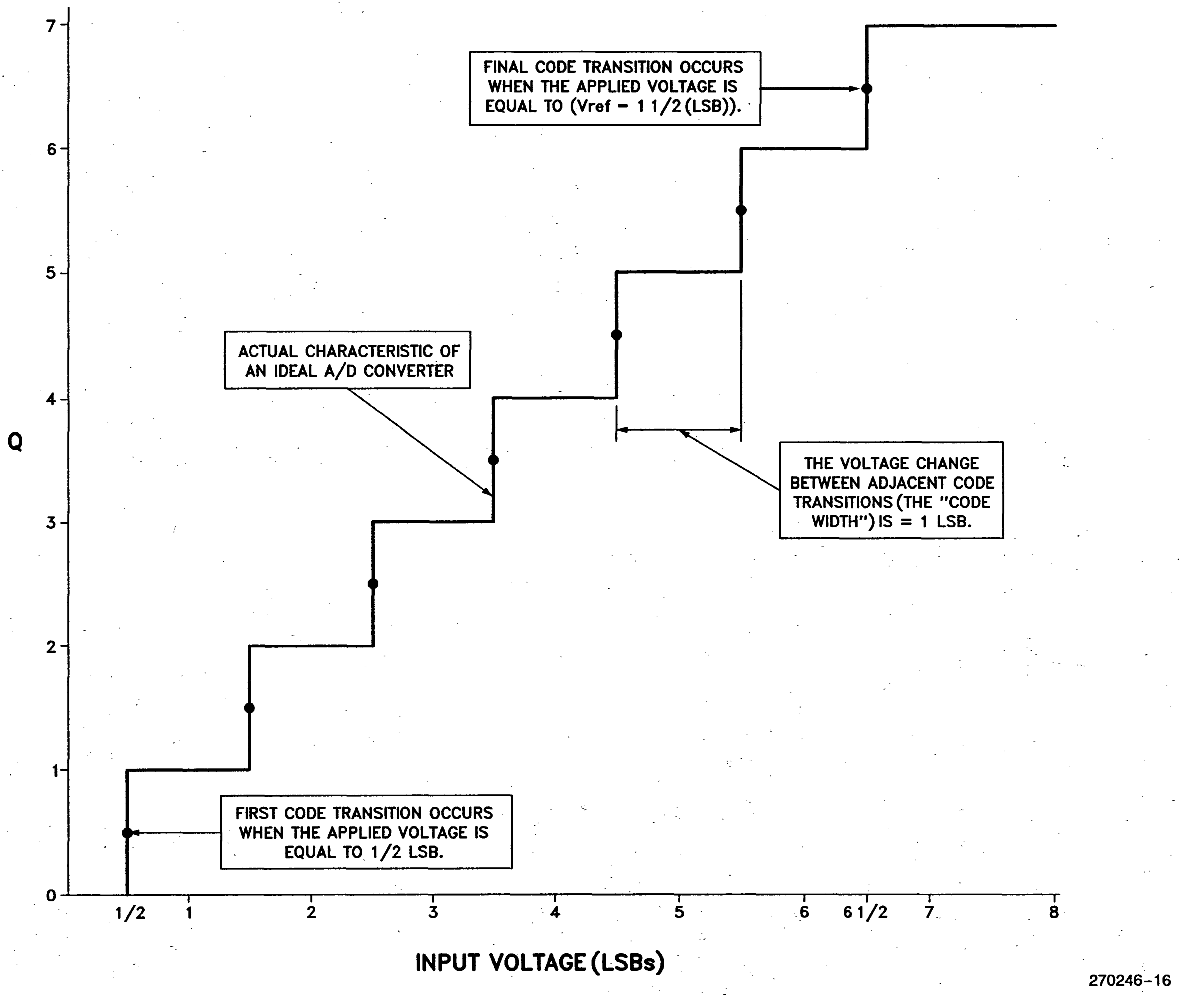
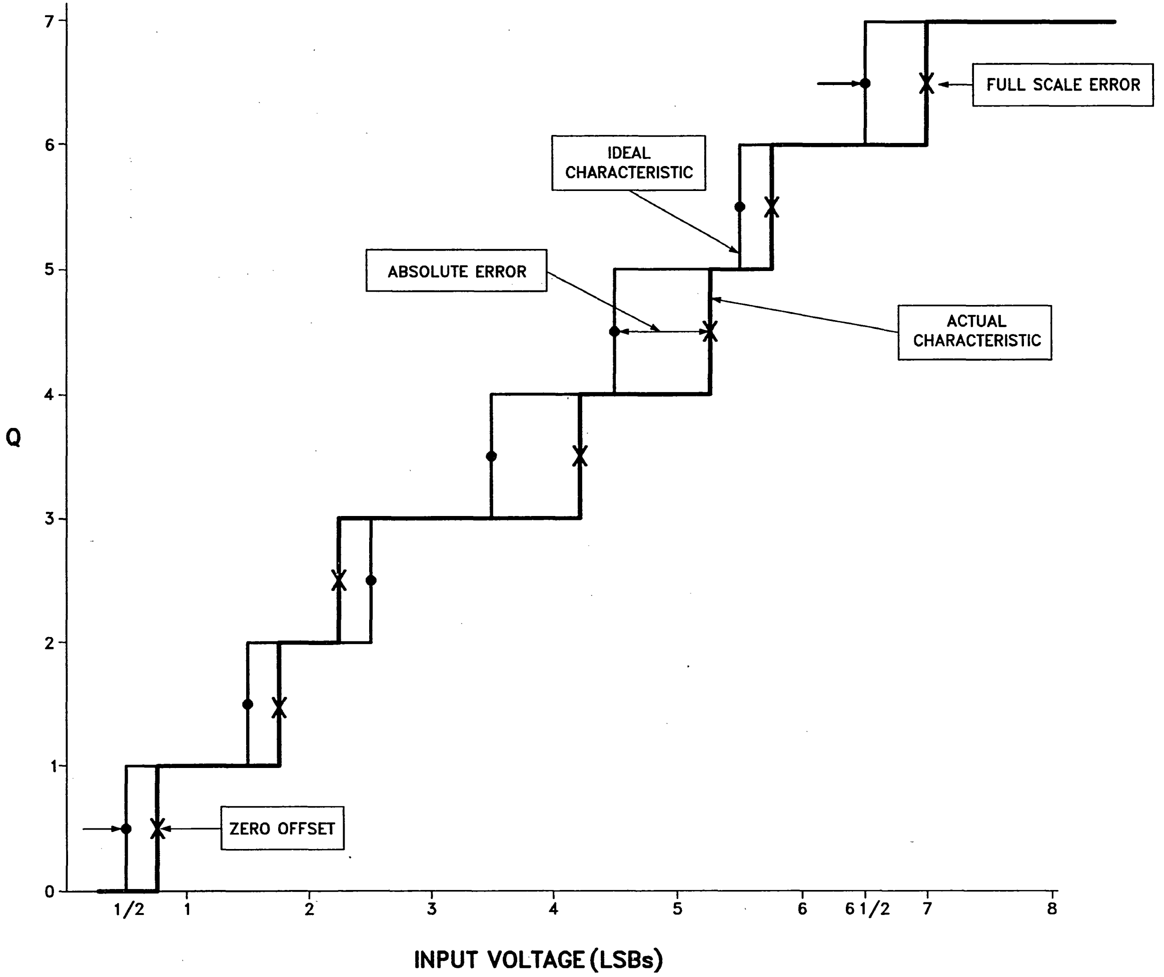
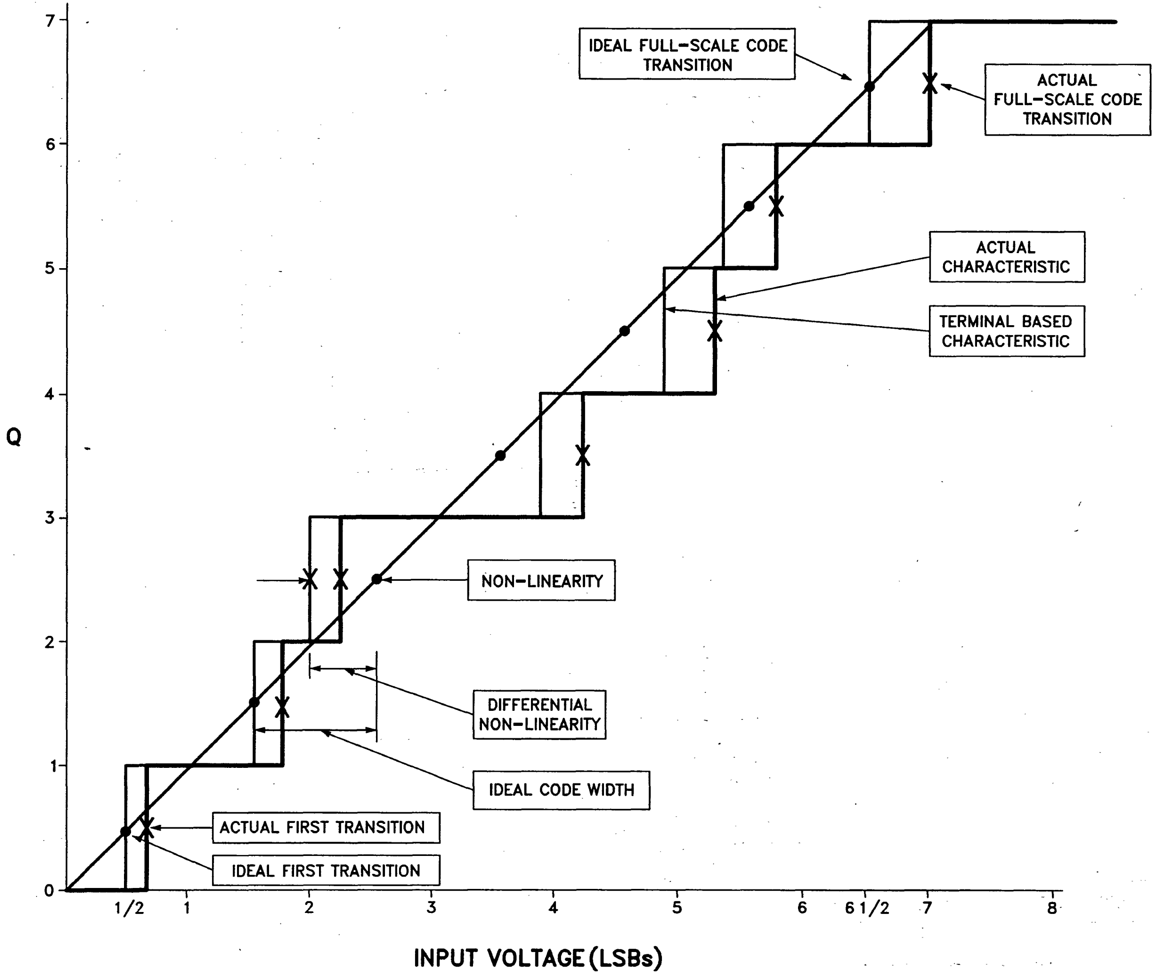


Figure 15. Ideal A/D Characteristic



270246–17

Figure 16. Actual and Ideal Characteristics



270246-18

Figure 17. Terminal Based Characteristic

Figure 16 shows an Actual Characteristic of a hypothetical 3-bit converter, which is not perfect. When the Ideal Characteristic is overlaid with the imperfect characteristic, the actual converter is seen to exhibit errors in the location of the first and final code transitions and code widths. The deviation of the first code transition from ideal is called "zero offset", and the deviation of the final code transition from ideal is "full-scale error". The deviation of the code widths from ideal causes two types of errors. Differential Non-Linearity and Non-Linearity. Differential Non-Linearity is a local linearity error measurement, whereas Non-Linearity is an overall linearity error measure.

Differential Non-Linearity is the degree to which actual code widths differ from the ideal one LSB width. Differential Non-Linearity gives the user a measure of how much the input voltage may have changed in order to produce a one count change in the conversion result. Non-Linearity is the worst case deviation of code transitions from the corresponding code transitions of the Ideal Characteristic. Non-Linearity describes how much Differential Non-Linearities could add up to produce an overall maximum departure from a linear characteristic. If the Differential Non-Linearity errors are too large, it is possible for an A/D converter to miss codes or exhibit non-monotonicity. Neither behavior is desireable in a closed-loop system. A converter has no missed codes if there exists for each output code a unique input voltage range that produces that code only. A converter is monotonic if every subsequent code change represents an input voltage change in the same direction.

Differential Non-Linearity and Non-Linearity are quantified by measuring the Terminal Based Linearity Errors. A Terminal Based Characteristic results when an Actual Characteristic is shifted and rotated to eliminate zero offset and full-scale error (see Figure 17). The Terminal Based Characteristic is similar to the Actual Characteristic that would be seen if zero offset and full-scale error were externally trimmed away. In practice, this is done by using input circuits which include gain and offset trimming. In addition, VREF on the 8096BH could also be closely regulated and trimmed within the specified range to affect full-scale error.

Other factors that affect a real A/D Converter system include sensitivity to temperature, failure to completely reject all unwanted signals, multiplexer channel dissimilarities and random noise. Fortunately these effects are small.

Temperature sensitivities are described by the rate at which typical specifications change with a change in temperature.

Undesired signals come from three main sources. First, noise on VCC—VCC Rejection. Second, input signal changes on the channel being converted after the sample window has closed—Feedthrough. Third, signals applied to channels not selected by the multiplexer—Off-Isolation.

Finally, multiplexer on-channel resistances differ slightly from one channel to the next causing Channel-to-Channel Matching errors, and random noise in general results in Repeatability errors.

3.5 8X9X A/D Converter Differences

The 8X9X A/D Converter does not have an internal Sample-and-Hold, and the conversion time is 168 state times (42 μs with 12 MHz clock). These differences primarily influence the interface circuitry and the rate at which sampling can be done.

For the 8X9X, the idealized circuit in Figure 13 is still applicable. The only real difference is that the capacitor labeled C_S has a smaller value on 8X9X devices, but it is charged 10 times during a conversion. Since the actual C_S on 8X9X parts is about 0.5 pF, an effective C_S of 5.0 pF (10 × 0.5 pF) can be used as the internal capacitance that must be charged during a conversion. The value of R_I and I_L are nominally 5 kΩ and 3 μA respectively.

Given these values, external circuits with source impedances of 1 KΩ or less will be able to maintain an input voltage within a tolerance of about ±0.6 LSB (1.0 KΩ × 3.0 μA = 3.0 mV) given the D.C. leakage. Source impedances above 2 KΩ will induce an external error of at least one LSB due to the voltage drop caused by the 3 μA leakage. In addition, source impedances above 25 KΩ may degrade converter accuracy as a result of inadequate internal capacitor charging.

On 8X9X devices, the analog input is sampled 10 times while a conversion is in progress. Therefore, the input must remain stable so that conversion accuracy is not affected. If the input signal could vary significantly while a conversion is in progress, an external capacitor attached directly to the analog input pin could be used as a Sample-and-Hold. Since the internal capacitance is around 5.0 pF, an external 0.01 μF capacitor (2048 × 5.0 pF) should provide an accurate input voltage to ±0.5 LSB. If there is leakage on the capacitor, the value of the capacitor must be increased to compensate for the leakage. For example, assuming just the 3 μA D.C leakage caused by the 8X9X, 1 mV (less than 0.25 LSB) will be lost from a 0.15 μF capacitor in 42 μs. Therefore, the capacitor connected externally to the pin should be at least 0.2 μF. However, if the external signal changes slowly relative to the conversion time (168 state times), it is recommended that the largest acceptable capacitance be used given the input signal frequency.

Figure 14 shows a simple interface which could be applicable to 8X9X devices if the size of the capacitor attached to the analog input pin is increased to a value greater than 0.2 µF. The circuit in the figure also provides limited protection against over-voltage conditions on the analog inputs. However, before any circuit is used in an actual application, it should be thoroughly analyzed for applicability to the specific problem at hand.

3.6 A/D Glossary of Terms

Figures 15, 16 and 17 display many of these terms.

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An Actual Characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversion under the same conditions.

BREAK-BEFORE-MAKE—The property of a multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g. the converter will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE CENTER—The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION—The point at which the converter changes from an output code of Q, to a code of Q+1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK—See "Off-Isolation".

D.C. INPUT LEAKAGE—Leakage current to ground from an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic of a converter.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D converter after the sample window closes.

FULL SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at VIN = 0.5 LSB, its last code transition at VIN = (VREF − 1.5 LSB) and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB—LEAST SIGNIFICANT BIT: The voltage value corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For a 10-bit converter with a reference voltage of 5.12 volts, one LSB is 5.0 mV. Note that this is different than digital LSBs, since an uncertainty of two LSB, when referring to an A/D converter, equals 10 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 20 mV.)

MONOTONIC—The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES—For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY—The maximum deviation of code transitions of the terminal based characteristic from the corresponding code transitions of the ideal characteristics.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY—The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY—The variation in the Sample Delay.

SAMPLE TIME—The time that the sample window is open.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION—An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An Actual Characteristic which as been rotated and translated to remove zero offset and full-scale error.

VCC REJECTION—Attenuation of noise on the VCC line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

4.0 ANALOG OUTPUTS

Analog outputs can be generated by two methods, either by using the PWM output or the HSO. Either device will generate a rectangular pulse train that varies in duty cycle and (for the HSO only) period. If a smooth analog signal is desired as an output, the rectangular waveform must be filtered.

In most cases this filtering is best done after the signal is buffered to make it swing from 0 to 5 volts since both of the outputs are guaranteed only to TTL levels. A block diagram of the type of circuit needed is shown in Figure 18. By proper selection of components, accounting for temperature and power supply drift, a highly accurate 8-bit D to A converter can be made using either the HSO or the PWM output. Figure 19 shows two typical circuits. If the HSO is used the accuracy could be theoretically extended to 16-bits, however the temperature and noise related problems would be extremely hard to handle.

When driving some circuits it may be desirable to use unfiltered Pulse Width Modulation. This is particularly true for motor drive circuits. The PWM output can be used to generate these waveforms if a fixed period on the order of 64 µs is acceptable. If this is not the case then the HSO unit can be used. The HSO can generate a variable waveform with a duty cycle variable in up to 65536 steps and a period of up to 131 milliseconds. Both of these outputs produce TTL levels.

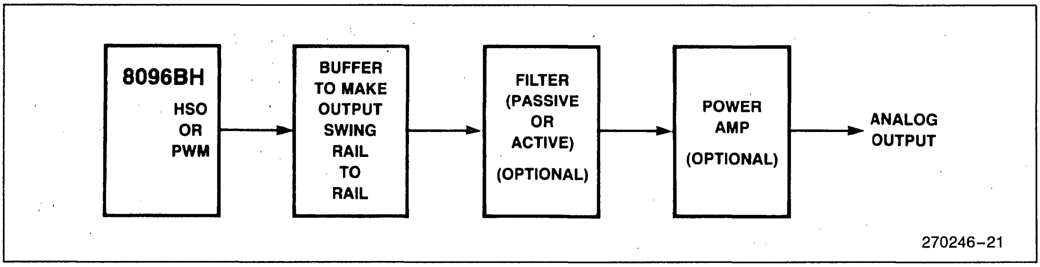
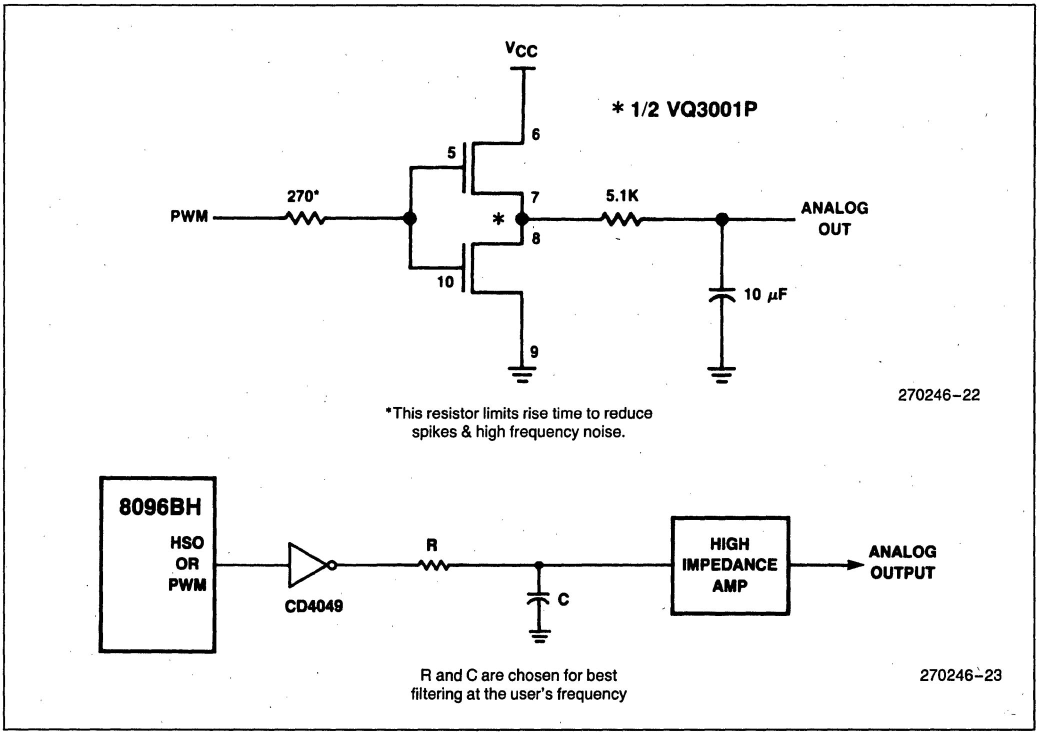


Figure 18. D/A Buffer Block Diagram



*This resistor limits rise time to reduce spikes & high frequency noise.

R and C are chosen for best filtering at the user's frequency

Figure 19. Buffer Circuits for D/A

5.0 I/O TIMINGS

The I/O pins on the 8096BH are sampled and changed at specific times within an instruction cycle. The changes occur relative to the internal phases shown in Figure 4. Note that the delay from XTAL1 to the internal clocks range from about 30 ns to 100 ns over process and temperature. Signals generated by internal phases are further delayed by 5 ns to 15 ns. The timings shown in this section are idealized; no propagation delay factors have been taken into account. Designing a system that depends on an I/O pin to change within a window of less than 50 ns using the information in this section is not recommended.

5.1 HSO Outputs

Changes in the HSO lines are synchronized to Timer 1. All of the external HSO lines due to change at a certain value of a timer will change just pior to the incrementing of Timer 1. This corresponds to an internal change during Phase B every eight state times. From an external perspective the HSO pin should change just prior to the rising edge of CLKOUT and be stable by its falling edge. Information from the HSO can be latched on the CLKOUT falling edge. Internal events can occur anytime during the 8 state time window.

Timer 2 is synchronized to increment no faster than Timer 1, so there will always be at least one incrementing of Timer 1 while Timer 2 is at a specific value.

5.2 HSI Input Sampling

The HSI pins are sampled internally once each state time. Any value on these pins must remain stable for at least 1 full state time to guarantee that it is recognized. The actual sample occurs at the end of Phase A, which, due to propagation delay, is just after the rising edge of CLKOUT. Therefore, if information is to be synchronized to the HSI it should be latched-in on CLKOUT falling. The time restriction applies even if the divide by eight mode is being used. If two events occur on the same pin within the same 8 state time window, only one of the events will be recorded. If the events occur on different pins they will always be recorded, regardless of the time difference. The 8 state time window, (i.e. the amount of time during which Timer 1 remains constant), is stable to within about 20 ns. The window starts roughly around the rising edge of CLKOUT, however this timing is very approximate due to the amount of internal circuitry involved.

5.3 Standard I/O Port Pins

Port 0 is different from the other digital ports in that it is actually part of the A/D converter. The port is sampled once every state time, however, sampling is not synchronized to Timer 1. If this port is used, the input signal on the pin must be stable one state time before the reading of the SFR.

On 8X9X devices, Port 0 is sampled every eight state times (the same frequency at which the comparator is charged-up during an A/D conversion). This 8 state time counter is not synchronized with Timer 1. If this port is used, the input signal on the pin must be stable 8 state times prior to reading the SFR.

Port 1 and Port 2 have quasi-bidirectional I/O pins. When used as inputs the data on these pins must be stable one state time prior to reading the SFR. This timing is also valid for the input-only pins of Port 2 and is similar to the HSI in that the sample occurs just after the rising edge of CLKOUT. When used as outputs, the quasi-bidirectional pins will change state shortly after CLKOUT falls. If the change was from '0' to a '1' the low impedance pullup will remain on for one state time after the change.

Ports 3 and 4 are addressed as off-chip memory-mapped I/O. The port pins will change state shortly after the rising edge of CLKOUT. When these pins are used as Ports 3 and 4 they are open drains, their structure is different when they are used as part of the bus. See Section 10.4 of the MCS-96 Architecture chapter. Additional information on port reconstruction is available in Section 7.8 of this chapter.

6.0 SERIAL PORT TIMINGS

The serial port on the 8096BH was designed to be compatible with the 8051 serial port. Since the 8051 uses a divide by 2 clock and the 8096BH uses a divide by 3, the serial port on the 8096BH had to be provided with its own clock circuit to maximize its compatibility with the 8051 at high baud rates. This means that the serial port itself does not know about state times. There is circuitry which is synchronized to the serial port and to the rest of the 8096BH so that information can be passed back and forth.

The baud rate generator is clocked by either XTAL1 or T2CLK. Because T2CLK needs to be synchronized to the XTAL1 signal its speed must be limited to $1/16$ that of XTAL1. The serial port will not function during the time between the consecutive writes to the baud rate register. Section 11.4 of the MCS-96 Architecture chapter discusses programming the baud rate generator.

6.1 Mode 0

Mode 0 is the shift register mode. The TXD pin sends out a clock train, while the RXD pin transmits or receives the data. Figure 20 shows the waveforms and timing. Note that the port starts functioning when a '1' is written to the REN (Receiver Enable) bit in the serial port control register. If REN is already high, clearing the RI flag will start a reception.

In this mode the serial port can be used to expand the I/O capability of the 8096BH by simply adding shift registers. A schematic of a typical circuit is shown in Figure 21. This circuit inverts the data coming in, so it must be reinverted in software. The enable and latch connections to the shift registers can be driven by decoders, rather than directly from the low speed I/O ports, if the software and hardware are properly designed.

6.2 Mode 1 Timings

Mode 1 operation of the serial port makes use of 10-bit data packages, a start bit, 8 data bits and a stop bit. The transmit and receive functions are controlled by separate shift clocks. The transmit shift clock starts when the baud rate generator is initialized, the receive shift clock is reset when a '1 to 0' transition (start bit) is received. The transmit clock may therefore not be in sync with the receive clock, although they will both be at the same frequency.

The TI (Transmit Interrupt) and RI (Receive Interrupt) flags are set to indicate when operations are complete. TI is set when the last data bit of the message has been sent, not when the stop bit is sent. If an attempt to send another byte is made before the stop bit is sent the port will hold off transmission until the stop bit is complete. RI is set when 8 data bits are received, not when the stop bit is received. Note that when the serial port status register is read both TI and RI are cleared.

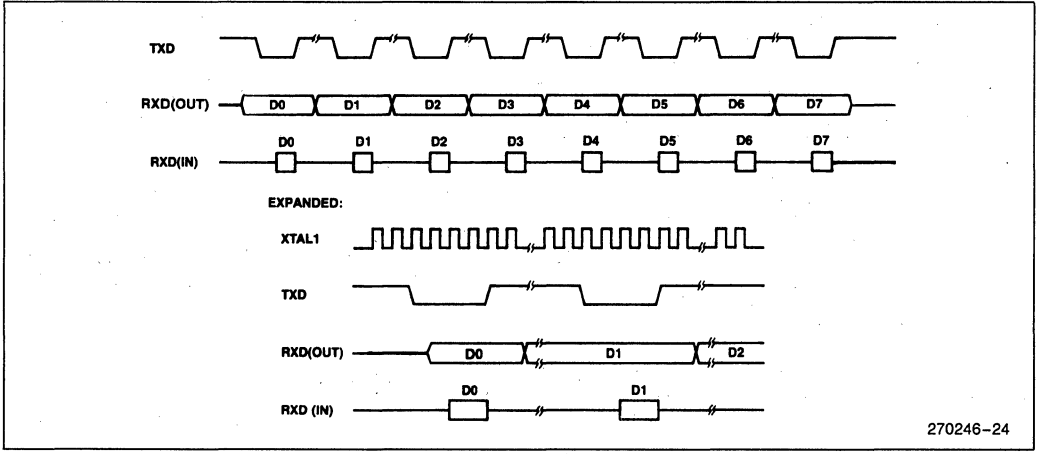


Figure 20. Serial Port Timings in Mode 0

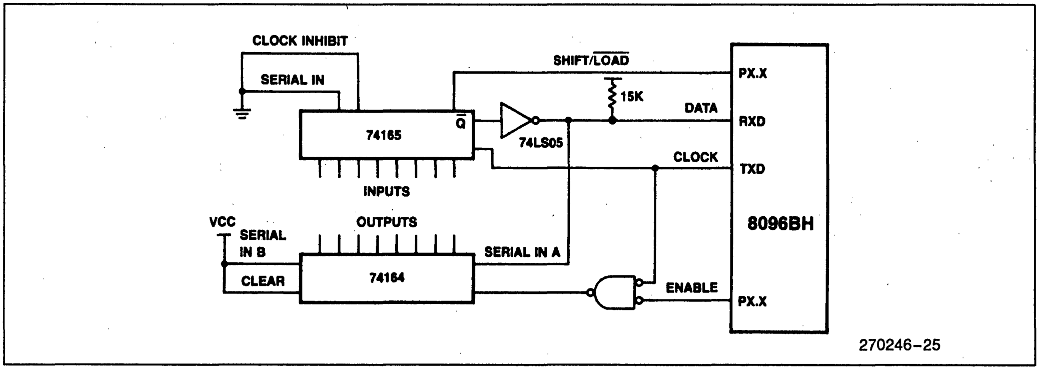


Figure 21. Mode 0 Serial Port Example

Caution should be used when using the serial port to connect more than two devices in half-duplex, (i.e. one wire for transmit *and* receive). If the receiving processor does not wait for one bit time after RI is set before starting to transmit, the stop bit on the link could be squashed. This could cause a problem for other devices listening on the link.

6.3 Mode 2 and 3 Timings

Modes 2 and 3 operate in a manner similar to that of Mode 1. The only difference is that the data is now made up of 9 bits, so 11-bit packages are transmitted and received. This means that TI and RI will be set on the 9th data bit rather than the 8th. The 9th bit can be used for parity or multiple processor communications (see Section 11 of the MCS-96 Architecture chapter).

7.0 BUS TIMING AND MEMORY INTERFACE

7.1 Bus Functionality

The 8096BH has a multiplexed (address/data) bus which can be dynamically configured to have an 8-bit or 16-bit data width. There are control lines to demultiplex the bus (ALE or $\overline{\text{ADV}}$), indicate reads ($\overline{\text{RD}}$), indicate writes ($\overline{\text{WRL}}$ and $\overline{\text{WRH}}$, or $\overline{\text{WR}}$ with $\overline{\text{BHE}}$ and AD0), and a signal to indicate accesses that are for an instruction fetch (INST). Section 3.5 of the MCS-96 Architecture chapter contains an overview of the bus operation.

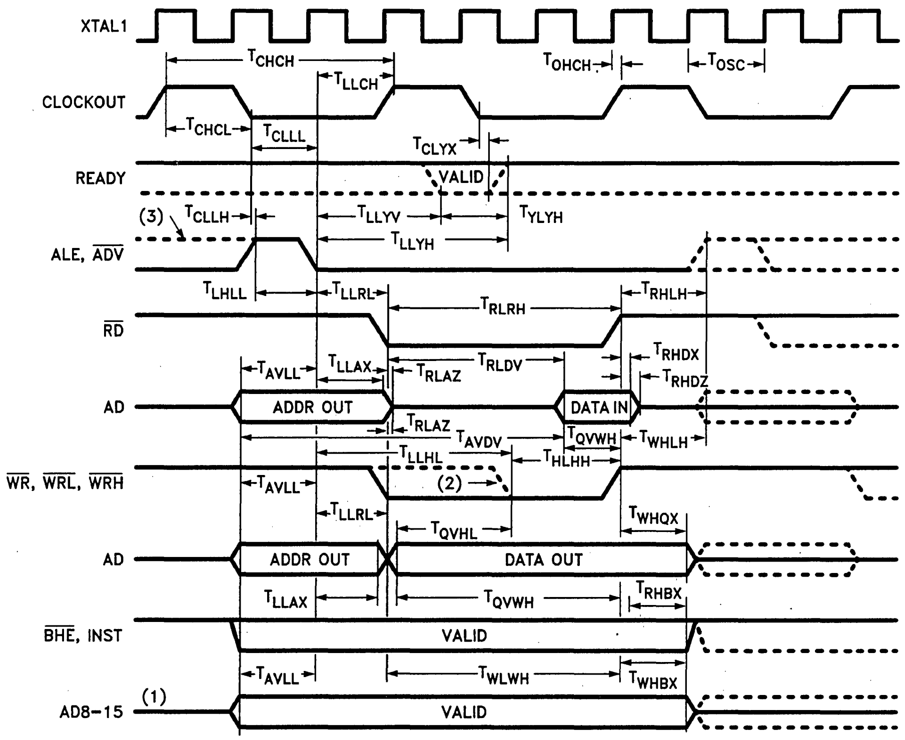
On 8X9X devices only the 16-bit multiplexed bus is available. In addition, on 8X9X devices the $\overline{\text{WRL}}$ and $\overline{\text{WRH}}$ signals are not available and the functionality of the $\overline{\text{BHE}}$ and INST lines differs from the 8X9XBH devices. See the data sheet of the device that you use.

7.2 Timing Specifications

Figure 22 shows the timing of the bus signals and data lines. Please refer to the latest data sheet for the exact device you are using to ensure that your system is designed to the proper specifications. The major timing specifications are described in Figure 23.

7.3 READY Line Usage

When the processor has to address a memory location that cannot respond within the standard specifications, it is necessary to use the READY line to generate wait states. When the READY line is held low, the processor waits in a loop for the line to come high or until the



270246–26

NOTES:

1. When ALE function is selected, the signal is always high for TLHLL. When $\overline{\text{ADV}}$ function is selected, the signal is high for at least TLHLL.
2. The dotted line applies for all 8-bit bus writes and 16-bit bus writes with the write strobe mode selected.
3. 8-bit bus only.

Figure 22. Bus Signal Timings

number of inserted wait states is equal to the limit set in the Chip Configuration Register (see Section 2 of the MCS-96 Architecture chapter). There is a maximum time that the READY line can be held low without risking a processor malfunction due to dynamic nodes that have not been refreshed during the wait states. This time is shown as TYLYH in the data sheet.

In most cases the READY line is brought low after the address is decoded and it is determined that a wait state is needed. It is very likely that some addresses, such as those addressing memory mapped peripherals, would need wait states, and others would not. The READY line must be stable within the TLLYV specification after ALE falls or the processor could lock-up. There is no requirement as to when READY may go high, as long as the maximum READY low time (TYLYH) is not violated. To ensure that only one wait state is inserted it is necessary to provide external circuitry which brings $\overline{\text{READY}}$ high TLLYH after the falling edge of ALE/$\overline{\text{ADV}}$, or program the Chip Configuration Register to select a Ready Control limit of one.

Internally, the chip latches $\overline{\text{READY}}$ on the first falling edge of Phase A after ALE/$\overline{\text{ADV}}$ falls. Phase A is buffered and brought out externally as CLOCKOUT, so CLOCKOUT is a delayed Phase A. If a 1 is seen, the bus cycle proceeds uninterrupted with no wait state insertions. If a 0 is seen, one wait state (3 Tosc) is inserted.

If a wait state is inserted, READY is internally latched on the next rising edge of Phase A. If a 1 is found the bus cycle resumes with the net impact being the insertion of one wait state. If a 0 is seen, a second wait state is inserted.

The READY pin is again latched on the next rising edge of CLOCKOUT if two wait states were inserted. If the chip sees a 1, the bus cycle is resumed with the result being an insertion of two wait states. If another 0 is seen, a third wait state is inserted in the bus cycle and

Definitions of A.C. timing specifications differ slightly on 8X9X devices. See the data sheet for the part you are using for more information.

Tosc—Oscillator Period, one cycle time on XTAL1.

Timings the Memory System Must Meet

TLLYH—ALE/$\overline{\text{ADV}}$ low to READY high: Maximum time after ALE/$\overline{\text{ADV}}$ falls until READY is brought high to ensure no more wait states. If this time is exceeded unexpected wait states may result. Nominally 1 Tosc + 3 Tosc × number of wait states desired.

TLLYV—ALE/$\overline{\text{ADV}}$ low to READY low: Maximum time after ALE/$\overline{\text{ADV}}$ falls until READY must be valid. If this time is exceeded the part could malfunction necessitating a chip reset. Nominally 2 Tosc periods.

TCLYX—READY hold after CLOCKOUT low: Minimum time that the value on the READY pin must be valid after CLOCKOUT falls. The minimum hold time is always zero nanoseconds.

TYLYH—READY low to READY high: Maximum time the part can be in the not-ready state. If it is exceeded, the 8096BH dynamic nodes which hold the current instruction may 'forget' how to finish the instruction.

TAVDV—ADDRESS valid to DATA valid: Maximum time that the memory has to output valid data after the 8096BH outputs a valid address. Nominally, a maximum of 5 Tosc periods.

TAVGV—ADDRESS valid to BUSWIDTH valid: Maximum time after ADDRESS becomes valid until BUSWIDTH must be valid. Nominally less than 2 Tosc periods.

TLLGV—ALE/$\overline{\text{ADV}}$ low to BUSWIDTH valid: Maximum time after ALE/$\overline{\text{ADV}}$ is low until BUSWIDTH must be valid. If this time is exceeded the part could malfunction necessitating a chip reset. Nominally less than 1 Tosc.

TLLGX—BUSWIDTH hold after ALE/$\overline{\text{ADV}}$ low: Minimum time that BUSWIDTH must be valid after ALE/$\overline{\text{ADV}}$ is low Nominally 1 Tosc.

TRLDV—$\overline{\text{READ}}$ low to DATA valid: Maximum time that the memory has to output data after $\overline{\text{READ}}$ goes low. Nominally, a maximum of 3 Tosc periods.

TRHDZ—$\overline{\text{READ}}$ high to DATA float: Time after $\overline{\text{READ}}$ is high until the memory must float the bus. The memory signal can be removed as soon as $\overline{\text{READ}}$ is not low, and must be removed within the specified maximum time from when $\overline{\text{READ}}$ is high. Nominally a maximum of 1 Tosc period.

TRHDX—DATA hold after $\overline{\text{READ}}$ goes high: Minimum time that memory must hold input DATA valid after $\overline{\text{RD}}$ is high. The hold time minimum is always zero nanoseconds.

Figure 23. Timing Specification Explanations

Timings the 8096 Will Provide

TOHCH—XTAL1 high to CLOCKOUT high: Delay from the rising edge of XTAL1 to the resultant rising edge on CLOCKOUT. Needed in systems where the signal driving XTAL1 is also used as a clock for external devices. Typically 50 to 100 nanoseconds.

TCHCH—CLKOUT high to CLKOUT high: The period of CLKOUT and the duration of one state time. Always 3 Tosc average, but individual periods could vary by a few nanoseconds.

TCHCL—CLKOUT high to CLKOUT low: Nominally 1 Tosc period.

TCLLH—CLKOUT low to ALE high: A help in deriving other timings. Typically plus or minus 5 ns to 10 ns.

TCLVL—CLOCKOUT low to ALE/$\overline{\text{ADV}}$ low: A help in deriving other timings. Nominally 1 Tosc.

TLLCH—ALE/$\overline{\text{ADV}}$ low to CLKOUT high: Used to derive other timings, nominally 1 Tosc period.

TLHLL—ALE/$\overline{\text{ADV}}$ high to ALE/$\overline{\text{ADV}}$ low: ALE/$\overline{\text{ADV}}$ high time. Useful in determining ALE/$\overline{\text{ADV}}$ rising edge to ADDRESS valid time. Nominally 1 Tosc period for ALE and 1 Tosc for $\overline{\text{ADV}}$ with back-to-back bus cycles.

TAVLL—ADDRESS valid to ALE/$\overline{\text{ADV}}$ low: Length of time ADDRESS is valid before ALE/$\overline{\text{ADV}}$ falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLAX—ALE/$\overline{\text{ADV}}$ low to ADDRESS invalid: Length of time ADDRESS is valid after ALE/$\overline{\text{ADV}}$ falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLRL—ALE/$\overline{\text{ADV}}$ low to $\overline{\text{READ}}$ or $\overline{\text{WRITE}}$ low: Length of time after ALE/$\overline{\text{ADV}}$ falls before $\overline{\text{RD}}$ or $\overline{\text{WR}}$ fall. Could be needed to ensure that proper memory decoding takes place before it is output enabled. Nominally 1 Tosc period.

TLLHL—ALE/$\overline{\text{ADV}}$ low to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ low: Minimum time after ALE/$\overline{\text{ADV}}$ is low that the write strobe signals will go low. Could be needed to ensure that proper memory decoding takes place before it is output enabled. Nominally 2 Tosc periods.

TRLRH—$\overline{\text{READ}}$ low to $\overline{\text{READ}}$ high: $\overline{\text{RD}}$ pulse width, nominally 1 Tosc period.

TRHLH—$\overline{\text{READ}}$ high to ALE/$\overline{\text{ADV}}$ high: Time between $\overline{\text{RD}}$ going inactive and next ALE/$\overline{\text{ADV}}$, also used to calculate time between $\overline{\text{RD}}$ inactive and next ADDRESS valid. Nominally 1 Tosc period.

TRHBX—$\overline{\text{READ}}$ high to INST, $\overline{\text{BHE}}$, AD8–15 Inactive: Minimum time that the INST and BHE lines will be valid after $\overline{\text{RD}}$ goes high. Also the minimum time that the upper eight address lines (8-bit bus mode) will remain valid after $\overline{\text{RD}}$ goes high. Nominally 1 Tosc.

TWHBX—$\overline{\text{WRITE}}$ high to INST, $\overline{\text{BHE}}$, AD8–15 Inactive: Minimum time that the INST and $\overline{\text{BHE}}$ lines will be valid after $\overline{\text{WR}}$ goes high. Also the minimum time that the upper eight address lines (8-bit bus mode) will remain valid after $\overline{\text{WR}}$ goes high. Nominally 1 Tosc.

TWLWH—$\overline{\text{WRITE}}$ low to $\overline{\text{WRITE}}$ high: Write pulse width, nominally 3 Tosc periods.

THLHH—$\overline{\text{WRL}}$, $\overline{\text{WRH}}$ low to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ high: Write strobe signal pulse width. Nominally 2 Tosc periods.

TQVHL—OUTPUT valid to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ low: Minimum time that OUTPUT data is valid prior to write strobes becoming active. Needed for interfacing to memories that read data on the falling edge of write. Nominally 1 Tosc.

TQVWH—OUTPUT valid to $\overline{\text{WRITE}}$ high: Time that the OUTPUT data is valid before $\overline{\text{WR}}$ is high. Nominally 3 Tosc periods.

TWHQX—$\overline{\text{WRITE}}$ high to OUTPUT not valid: Time that the OUTPUT data is valid after $\overline{\text{WR}}$ is high. Nominally 1 Tosc period.

TWHLH—$\overline{\text{WRITE}}$ high to ALE/$\overline{\text{ADV}}$ high: Time between write high and next ALE/$\overline{\text{ADV}}$, also used to calculate the time between $\overline{\text{WR}}$ high and next ADDRESS valid. Nominally 2 Tosc periods.

Figure 23. Timing Specification Explanations (Continued)

the READY pin is again latched on the following rising edge of CLOCKOUT. If internal Ready Control is not used, the READY line must at this point be a 1 to ensure proper operation.

On 8X9X devices there is no internal Ready Control, therefore, external circuitry must completely control the insertion of wait states into 8X9X bus cycles.

7.4 INST Line Usage

The INST (Instruction) line is high during bus cycles that are for an instruction fetch and low for any other bus cycle. The INST signal (not present on 48-pin versions) can be used with a logic analyzer to debug a system. In this way it is possible to determine if a fetch was for instructions or data, making the task of tracing the program much easier.

On 8X9X devices the INST line is high during the output of an address that is for an instruction fetch. It is low during the same time for any other memory access. At any other time it is not valid.

7.5 BUSWIDTH Pin Usage

The BUSWIDTH pin is a control input which determines the width of the bus access in progress. BUSWIDTH is sampled after the rising edge of the first CLOCKOUT after ALE/$\overline{\text{ADV}}$ goes low. If a one is seen, the bus access progresses as a 16-bit cycle. If a zero is seen, the bus access progresses as an 8-bit cycle. The BUSWIDTH setup and hold timing requirements appear in the data sheet.

The BUSWIDTH pin can be overridden by causing the BUS WIDTH SELECT bit in the Chip Configuration Register (CCR) to be zero. This will permanently select an 8-bit bus width. However, if the BUS WIDTH SELECT bit in the CCR is a one, the BUSWIDTH pin determines the bus width. See Section 3.5 of the MCS-96 Architecture chapter. Since the BUSWIDTH pin is not available on 48-pin parts, the BUS WIDTH SELECT bit in the CCR determines bus width.

On 8X9X devices, the 8-bit bus is not available, the CCR does not exist and the BUSWIDTH pin is named the TEST pin. The TEST pin is used for testing purposes and should be tied to VCC in application circuits.

7.6 Address Decoding

The multiplexed bus of the 8096BH must be demultiplexed before it can be used. This can be done with two 74LS373 transparent latches for an 8096BH in 16-bit bus mode, or one 74LS373 for an 8096BH in 8-bit bus mode. As explained in Section 3.5 of the MCS-96 Architecture chapter, the latched address signals will be referred to as MA0 through MA15 (Memory Address), and the data lines will be called MD0 through MD15 (Memory Data).

Since the 8096BH can make accesses to memory for either bytes or words, it is necessary to have a way of determining the type of access desired when the bus is 16-bits wide. For write cycles, the signals Write Low ($\overline{\text{WRL}}$) and Write High ($\overline{\text{WRH}}$) are provided. $\overline{\text{WRL}}$ will go low during all word writes and during all byte writes to an even location. Similarly, $\overline{\text{WRH}}$ will go low during all word writes and during all byte writes to an odd location. During read cycles, an 8096BH in 16-bit bus mode will always do a word read of an even location. If only one byte of the word is needed, the chip discards the byte it does not need.

Since 8096BH memory accesses over an 8-bit wide bus are always bytes, only one write strobe is needed for write cycles. For this purpose the $\overline{\text{WRL}}$ signal was made to go low for all write cycles during 8-bit bus accesses. When a word operation is requested, the bus controller performs two byte-wide bus cycles.

In many cases it may be desirable to have a write signal with a longer pulse width than $\overline{\text{WRL}}/\overline{\text{WRH}}$. The Write ($\overline{\text{WR}}$) line of the 8096BH is an alternate control signal that shares a pin with $\overline{\text{WRL}}$ and is only available in 16-bit bus mode. $\overline{\text{WR}}$ is nominally one Tosc longer than the $\overline{\text{WRL}}/\overline{\text{WRH}}$ signals, but goes low for any write cycle. Therefore it is necessary to decode for the type of write (byte or word) desired.

The Byte High Enable ($\overline{\text{BHE}}$) signal and MA0 can be used for this purpose. $\overline{\text{BHE}}$ is an alternate control sig-

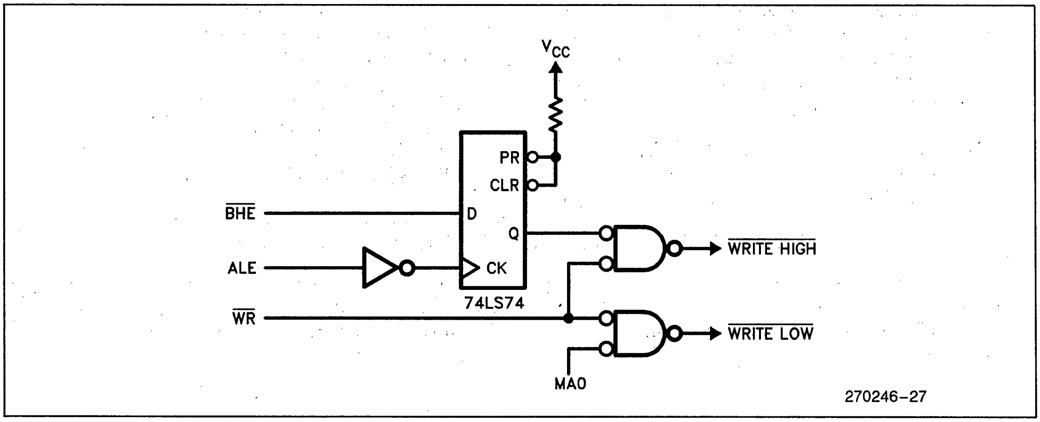


Figure 24. Decoding $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ to Generate $\overline{\text{WriteLow}}$ and $\overline{\text{WriteHigh}}$

nal that shares a pin with $\overline{\text{WRH}}$. When $\overline{\text{BHE}}$ is low, the high byte of the 16-bit bus is enabled. When MA0 is low, the lower byte is enabled. When MA0 is low and $\overline{\text{BHE}}$ is low, both bytes are enabled. Figure 24 shows how to use $\overline{\text{WR}}$, $\overline{\text{BHE}}$ and $\overline{\text{MA0}}$ to decode bus accesses. It's important to note that this decoding inserts a delay in the write signal which must be considered in a system timing analysis.

On 8X9X devices, only the $\overline{RD}$, $\overline{WR}$ and $\overline{BHE}$ signals are available for bus control. This means that discriminating between byte and word bus accesses must be done by decoding $\overline{WR}$, $\overline{BHE}$ and MA0 as described above. Further, the $\overline{WR}$ signal on 8X9X devices is nominally the same width as the $\overline{WRL}$ and $\overline{WRH}$ signals. 8X9XBH devices (2 Tosc), and the $\overline{BHE}$ signal must be latched since it is valid only while the address is valid. See Figure 24 and the data sheet of the device that you use.

External memory systems for the 8096BH can be set up in many ways. Figures 25 through 28 show block diagrams of memory systems using an 8-bit bus with a single EPROM, using an 8-bit bus with RAM and EPROM, using a 16-bit bus with two external EPROMs and using a 16-bit bus in a RAM and ROM system.

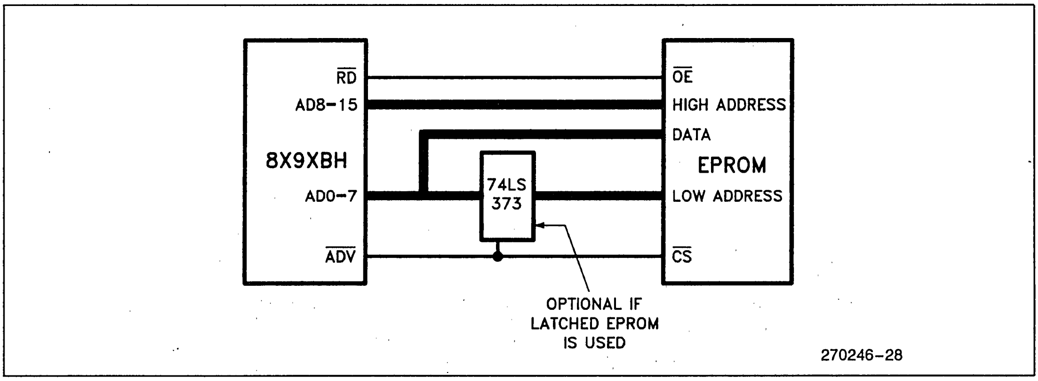


Figure 25. An 8-Bit Bus with EPROM Only

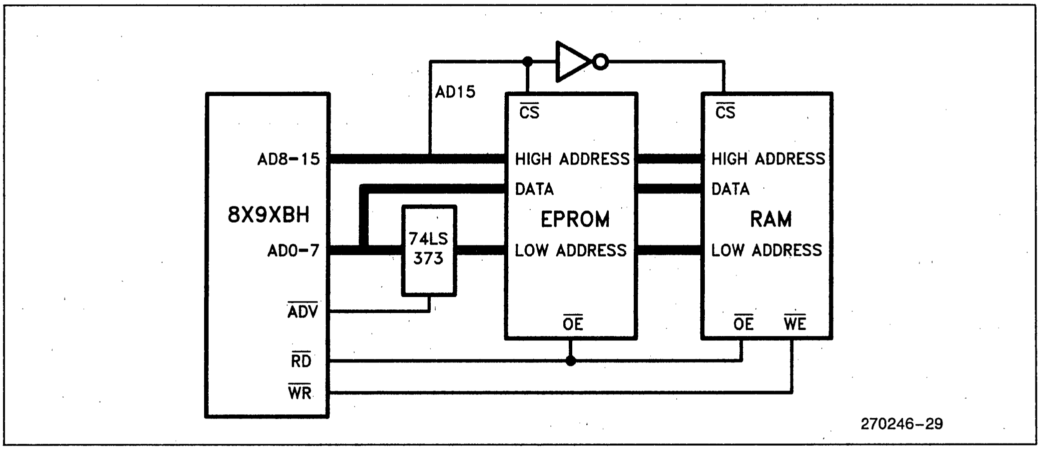


Figure 26. An 8-Bit Bus with EPROM and RAM

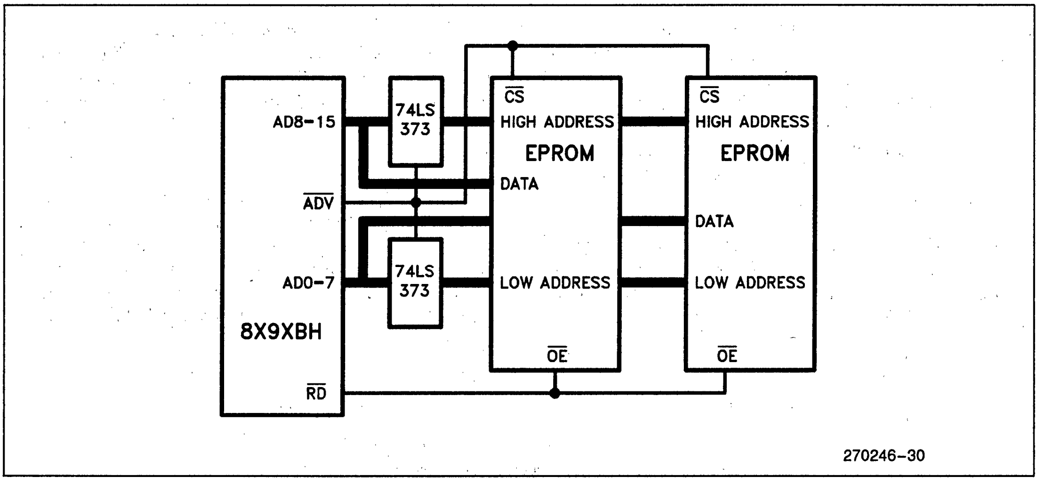


Figure 27. A 16-Bit Bus with EPROM Only

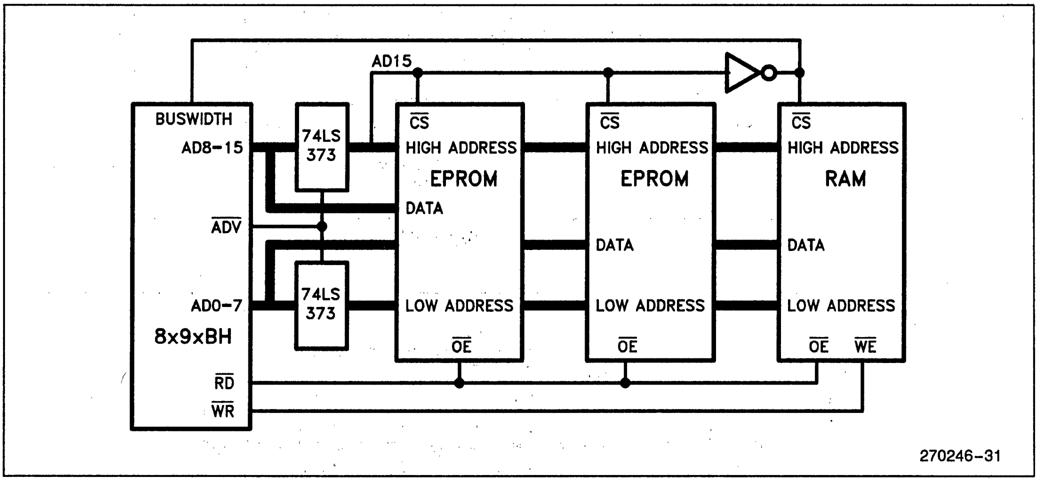


Figure 28. Memory System with Dynamic Bus Width

7.7 System Verification Example

To verify that a system such as the one in Figure 29 will work with the 8096BH, it is necessary to check all of the timing parameters. Let us examine this system one parameter at a time using representative 8096BH specifications. These specifications will be different for each part number and temperature range, so the results of this example must be modified based on the most recent data sheet for the specific part to be used.

The timings of signals that the processor and memory use are affected by the latch and buffer circuitry. The timings of the signal provided by the processor are delayed by various amounts of time. Similarly, the signals coming back from the memory are also delayed. The calculations involved in verifying this system follow:

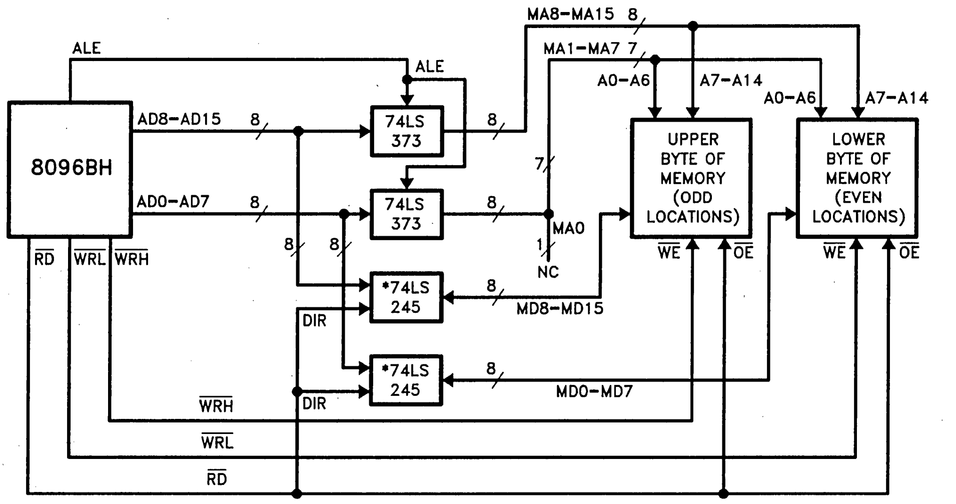
Address Valid Delay—30 nanoseconds

The address lines are delayed by passing them through the 74LS373s, this delay is specified at 18 ns after Address is valid or 30 ns after ALE is high. Since the signal may be limited by either the ALE timing or the Address timing, these two cases must be considered.

If Limited by ALE

Minimum ALE pulse width = Tosc − 25 (TLHLL)

Minimum Addr set-up to ALE falling = Tosc − 25 (TAVLL)



270246–32

*Required only in larger systems.

Figure 29. RAM/ROM Memory System

Therefore, in the worst case, ALE would occur 0 ns before Address valid.

Total delay from 8096BH Address stable to MA (Memory Address) stable would be:

ALE delay from address	− 0
74LS373 clock to output	30
	30 nanoseconds

If Limited by Address Valid

74LS373 Data Valid to Data Output = 18 nanoseconds

In the worst case, the delay in Address valid is controled by ALE and has a value of 30 nanoseconds.

Delay of Data Transfer to/from Processor—12 nanoseconds

The $\overline{\text{RD}}$ low to Data valid specification (TRLDV) is 3 Tosc − 50, (200 ns at 12 MHz). The 74LS245 is enabled by $\overline{\text{RD}}$ and has a delay of 40 ns from enable. The enable delay is clearly not a problem.

The 74LS245 is enabled for write, except during a read, so there is no enable delay to consider for write operations.

The Data In to Data Out delay of the 74LS245 is 12 ns.

CHARACTERISTICS OF A 12 MHz 8096BH SYSTEM WITH LATCHES

Required by system:

Address valid to Data in;

TAVDV	:	345.6 ns max. (5 Tosc − 70)
Address Delay	: −	30.0 ns maximum
Data Delay	: −	12.0 ns maximum
		303.6 ns maximum

Read low to Data in:

TRLDV	:	200.0 ns max. (3 Tosc − 50)
Address Delay	: −	00.0 ns maximum
Data Delay	: −	12.0 ns maximum
		188.0 ns maximum

Provided by system:

Address valid to Control;

TLLRL	:	63.3 ns min. (Tosc − 20)
TAVLL	:	158.3 ns min. (Tosc − 25)
Address Delay	: −	30.0 ns maximum
$\overline{\text{WR}}$ Delay	: −	00.0 ns minimum
		91.6 ns minimum

Write Pulse Width;

THLHH	:	146.6 ns min. (2 Tosc − 20)
		146.6 ns minimum

Data Setup to $\overline{WR}$ rising;

TQVWH	:	200.0 ns min. (3 Tosc − 50)
Data Delay	:	− 12.0 ns maximum
		188.0 ns minimum

Data Hold after $\overline{WR}$;

TWHQX	:	58.3 ns min. (Tosc − 25)
Data Delay	:	0.0 ns minimum (no spec)
		58.3 ns minimum

The two memory devices which are expected to be used most often with the 8096BH are the 2764 EPROM and the 2128 RAM. The system verification for the 2764 is simple.

2764 Tac

(Address valid to Output) < Address valid to Data in
250 ns < 303 ns O.K.

2764 Toe

(Output Enable to Output) < $\overline{\text{Read}}$ low to Data in
100 ns < 188 ns O.K.

These calculations assume no address decoder delays and no delays on the $\overline{RD}$ (OE) line. If there are delays in these signals the delays must be added to the 2764's timing.

The read calculations for the 2128 are similar to those for the 2764.

2128-20 Tac < Address valid to Data in
200 ns < 303 ns O.K.

2128-20 Toe < $\overline{\text{Read}}$ low to Data in
65 ns < 188 ns O.K.

The write calculation are a little more involved, but still straight-forward.

2128 Twp (Write Pulse) < Write Pulse Width
100 ns < 146 ns O.K.

2128 Tds (Data Setup) < Data Setup to $\overline{WR}$ rising
65 ns < 188 ns O.K.

2128 Tdh (Data Hold) < Data Hold after $\overline{WR}$
0 ns < 58 ns

All of the above calculations have been done assuming that no components are in the circuit except for those shown in Figure 29. If additional components are added, as may be needed for address decoding or memory bank switching, the calculations must be updated to reflect the actual circuit.

7.8 I/O Port Reconstruction

When a single-chip system is being designed using a multiple chip system as a prototype, it may be necessary to reconstruct I/O Ports 3 and 4 using a memory-mapped I/O technique. The circuit shown in Figure 30 provides this function. It can be attached to a 8096BH system which has the required address decoding and bus demultiplexing.

The output circuitry is basically just a latch that operates when 1FFEH or 1FFFH are placed on the MA lines. The inverters surrounding the latch create an open-collector output to emulate the open-drain output found on the 8096BH. The 'reset' line is used to set the ports to all 1's when the 8096BH is reset. It should be noted that the voltage and current characteristics of the port will differ from those of the 8096BH, but the basic functionality will be the same.

The input circuitry is just a bus transceiver that is addressed at 1FFEH or 1FFFH. If the ports are going to be used for either input or output, but not both, some of the circuitry can be eliminated.

8.0 NOISE PROTECTION TIPS

Designing controllers differs from designing other computer equipment in the area of noise protection. A microcontroller circuit under the hood of a car, in a photocopier, CRT terminal, or a high speed printer is subject to many types of electrical noise. Noise can get to the processor directly through the power supply, or it can be induced onto the board by electromagnetic fields. It is also possible for the PC board to find itself in the path of electrostatic discharges. Glitches and noise on the PC board can cause the processor to act unpredictably, usually by changing either the memory locations or the program counter.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The 8096BH has a Watchdog Timer which will reset the part if it fails to execute the software properly. The software should be set up to take advantage of this feature.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed all sorts of bad things can happen. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 8096BH instruction has 7 bytes) and a RST or jump to error routine instruction. This will help to ensure a speedy recovery should the processor have a glitch in the program flow.

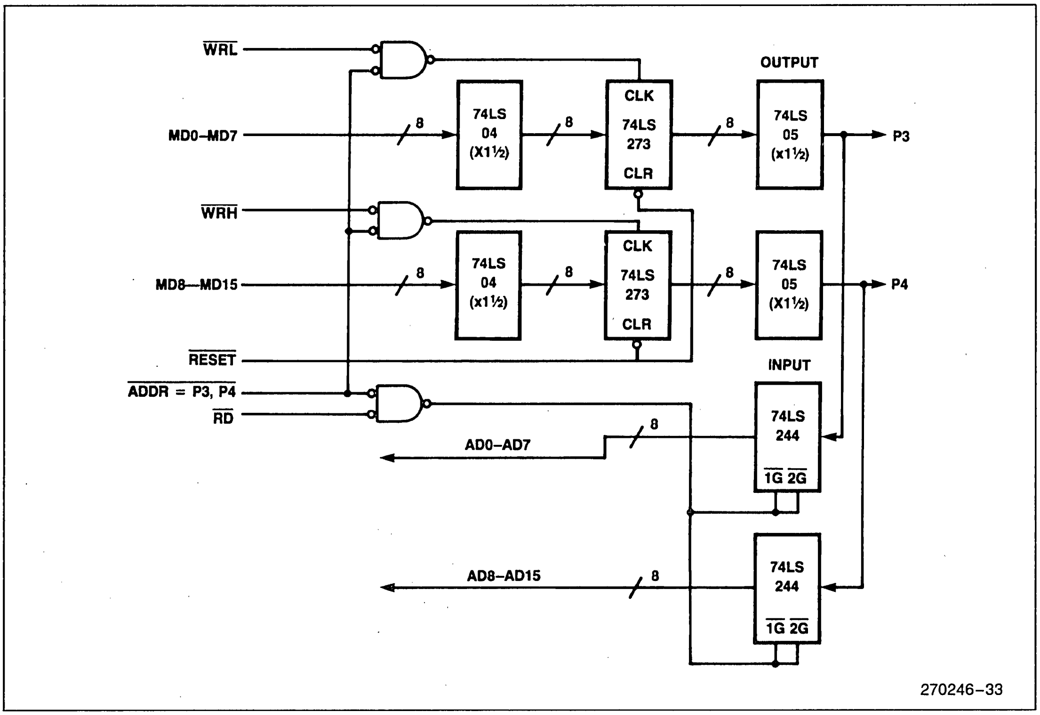


Figure 30. I/O Port Reconstruction

Many hardware solutions exist for keeping PC board noise to a minimum. Ground planes, gridded ground and VCC structures, bypass capacitors, transient absorbers and power busses with built-in capacitors can all be of great help. It is much easier to design a board with these features than to try to retrofit them later. Proper PC board layout is probably the single most important and, unfortunately, least understood aspect of project design. Minimizing loop areas and inductance, as well as providing clean grounds are very important. More information on protecting against noise can be found in the Application Note AP-125, "Designing Microcontroller Systems for Noisy Environments".

9.0 PACKAGING

The MCS-96 family of products is offered in many versions. They are available in 48-pin or 68-pin packages, with or without on-chip ROM/EPROM and with or without an A/D converter. A summary of the available options is shown in Figure 31.

The 48-pin versions are available in ceramic and plastic 48-pin Dual-In-Line package (DIP). The ceramic versions have part numbers with the prefix "C". The plastic versions have the prefix "P".

The 68-pin versions are available in a ceramic pin grid array (PGA), a plastic leaded chip carrier (PLCC) and a Type B leadless chip carrier (LCC). PGA devices have part numbers with the prefix "C". PLCC devices have the prefix "N". LCC devices have the prefix "R".

Specifications for the various members of the MCS-96 family are contained in the next chapter.

	ROMless		With ROM		With EPROM	
	68-pin	48-pin	68-pin	48-pin	68-pin	48-pin
Without A to D	8096		8396		8796	
With A to D	8097	8095	8397	8395	8797	8795

Figure 31. The MCS®-96 Family of Products

10.0 EPROM PROGRAMMING (8X9XBH ONLY)

The 879XBH contains 8K bytes of ultraviolet Erasable and Electrically Programmable Read Only Memory (EPROM) for internal storage. This memory can be programmed in a variety of ways—including at run-time under software control.

The EPROM is mapped into memory locations 2000H through 3FFFH if $\overline{EA}$ is a TTL high. However, applying +12.75V to $\overline{EA}$ when the chip is reset will place the 879XBH in EPROM Programming Mode. The Programming Mode has been implemented to support EPROM programming and verification.

When an 879XBH is in Programming Mode, special hardware functions are available to the user. These functions include algorithms for slave, gang and auto EPROM programming.

10.1 Programming the 879XBH

Three flexible EPROM programming modes are available on the 879XBH—auto, slave and run-time. These modes can be used to program 879XBHs in a gang, stand alone or run-time environment.

The Auto Programming Mode enables an 879XBH to program itself, and up to 15 other 879XBHs, with the 8K bytes of code beginning at address 4000H on its external bus. The Slave Mode provides a standard interface that enables any number of 879XBHs to be programmed by a master device such as an EPROM programmer. The Run-Time Mode allows individual EPROM locations to be programmed at run-time under complete software control.

In the Programming Mode, some I/O pins have been renamed. These new pin functions are used to determine the programming function that is performed, provide programming ALEs, provide slave ID numbers and pass error information. Figure 33 shows how the pins are renamed. Figure 34 describes each new pin function.

While in Programming Mode, PMODE selects the programming function that is performed (see Figure 32). When not in the Programming Mode, Run-Time programming can be done at any time.

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming Mode
0DH	Program Configuration Byte
0EH–0FH	Reserved

Figure 32. Programming Function PMODE Values

To guarantee proper execution, the pins of PMODE and SID must be in their desired state before the $\overline{RESET}$ pin is allowed to rise and reset the part. Once the part is reset, it is in the selected mode and should not be switched to another mode without a new reset sequence.

When $\overline{EA}$ selects the Programming Mode, the chip reset sequence loads the CCR from the Programming Chip Configuration Byte (PCCB). This is a separate EPROM location that is not mapped under normal operation. PCCB is only important when programming in the Auto Programming Mode. In this mode, the 879XBH that is being programmed gets the data to be programmed from external memory over the system bus. Therefore, PCCR must correctly correspond to the memory system in the programming setup, which is not necessarily the memory organization of the application.

The following sections describe 879XBH programming in each programming mode.

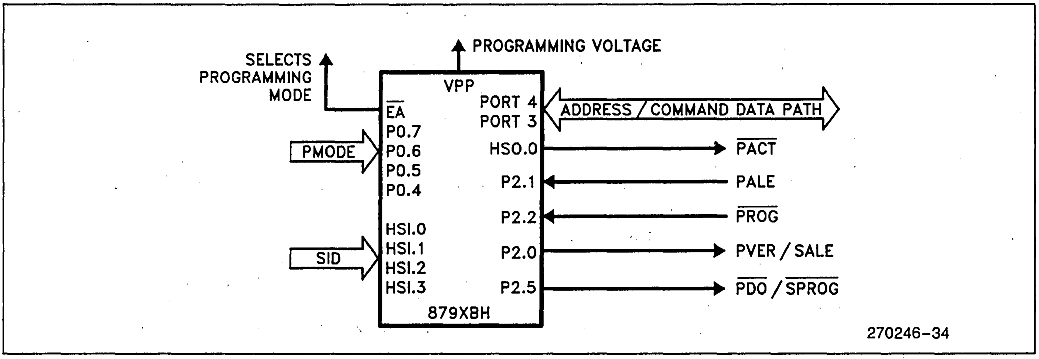


Figure 33. Programming Mode Pin Function

Name	Function
PMODE	**PROGRAMMING MODE SELECT:** Determines the EPROM programming algorithm that is performed. PMODE is sampled after a chip reset and should be static while the part is operating.
SID	**SLAVE ID NUMBER:** Used to assign each slave pin of Port 3 or 4 to use for passing programming verification acknowledgement. For example, if gang programming in the Slave Programming Mode, the slave with SID = 0001 will use Port 3.1 to signal correct or incorrect program verification.
PALE	**PROGRAMMING ALE INPUT**: Accepted by an 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain a command/address.
$\overline{\text{PROG}}$	**PROGRAMMING PULSE:** Accepted by 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain the data to be programmed. A falling edge on $\overline{\text{PROG}}$ signifies data valid and starts the programming cycle. A rising edge on $\overline{\text{PROG}}$ will halt programming in the slaves.
$\overline{\text{PACT}}$	**PROGRAMMING ACTIVE:** Used in the Auto-Programming Mode to indicate when programming activity is complete.
PVER	**PROGRAM VERIFIED:** A signal outut after a programming operation by parts in the Slave Programming Mode.
$\overline{\text{PDO}}$	**PROGRAMMING DURATION OVERFLOWED:** A signal output by parts in the Slave Programming Mode. Used to signify that the $\overline{\text{PROG}}$ pulse applied for a programming operation was longer than allowed.
SALE	**SLAVE ALE:** Output signal from an 879XBH in the Auto Programming Mode. A falling edge on SALE indicates that Ports 3 and 4 contain valid address/command information for slave 879XBHs that may be attached to the master.
$\overline{\text{SPROG}}$	**SLAVE PROGRAMMING PULSE:** Output from an 879XBH in the Auto Programming Mode. A falling edge on $\overline{\text{SPROG}}$ indictates that Ports 3 and 4 contain valid data for programming into slave 879XBHs that may be attached to the master.
PORTS 3 and 4	**ADDRESS/COMMAND/DATA BUS:** Used to pass commands, addresses and data to and from slave mode 879XBHs. Used by chips in the Auto Programming Mode to pass command, addresses and data to slaves. Also used in the Auto Programming Mode as a regular system bus to access external memory. Each line should be pulled up to VCC through a resistor.

Figure 34. Programming Mode Pin Definitions

10.2 Auto Programming Mode

The Auto Programming Mode provides the ability to program the internal 879XBH EPROM without having to use a special EPROM programmer. In this mode, the 879XBH simply programs itself with the data found at external locations 4000H through 5FFFH. All that is required is that some sort of external memory reside at these locations, that EA selects the programming mode and that VPP is applied. Figure 35 shows a minimum configuration for using an 8K x 8 EPROM to program one 879XBH in the Auto Programming Mode.

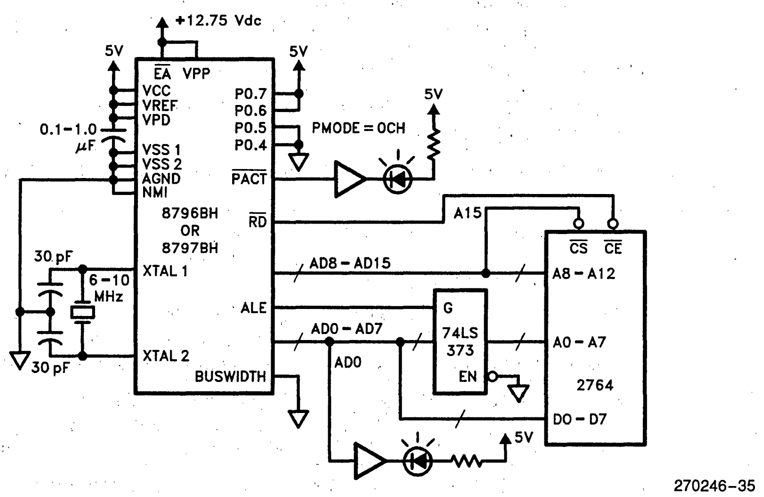
The 879XBH first reads a word from external memory, then the Modified Quick-Pulse Programming™ Algorithm (described later) is used to program the appropriate EPROM location. Since the erased state of a byte is 0FFH, the Auto Programming Mode will skip locations where the data to be programmed is 0FFH. When all 8K has been programmed, PACT goes high and the part outputs a 0 on Port 3.0 if it programmed correctly and a 1 if it failed.

10.2.1 GANG PROGRAMMING WITH THE AUTO PROGRAMMING MODE

An 879XBH in the Auto Programming Mode can also be used as a programmer for up to 15 other 879XBHs that are configured in the Slave Programming Mode. To accomplish this, the 879XBH acting as the master outputs the slave command/data pairs on Ports 3 and 4 necessary to program slave parts with the same data it is programming itself with. Slave ALE (SALE) and Slave PROG (SPROG) signals are provided by the master to the slaves to demultiplex the commands from the data. Figure 36 is a block diagram of a gang programming system using one 879XBH in the Auto Programming Mode. The Slave Programming Mode is described in the next section.

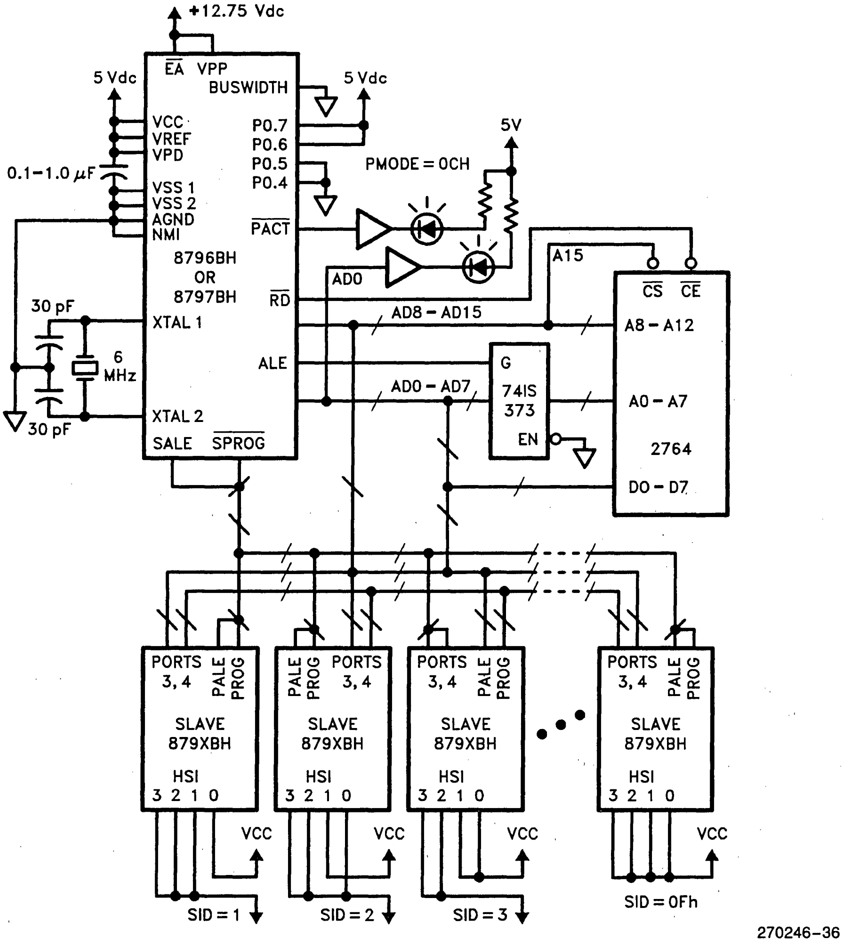
The master 879XBH first reads a word from the external memory controlled by ALE, RD and WR. It then drives Ports 3 and 4 with a Data Program command using the appropriate address and alerts the slaves with a falling edge on SALE. Next, the data to be programmed is driven onto Ports 3 and 4 and slave programming begins with a falling edge on SPROG. At the same time, the master begins to program its own EPROM location with the data read in. Intel's Modified Quick-Pulse Programming™ Algorithm is used, with Data Verify commands being given to the slaves after each programming pulse.

When programming is complete PACT goes high and Ports 3 and 4 are driven with all 1s if all parts programmed correctly. Individual bits of Port 3 and 4 will be driven to 0 if the slave with that bit number as an SID did not program correctly. The 879XBH used as the master assigns itself an SID of 0.



NOTE:
Ports 3 and 4 should have pullups to VCC.

Figure 35. The Auto Programming Mode



NOTE:
EA and VPP on slaves must be at +12.75 Vdc. Each slave's PMODE must equal 05H. Ports 3 and 4 should have pullups to VCC. Minimum configuration connections must also be made for slaves. A 10 MHz clock is recommended for the slaves.

Figure 36. Gang Programming with the Auto Programming Mode

10.3 Slave Programming Mode

Any number of 879XBHs can be programmed by a master programmer through the Slave Programming Mode.

The programming device uses Ports 3 and 4 of the parts being programmed as a command/data path. The slaves accept signals on PALE (Program ALE) and $\overline{\text{PROG}}$ (Program Enable) to demultiplex the commands and data. The slaves also use PVER, $\overline{\text{PDO}}$ and Ports 3 and 4 to pass error information to the programmer. Support for gang programming of up to 16 879XBHs is provided. If each part is given a unique SID (Slave ID Number) an 879XBH in the Auto Programming Mode can be used as a master to program itself and up to 15 other slave 879XBHs. There is, however, no 879XBH dependent limit to the number of parts that can be gang programmed in the slave mode.

It is important to note that the interface to an 879XBH in the slave mode is similar to a multiplexed bus. Attempting to issue consecutive PALE pulses without a corresponding $\overline{\text{PROG}}$ pulse will produce unexpected results. Similarly, issuing consecutive $\overline{\text{PROG}}$ pulses without the corresponding PALE pulses immediately preceding is equally unpredictable.

10.3.1 SLAVE PROGRAMMING COMMANDS

The commands sent to the slaves are 16-bits wide and contain two fields. Bits 14 and 15 specify the action that the slaves are to perform. Bits 0 through 13 specify the address upon which the action is to take place. Commands are sent via Ports 3 and 4 and are available to cause the slaves to program a word, verify a word, or dump a word (Table 1). The address part of the command sent to the slaves ranges from 2000H to 3FFFH and refers to the internal EPROM memory space. The following sections describe each slave programming mode command.

Table 1. Slave Programming Mode Commands

P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

DATA PROGRAM COMMAND—After a Data Program Command has been sent to the slaves, $\overline{\text{PROG}}$ must be pulled low to cause the data on Ports 3 and 4 to be programmed into the location specified during the command. The falling edge of $\overline{\text{PROG}}$ is not only used to indicate data valid, but also triggers the hardware programming of the word specified. The slaves will begin programming 48 states after $\overline{\text{PROG}}$ falls, and will continue to program the location until $\overline{\text{PROG}}$ rises.

After the rising edge of $\overline{\text{PROG}}$, the slaves automatically perform a verification of the address just programmed. The result of this verification is then output on PVER (Program Verify) and $\overline{\text{PDO}}$ (Program Duration Overflowed). Therefore, verification information is available following the Data Program Command for programming systems that cannot use the Data Verify command.

If PVER and $\overline{\text{PDO}}$ of all slaves are 1s after $\overline{\text{PROG}}$ rises then the data program was successful everywhere. If PVER is a 0 in any slave, then the data programmed did not verify correctly in that part. If $\overline{\text{PDO}}$ is a 0 in any slave, then the programming pulse in those parts was terminated by an internal safety feature rather than the rising edge of $\overline{\text{PROG}}$. The safety feature prevents over-programming in the slave mode. Figure 37 shows the relationship of PALE, $\overline{\text{PROG}}$, PVER and $\overline{\text{PDO}}$ to the Command/Data Path on Ports 3 and 4 for the Data Program Command.

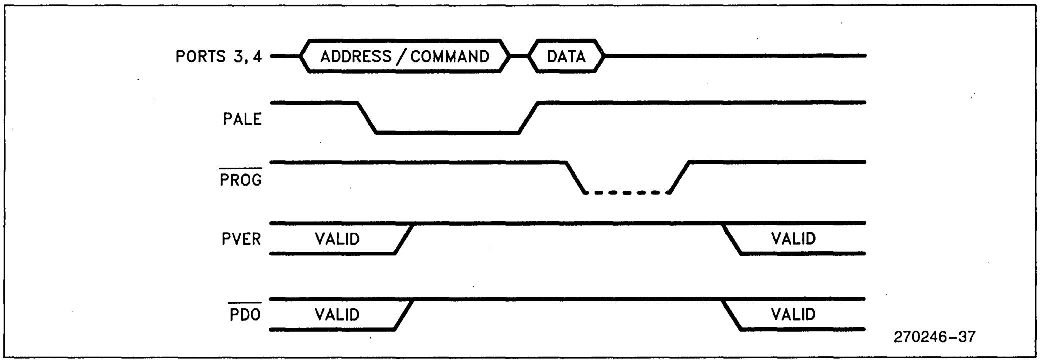


Figure 37. Data Program Signals in Slave Programming Mode

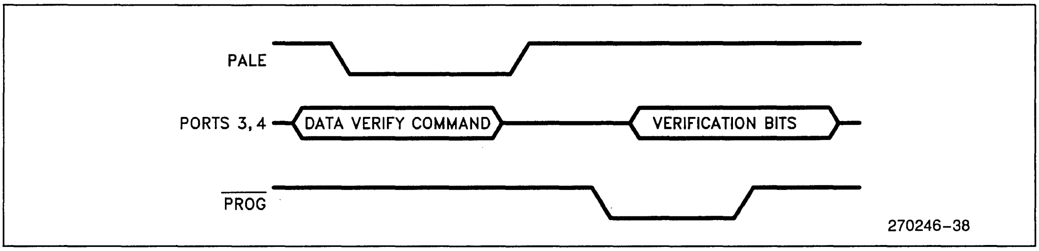


Figure 38. Data Verify Command Signals

DATA VERIFY COMMAND—When the Data Verify Command is sent, the slaves respond by driving one bit of Port 3 and 4 to indicate correct or incorrect verification of the previous Data Program. A 1 indicates correct verification, while a 0 indicates incorrect verification. The SID (Slave ID Number) of each slave determines which bit of the command/data path is driven. $\overline{\text{PROG}}$ from the programmer governs when the slaves drive the bus. Figure 38 shows the relationship of Ports 3 and 4 to PALE and $\overline{\text{PROG}}$.

This command is always preceded by a Data Program Command in a programming system with as many as 16 slaves. However, a Data Verify Command does not have to follow every Data Program Command.

WORD DUMP COMMAND—When the Word Dump Command is issued, the 879XBH being programmed adds 2000H to the address field of the command and places the value found at the new address on Ports 3 and 4. For example, sending the command #0100H to a slave will result in the slave placing the word found at location 2100H on Ports 3 and 4. $\overline{\text{PROG}}$ from the programmer governs when the slave drives the bus. The signals are the same as shown in Figure 22.

Note that this command will work only when just one slave is attached to the bus, and that there is no restriction on commands that precede or follow a Word Dump Command.

10.3.2 GANG PROGRAMMING WITH THE SLAVE PROGRAMMING MODE

Gang programming of 879XBHs can be done using the Slave Programming Mode. There is no 879XBH based limit on the number of chips that may be hooked to the same Port 3/Port 4 data path for gang programming.

If more than 16 chips are being gang programmed, the PVER and $\overline{\text{PDO}}$ outputs of each chip could be used for verification. The master programmer could issue a data program command then either watch every chip's error signals, or AND all the signals together to get a system PVER and $\overline{\text{PDO}}$.

If 16 or fewer 879XBHs are to be gang programmed at once, a more flexible form of verification is available. By giving each chip being programmed a unique SID, the master programmer could then issue a data verify command after the data program command. When a verify command is seen by the slaves, each will drive one pin of Port 3 or 4 with a 1 if the programming verified correctly or a 0 if programming failed. The SID is used by each slave to determine which Port 3, 4 bit it is assigned. An 879XBH in the Auto Programming Mode could be the master programmer if 15 or fewer slaves need to be programmed (see Gang Programming with the Auto Programming Mode).

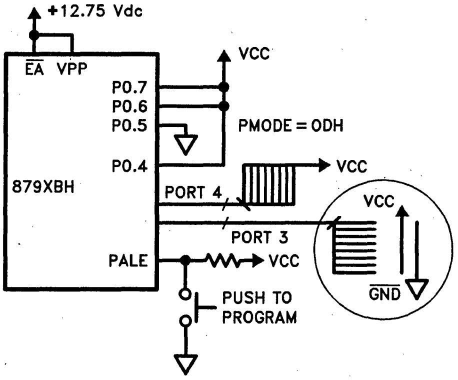
10.4 Auto Configuration Byte Programming Mode

The CCB (location 2018H) can be treated just like any other EPROM location, and programmed using any programming mode. But to provide for simple programming of the CCB when no other locations need to be programmed, the Auto Configuration Byte Programming Mode is provided. Programming in this mode also programs PCCB. Figure 39 shows a block diagram for using the Auto Configuration Byte Programming Mode.

With PMODE = 0DH and 0FFH on Port 4, CCB and PCCB will be programmed to the value on Port 3 when a logic 0 is placed on PALE. After programming is complete, PVER will be driven to a 1 if the bytes programmed correctly, and a 0 if the programming failed.

This method of programming is the only way to program PCCB. PCCB is a non-memory mapped EPROM location that gets loaded into CCR during the reset sequence when the voltage on $\overline{EA}$ puts the 879XBH in Programming Mode. If PCCB is not programmed using the Auto Configuration Byte Programming Mode, every time the 879XBH is put into Programming Mode the CCR will be loaded with 0FFH (the value of the erased PCCB location).

However, if programming the CCB and PCCB is done using this Programming Mode, the PCCB will take on the value programmed into CCB. This means that until the part is erased, programming activities that use the system bus will employ the bus width and controls selected by the user's CCB.



270246–39

NOTES:
1. Tie Port 3 to the value desired to be programmed into CCB, and PCCB.
2. Make all necessary minimum connections for power, ground and clock.

Figure 39. The Auto CCR Programming Mode

10.5 Run-Time Programming

Run-Time Programming of the 879XBH is provided to allow the user complete flexibility in the ways in which the internal EPROM is programmed. That flexibility includes the ability to program just one byte or one word instead of the whole EPROM, and extends to the hardware necessary to program. The only additional requirement of a system is that a programming voltage is applied to VPP. Run-Time Programming is done with $\overline{EA}$ at TTL-high (normal operation—internal/external access).

To Run-Time program, the user writes a byte or word to the location to be programmed. Once this is done, the 879XBH will continue to program that location until another data read from or data write to the EPROM occurs. The user can therefore control the duration of the programming pulse to within a few mircoseconds. An intelligent algorithm should be implemented in software. It is recommended that the Modified Quick-Pulse Programming Algorithm be implemented.

After the programming of a location has started, care must be taken to ensure that no program fetches (or pre-fetches) occur from internal memory. This is of no concern if the program is executing from external memory. However, if the program is executing from internal memory when the write occurs, it will be necessary to use the built in "Jump to Self" located at 201AH.

"Jump to Self" is a two byte instruction in the Intel test ROM which can be CALLed after the user has started programming a location by writing to it. A software timer interrupt could then be used to escape from the "Jump to Self" when the proper programming pulse duration has elapsed. Figure 40 is an example of how to program an EPROM location while execution is entirely internal.

Upon entering the PROGRAM routine, the address and data are retrieved from the STACK and a Software Timer is set to expire one programming pulse later. The data is then written to the EPROM location and a CALL to location 201AH is made. Location 201AH is in Intel reserved test ROM, and contains the two byte opcode for a "Jump to Self". The minimum interrupt service routine would remove the 201AH return address from the STACK and return.

```
PROGRAM:

    POP  temp                                   ;take parameters from the
                                                 STACK
    POP  address_temp
    POP  data_temp
    PUSH temp

    PUSHF                                       ;save current status
    LDB  int_mask , #enable_swt_only            ;enable only swt interrups
    LDB  HSO_COMMAND , #SWT0_ovf                ;load swt command to interrupt
    ADD  HSO_TIME,TIMER1, #program_pulse        ;when program pulse time
                                                ;has elapsed
    EI
    ST   data-temp, [address_temp]
    CALL 201AH
    POPF
    RET

SWT_ISR:
    • • •

swt0_expired:
    POP  0
    RET
    • • •
```

Figure 40. Programming the EPROM from Internal Memory Execution

10.6 ROM/EPROM Program Lock

Protection mechanisms have been provided on the ROM and EPROM versions of the 8096BH to inhibit unauthorized accesses of internal program memory. However, there must always be a way to allow authorized program memory dumps for testing purposes. The following describes 839XBH, 879XBH program lock features and the mode provided for authorized memory dumps.

10.6.1 LOCK FEATURES

Write protection is provided for EPROM parts, while READ protection is provided for both ROM and EPROM parts.

Write protection is enabled by causing the LOC0 bit in the CCR to take the value 0. When WRITE protection is selected, the bus controller will cycle through the write sequence, but will not actually drive data to the EPROM and will not enable VPP to the EPROM. This protects the entire EPROM 2000H–3FFFH from inadvertant or unauthorized programming, and also prevents writes to the EPROM from upsetting program execution. If write protection is not enabled, a data write to an internal EPROM location will begin programming that location, and continue programming the location until a data read of the internal EPROM is executed. While programming, instruction fetches from internal EPROM will not be successful.

READ protection is selected by causing the LOC1 bit in the CCR to take the value 0. When READ protection is enabled, the bus controller will only perform a data read from the address range 2020H–3FFFH if the slave program counter is in the range 2000H–3FFFH. Note that since the slave PC can be many bytes ahead of the CPU program counter, an instruction that is located after address 3FFAH may not be allowed to access protected memory, even though the instruction is itself protected.

If the bus controller receives a request to perform a READ of protected memory, the READ sequence occurs with indeterminant data being returned to the CPU.

Other enhancements were also made to the 8096BH for program protection. For example, the value of $\overline{EA}$ is latched on reset so that the device cannot be switched from external to internal execution mode at run-time. In addition, if READ protection is selected, an NMI event will cause the device to switch to external only execution mode. Internal execution can only resume by resetting the chip.

10.6.2 AUTHORIZED ACCESS OF PROTECTED MEMORY

To provide a method of dumping the internal ROM/EPROM for testing purposes a "Security Key" mechanism and ROM dump mode have been implemented.

The security key is a 128 bit number, located in internal memory, that must be matched before a ROM dump will occur. The application code contains the security key starting at location 2020H.

The ROM dump mode is entered just like any programming mode ($\overline{EA}$ = 12.75V), except that a special PMODE strapping is used. The PMODE for ROM dump is 6H (0110B).

The ROM dump sequence begins with a security key verification. Users must place at external locations 4020H–402FH the same 16 byte key that resides inside the chip at locations 2020H–202FH. Before doing a ROM dump, the chip checks that the keys match.

After a successful key verification, the chip dumps data to external locations 1000H–11FFH and 4000H–5FFFH. Unspecified data appears at the low addresses.

Internal EPROM/ROM is dumped to 4000H–5FFFH, beginning with internal address 2000H.

If a security key verification is not successful, the chip will put itself into an endless loop of internal execution.

NOTE:

Substantial effort has been expended to provide an excellent program protection scheme. However, Intel cannot, and does not guarantee that the protection methods that we have devised will prevent unauthorized access.

10.7 Modified Quick-Pulse Programming™ Algorithm

The Modified Quick-Pulse Programming Algorithm calls for each EPROM location to receive 25 separate 100 μs (±5 μs) program cycles. Verification of correct programming is done after the 25 pulses. If the location verifies correctly, the next location is programmed. If the location fails to verify, the location has failed.

Once all locations are programmed and verified, the entire EPROM is again verified.

Programming of 879XBH parts is done with VPP = 12.75V ±0.25V and VCC = 5.0V ±0.5V.

10.8 Signature Word

The 8X9XBH contains a signature word at location 2070H. The word can be accessed in the slave mode by executing a word dump command.

Table 2. 8X9XBH Signature Words

Device	Signature Word
879XBH	896FH
839XBH	896EH
809XBH	Undefined

10.9 Erasing the 879XBH EPROM

Initially, and after each erasure, all bits of the 879XBH are in the "1" state. Data is introduced by selectively programming "0s" into the desired bit locations. Although only "0s" will be programmed, both "1s" and "0s" can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The erasure characteristics of the 879XBH are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Constant exposure to room level fluorescent lighting could erase the typical 879XBH in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 879XBH is to be exposed to light for extended periods of time, opaque labels must be placed over the EPROM's window to prevent unintentional erasure.

The recommended erasure procedure for the 879XBH is exposure to shortwave ultraviolet light which has a wavelength of 2537Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15 Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 μW/cm^2 power rating. The 879XBH should be placed within 1 inch of the lamp tubes during erasure. The maximum integrated dose an 879XBH can be exposed to without damage is 7258 Wsec/cm^2 (1 week @ 12000 μW/cm^2). Exposure of the 879XBH to high intensity UV light for long periods may cause permanent damage.

80C196KA Architectural Overview

20

80C196KA ADVANCED CHMOS MICROCONTROLLER ARCHITECTURAL OVERVIEW

1.0 INTRODUCTION

All of the features available on the 8096BH are present on the 80C196KA including:

Register to Register Architecture

232 Bytes of Register File

22 Interrupt Sources With 8 Vector Locations

High Speed 16x16 Multiply

High Speed 32/16 Divide

Five 8-bit I/O Ports

Analog to Digital Converter (A/D Versions Only)

Pulse-Width-Modulated Output

Full Duplex Serial Port With Dedicated Baud Rate Generator

16-bit Watchdog Timer

High Speed Subsystem With
- Up to 4 Time Capture Inputs
- Up to 6 Time Triggered Outputs
- 2 16-bit Timer/Counters
- 4 Software Timers

In addition, the 80C196KA has:

Independent Capture of Timer2

Up and Down Counting on Timer2

2.33 μs 16x16 Multiply vs 6.25 μs on 8096BH

4.0 μs 32/16 Divide vs 6.25 μs on 8096BH

6 Additional Interrupt Sources / 10 Additional Vectors

6 Additional Instructions

Power Down and Idle Modes for Power Savings

and many other feature enhancements. The 80C196KA can be plugged into most 8096BH designs with only a few minor software changes.

This document can be used as a stand-alone guide to the features of the 80C196KA and as a programmer's guide and user's manual by experienced 8096 programmers. For those people who are not familiar with the details of programming an 8096, this manual should be used in conjunction with the current edition of the Embedded Controller Handbook.

2.0 ARCHITECTURAL OVERVIEW

For the purpose of describing its operation, the 80C196KA can be divided into three sections: the processing unit, peripheral (I/O) devices, and support circuitry. The processing unit consists of the 16-bit CPU with its register file, the interrupt controller and the memory controller. Peripheral devices, a clock generator, and some miscellaneous support circuitry make up the remainder of the chip. A block diagram of the 80C196KA is shown in Figure 1.

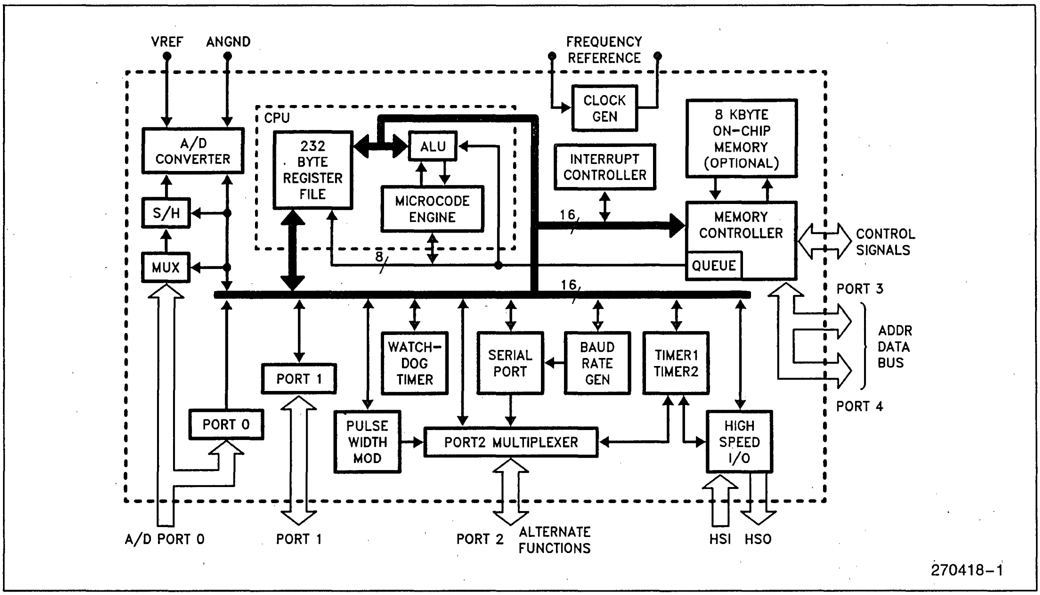


Figure 1. 80C196KA Block Diagram

2.1 INTERNAL TIMINGS

Internal operation of the chip is based on the oscillator frequency divided by two, giving the basic operating time unit, known as a "State Time". With a 12 MHz oscillator, a state time is 167 nanoseconds. With an 8 MHz oscillator, a state time is 250 nanoseconds, the same as that of an 8096 running with a 12 MHz oscillator. Since the 80C196KA will be run at many frequencies, the times given throughout this overview will be in state times or "states", unless otherwise specified.

Either a crystal or an external source can be used to drive the on-chip oscillator. Figure 2 shows a circuit for the oscillator connected to a crystal. When an external source is used, it is connected to the XTAL1 pin leaving the XTAL2 pin floating. The XTAL2 pin becomes a weak output in this mode and must be left unconnected.

Two non-overlapping internal phases are created by the clock generator: phase 1 and phase 2. Phase 2 is buffered and output on the CLKOUT pin. This is not the same as on the 8096, since it uses a three-phase clock. Changing from a three-phase clock to a two-phase one speeds up the operation of the chip for a set oscillator frequency. It should cause no compatibility problems in most designs, but does cause some differences in the system bus timings. A detailed description of the bus timing is included in the electrical characteristics section of this document.

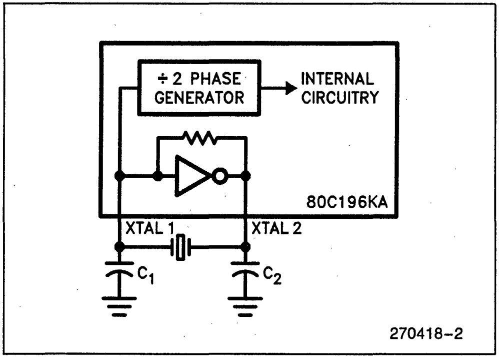


Figure 2. Oscillator

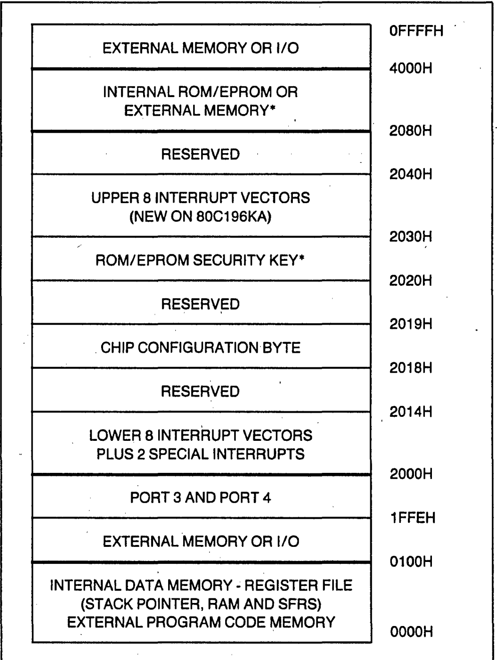
2.2 CPU

The CPU on the 80C196KA is 16 bits wide and is connected to the interrupt controller and the memory controller by a 16-bit bus. In addition, there is an 8-bit bus which is used to transfer opcodes from the memory controller to the CPU. On the 8096 there is no 16-bit bus between the CPU and memory controller, so the 8-bit bus is used for both data and opcode transfers. All of the peripheral devices on the 80C196KA are connected to the CPU by a 16-bit bus.

A microcode engine controls the CPU, allowing it to perform operations with any byte, word or double word in the 232-byte Register File. Operations can also be performed with any of the I/O Control Registers, also called Special Function Registers (SFRs). With a flat architecture, the programmer is not limited to a single accumulator since all 256 bytes in the register file and SFR space can be used as accumulators. This eliminates accumulator bottleneck and allows the use of 3 operand instructions. The internal hardware of the CPU is similar to that of the 8096, except that extra hardware has been added to provide a faster multiply.

2.3 MEMORY MAP

64 Kbytes of addressable memory space are available on the 80C196KA, most of which can be used for program or data storage. The space from 100H through 0FFFFH contains a small block of reserved or special function locations but is otherwise available to the user. The reserved locations must contain 0FFH. Resetting the chip sets the program counter to location 2080H, allowing 8 Kbytes of RAM contiguous with the internal RAM at location 0FFH. The interrupt vectors, configuration byte, and several reserved addresses are located between 2000H and 207FH. Figure 3 shows a memory map of the 80C196KA memory space.


0FFFFH
EXTERNAL MEMORY OR I/O
4000H
INTERNAL ROM/EPROM OR EXTERNAL MEMORY*
2080H
RESERVED
2040H
UPPER 8 INTERRUPT VECTORS (NEW ON 80C196KA)
2030H
ROM/EPROM SECURITY KEY*
2020H
RESERVED
2019H
CHIP CONFIGURATION BYTE
2018H
RESERVED
2014H
LOWER 8 INTERRUPT VECTORS PLUS 2 SPECIAL INTERRUPTS
2000H
PORT 3 AND PORT 4
1FFEH
EXTERNAL MEMORY OR I/O
0100H
INTERNAL DATA MEMORY - REGISTER FILE (STACK POINTER, RAM AND SFRS) EXTERNAL PROGRAM CODE MEMORY
0000H

*ROM/EPROM will be available on future versions of 80C196.

Figure 3. 80C196KA Memory Map

Between 0H and 0FFH program execution fetches will always be from external memory, even if the chip has an onboard ROM or EPROM. This area of external memory is reserved for use by Intel development systems and should not be used in applications which will require development tools. Data fetches will always come from the on-chip register file and SFRs. The internal RAM from location 01AH (26 decimal) to 0FFH is the register file. This memory region, as well as the status of the majority of the chip, is kept alive while the chip is in the powerdown mode. (On the 8096 only the top 16 bytes of RAM were kept alive.) Details on powerdown mode are discussed in a later section.

Locations 18H and 19H are considered part of the register file although they are used as the stack pointer. The stack can be located anywhere in memory, internal or external, by using the 16-bit pointer. If the stack is not being used, these two bytes can be used as regular RAM.

Locations 00H through 17H are the I/O control registers or SFRs. As shown in Figure 4, two SFR windows are provided on the 80C196KA. Selecting the active window is done by using the Window Select Register (WSR) at location 14H in all of the windows.

Only two values may be written to the WSR, 0 and 15. Other values are reserved for use in future parts and will cause unpredictable operation.

Window 0, the register window selected with WSR = 0, is a superset of the one used on the 8096. As depicted in Figure 5, it has 24 registers, some of which have different functions when read than when written. Registers which are new to the 80C196KA or have changed functions from the 8096 are indicated in the figure. Figure 6 contains brief descriptions of the registers. Detailed descriptions are contained in the section which discusses the peripheral device controlled by the register.

In register Window 15 (WSR = 15), the operation of the SFRs is changed, so that those which were read-only in the 8096 SFR space are write-only and vice versa. The only exception to this is that TIMER2 is read/write in Window 0, and T2 Capture is read/write in Window 15. (TIMER2 was read-only on the 8096.) Registers which can be read and written in Window 0 can also be read and written in Window 15. Details of using Window 15 are discussed in the peripheral description section.

Caution must be taken when using the SFRs as sources of operations or as base or index registers for indirect or indexed operations. It is possible to not get the desired results, since external events can change SFRs and some SFRs clear when read. The potential for an SFR to change value must be taken into account when operating on these registers. This is particularly important when high level languages are used as they do not always make allowances for SFR-type registers.

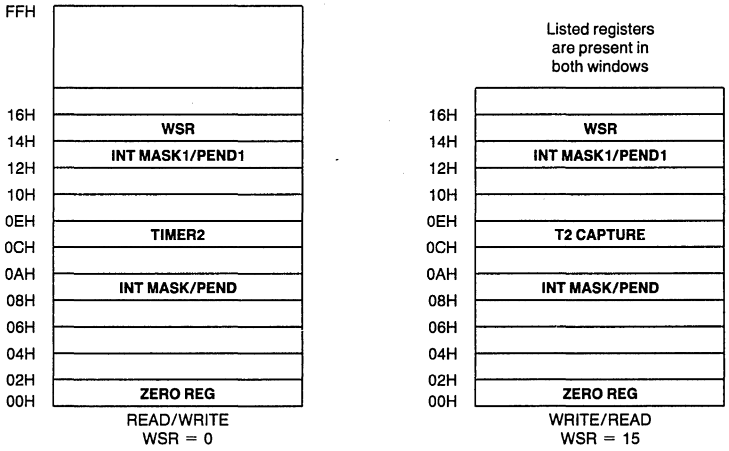


Figure 4. Multiple Register Windows

	WHEN READ		WHEN WRITTEN
19H 18H	STACK POINTER	19H 18H	STACK POINTER
17H	*IOS2	17H	PWM_CONTROL
16H	IOS1	16H	IOC1
15H	IOS0	15H	IOC0
14H	*WSR	14H	*WSR
13H	*INT_MASK 1	13H	*INT_MASK 1
12H	*INT_PEND 1	12H	*INT_PEND 1
11H	*SP_STAT	11H	*SP_CON
10H	PORT2	10H	PORT2
0FH	PORT1	0FH	PORT1
0EH	PORT0	0EH	BAUD RATE
0DH	TIMER2 (HI)	0DH	TIMER2 (HI)
0CH	TIMER2 (LO)	0CH	TIMER2 (LO)
0BH	TIMER1 (HI)	0BH	*IOC2
0AH	TIMER1 (LO)	0AH	WATCHDOG
09H	INT_PENDING	09H	INT_PENDING
08H	INT_MASK	08H	INT_MASK
07H	SBUF(RX)	07H	SBUF(TX)
06H	HSI_STATUS	06H	HSO_COMMAND
05H	HSI_TIME (HI)	05H	HSO_TIME (HI)
04H	HSI_TIME (LO)	04H	HSO_TIME (LO)
03H	AD_RESULT (HI)	03H	HSI_MODE
02H	AD_RESULT (LO)	02H	AD_COMMAND
01H	ZERO REG (HI)	01H	ZERO REG (HI)
00H	ZERO REG (LO)	00H	ZERO REG (LO)

WSR = 0

0DH	*T2 CAPTURE (HI)
0CH	*T2 CAPTURE (LO)

WSR = 15

OTHER SFRS IN WSR
15 BECOME READABLE
IF THEY WERE WRITABLE
IN WSR = 0 AND WRITABLE
IF THEY WERE READABLE
IN WSR = 0

*NEW OR CHANGED
REGISTER FUNCTION

Figure 5. Special Function Registers

Register	Description
R0	Zero Register - Always reads as a zero, useful for a base when indexing and as a constant for calculations and compares.
AD_RESULT	A/D Result Hi/Low - Low and high order results of the A/D converter
AD_COMMAND	A/D Command Register - Controls the A/D
HSI_MODE	HSI Mode Register - Sets the mode of the High Speed Input unit.
HSI_TIME	HSI Time Hi/Lo - Contains the time at which the High Speed Input unit was triggered.
HSO_TIME	HSO Time Hi/Lo - Sets the time or count for the High Speed Output to execute the command in the Command Register.
HSO_COMMAND	HSO Command Register - Determines what will happen at the time loaded into the HSO Time registers.
HSI_STATUS	HSI Status Registers - Indicates which HSI pins were detected at the time in the HSI Time registers and the current state of the pins.
SBUF(TX)	Transmit buffer for the serial port, holds contents to be outputted.
SBUF(RX)	Receive buffer for the serial port, holds the byte just received by the serial port.
INT_MASK	Interrupt Mask Register - Enables or disables the individual interrupts. (also IMASK)
INT_PENDING	Interrupt Pending Register - Indicates that an interrupt signal has occurred on one of the sources and has not been serviced. (also IPEND)
WATCHDOG	Watchdog Timer Register - Written to periodically to hold off automatic reset every 64K state times.
TIMER1	Timer 1 Hi/Lo - Timer1 high and low bytes.
TIMER2	Timer 2 Hi/Lo - Timer2 high and low bytes.
IOPORT0	Port 0 Register - Levels on pins of Port 0.
BAUD_RATE	Register which determines the baud rate, this register is loaded sequentially.
IOPORT1	Port 1 Register - Used to read or write to Port 1.
IOPORT2	Port 2 Register - Used to read or write to Port 2.
SP_STAT	Serial Port Status - Indicates the status of the serial port.
SP_CON	Serial Port Control - Used to set the mode of the serial port.
IOS0	I/O Status Register 0 - Contains information on the HSO status.
IOS1	I/O Status Register 1 - Contains information on the status of the timers and of the HSI.
IOC0	I/O Control Register 0 - Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.
IOC1	I/O Control Register 1 - Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.
PWM_CONTROL	Pulse Width Modulation Control Register - Sets the duration of the PWM pulse.
IPEND1	Interrupt Pending register for the 8 new interrupt vectors (also INT_PENDING1)
IMASK1	Interrupt Mask register for the 8 new interrupt vectors (also INT_MASK1)
IOC2	I/O Control Register 2 - Controls new 80C196KA features
IOS2	I/O Status Register 2 - Contains information on HSO events
WSR	Window Select Register - Selects register window

Figure 6. Special Function Register Description

2.4 MEMORY CONTROLLER

All of the program memory and the external data memory are transferred to the CPU through the memory controller. Within the memory controller is a slave program counter, an instruction queue, and a bus controller.

The slave program counter keeps track of the program counter in the CPU and requests the correct sequence of instructions to be fetched by the bus controller and stored in the queue.

Instruction Queue

A four byte instruction queue allows the CPU to run faster by keeping the next instruction byte almost always available. When the instruction flow changes, as with a branch or call instruction, the queue is flushed and refilled. The amount of time required to do this is included in the instruction execution times which are listed in other sections of this document.

When debugging code using a logic analyzer, one must be aware of the queue. It is not possible to determine when an instruction will begin executing by simply watching when it is read since the queue is filled in advance of instruction execution. In addition, the algorithms which are used to keep the queue full may cause instructions to be read into the 80C196KA multiple times.

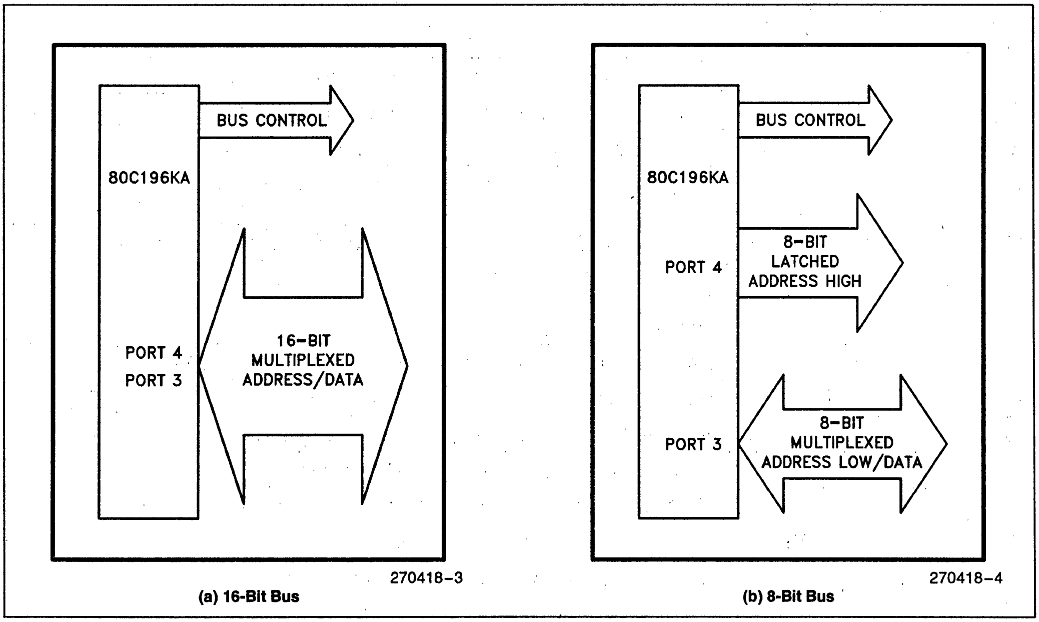
Bus Controller

Both 8-bit and 16-bit bus modes are supported by the bus controller. A block diagram of the two modes is shown in Figure 7. Each mode has several variations, all of which are controlled by the Chip Configuration Register (CCR), shown in Figure 8. This register is at an unmapped location within the 80C196KA and is loaded from location 2018H during the chip reset sequence.

Switching between 8 and 16-bit bus modes can be done using the buswidth pin if the CCR is set for a 16-bit bus. Dynamically switching between the two modes is possible by changing this pin on the fly. A system using 16-bit wide program memory for speed, but only needing one 8-bit RAM chip, could make use of this feature to avoid the use of another RAM or the software needed to convert word wide data into data stored in every other byte.

When CCR bits 2 and 3 are both set to 1 the standard 8096BH bus control signals are provided, as shown in Figure 9. $\overline{\text{WR}}$ will come out for each write. $\overline{\text{BHE}}$ will be valid throughout the bus cycle and can be combined with the $\overline{\text{WR}}$ and address line 0 to form $\overline{\text{WRL}}$ (Write Low byte) and $\overline{\text{WRH}}$ (Write High byte). ALE will rise as the address starts to come out and will fall to provide a signal to externally latch the address.

The Write Strobe mode eliminates the need to externally decode $\overline{\text{WRL}}$ and $\overline{\text{WRH}}$ (See Figure 10). In 16-bit bus modes, $\overline{\text{WRL}}$ and $\overline{\text{WRH}}$ are provided on the $\overline{\text{WR}}$


BUS CONTROL
80C196KA
PORT 4
PORT 3
16-BIT MULTIPLEXED ADDRESS/DATA
270418-3
BUS CONTROL
80C196KA
PORT 4
8-BIT LATCHED ADDRESS HIGH
PORT 3
8-BIT MULTIPLEXED ADDRESS LOW/DATA
270418-4

(a) 16-Bit Bus

(b) 8-Bit Bus

Figure 7. Bus Width Options

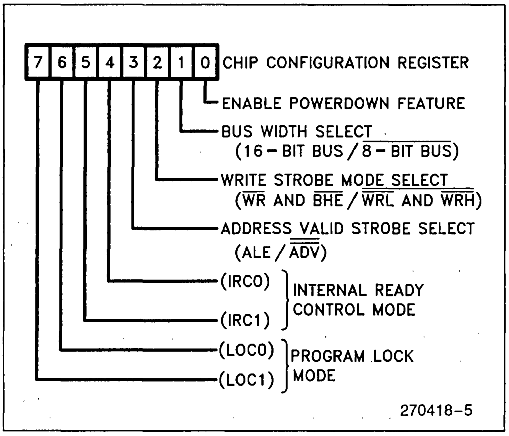
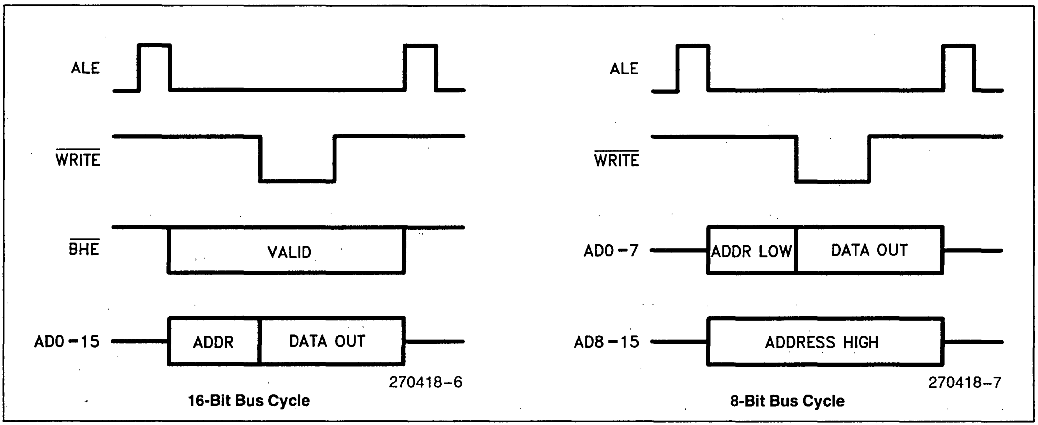

7 6 5 4 3 2 1 0
CHIP CONFIGURATION REGISTER
ENABLE POWERDOWN FEATURE
BUS WIDTH SELECT (16-BIT BUS / 8-BIT BUS)
WRITE STROBE MODE SELECT (WR AND BHE / WRL AND WRH)
ADDRESS VALID STROBE SELECT (ALE / ADV)
(IRC0) (IRC1) INTERNAL READY CONTROL MODE
(LOC0) (LOC1) PROGRAM LOCK MODE
270418-5

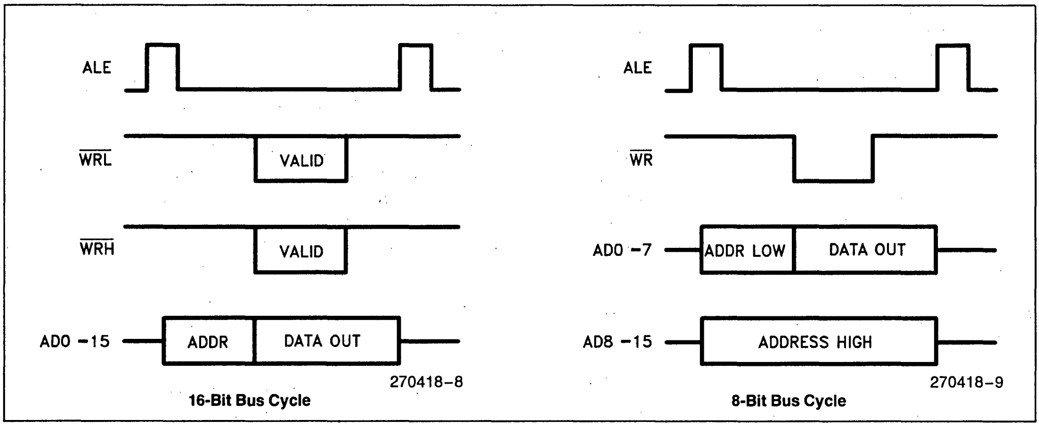
Figure 8. Format of the Chip Configuration Register



16-Bit Bus Cycle

8-Bit Bus Cycle

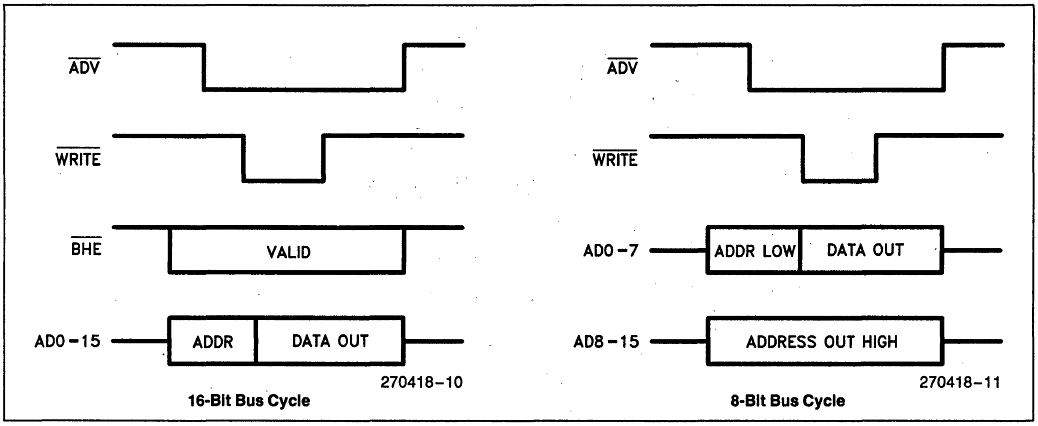
Figure 9. Standard Bus Control



16-Bit Bus Cycle

8-Bit Bus Cycle

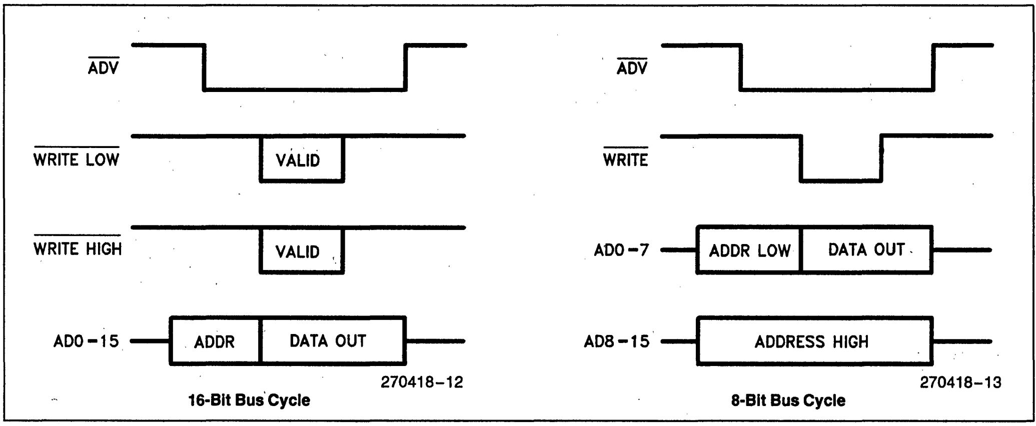
Figure 10. Write Strobe Mode



16-Bit Bus Cycle

8-Bit Bus Cycle

Figure 11. Address Valid Mode



16-Bit Bus Cycle

8-Bit Bus Cycle

Figure 12. Address Valid With Write Strobe Mode

and $\overline{\text{BHE}}$ lines, respectively. Both lines go low for word writes, while only one line will go low for a byte write. The $\overline{\text{WR}}$ line is provided for 8-bit bus modes and will go low for all writes. Clearing CCR bit 2 selects this mode.

An Address Valid ($\overline{\text{ADV}}$) strobe can be provided in place of ALE if CCR bit 3 is cleared (See Figure 11). In this mode, $\overline{\text{ADV}}$ will go low after an external address is set up and stay low until the end of the bus cycle, at which time it will go high. This signal can be used to provide a chip select for external memory.

Both the Address Valid Strobe mode and the Write Strobe Mode can be enabled at the same time providing the signals shown in Figure 12.

The $\overline{\text{EA}}$ pin is used to determine whether program code in the address range 2000H through 3FFFH is fetched from internal memory or external memory. Since the 80C196KA does not have internal memory this pin must be externally tied low. Future ROM and EPROM parts, the 83C196KB and 87C196KB respectively, will execute from internal memory if the pin is tied high. The $\overline{\text{EA}}$ pin is latched on chip reset and cannot be changed without resetting the chip.

A READY pin limit can be set with the CCR, determining the maximum number of wait states that will be allowed when the READY pin is pulled low. This eliminates the need for external hardware to remove the READY signal prior to the next bus cycle. The IRC0 and IRC1 bits control wait states as follows:

IRC1	IRC0	Description
0	0	Limit to one wait state
0	1	Limit to two wait states
1	0	Limit to three wait states
1	1	Wait states not limited internally

When internal program memory is used, the CCR can set read and write protection using the LOC0 and LOC1 bits (CCR bits 6 and 7). A zero on LOC0 enables read protections and a zero on LOC1 enables write protection. Both read and write protection may be enabled at the same time by clearing both bits.

2.5 INTERRUPTS

Twenty-eight (28) sources of interrupts are available on the 80C196KA. These sources are gathered into 15 vectors plus special vectors for NMI, the TRAP instruction, and Unimplemented Opcodes. Figure 13 shows the routing of the interrupt sources into their vectors as well as the control bits which enable some of the sources.

NMI, the external Non-Maskable Interrupt, is the highest priority peripheral interrupt. It vectors indirectly through location 203EH. For design symmetry, a mask bit exists in INT__MASK1 for the NMI. To prevent accidental masking of an NMI, the bit does not function and will not stop an NMI from occurring. NMI on the 8096 vectored directly to location 0000H, so for the 80C196KA to be compatible with 8096 software, which uses the NMI, location 203EH must be loaded with 0000H.

Opcode F7H, the TRAP instruction, causes an indirect vector through location 2010H. All unimplemented opcodes are mapped into a special interrupt vector through location 2012H. They act as uninterruptable instructions and take one more state time than the TRAP instruction.

The interrupt sources in the 80C196KA are arranged in a fixed priority. Figure 14 shows the priorities (15 is highest) of the interrupts and their vector locations. If simultaneous interrupt requests are received, the highest priority source that is both pending and enabled will get serviced. Software priorities can be provided by enabling and disabling different interrupts in different routines. When an interrupt occurs, the 80C196KA's response is identical to that of the 8096; it decrements the stack pointer value by 2 and then stacks the program counter value. Because of the additional 16-bit internal bus, the 80C196KA interrupt response takes only 16/18 states compared with 21/24 states on the 8096 (states: stack internal/external).

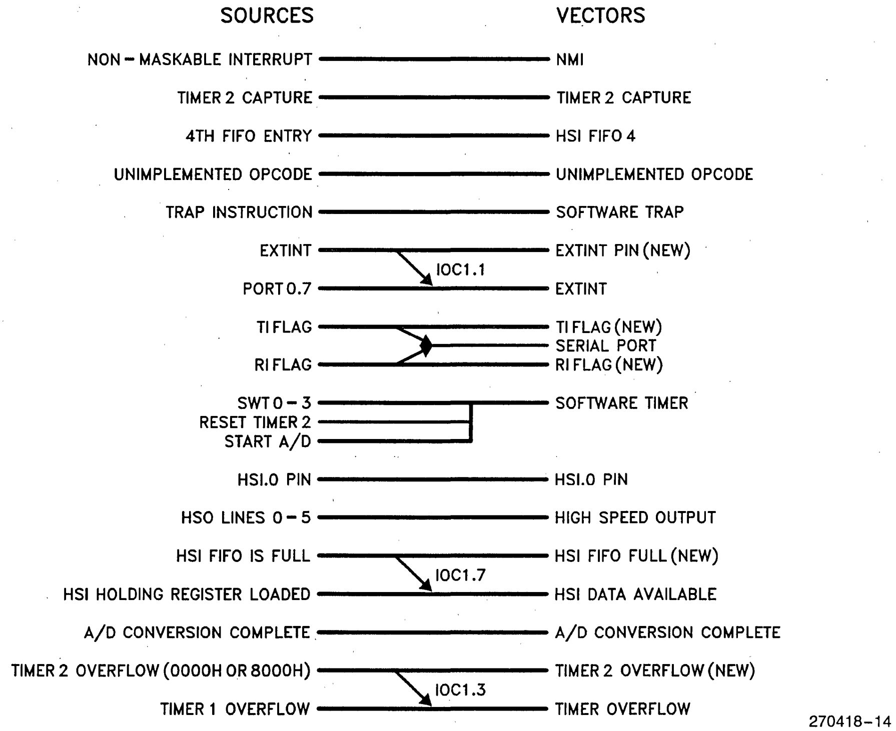


Figure 13. All Possible Interrupt Sources

80C196KA INTERRUPTS

Number	Source	Vector Location	Priority
INT15	NMI	203EH	15
INT14	HSI FIFO Full	203CH	14
INT13	EXTINT Pin	203AH	13
INT12	TIMER2 Overflow	2038H	12
INT11	TIMER2 Capture	2036H	11
INT10	4th Entry into HSI FIFO	2034H	10
INT09	RI	2032H	9
INT08	TI	2030H	8
SPECIAL	Unimplemented Opcode	2012H	N/A
SPECIAL	Trap	2010H	N/A
INT07	EXTINT	200EH	7
INT06	Serial Port	200CH	6
INT05	Software Timer	200AH	5
INT04	HSI.0 Pin	2008H	4
INT03	High Speed Outputs	2006H	3
INT02	HSI Data Available	2004H	2
INT01	A/D Conversion Complete	2002H	1
INT00	Timer Overflow	2000H	0

Figure 14. Interrupt Vector Locations

The 8 lowest priority interrupts (INT0-INT7) and the TRAP instruction are identical to the interrupts on the 8096. Many of the new interrupt vectors were created by separating vectors which were formerly tied to the same interrupt. These include: Transmit Interrupt, Receive Interrupt, HSI FIFO Full, Timer2 Overflow and EXTINT (as opposed to P0.7). The new interrupts added are:

1. HSI FIFO (not including holding register) has 4 or more entries
2. Timer 2 Capture occurred (P2.7 rising edge).

Processing of interrupts is controlled by the Interrupt Pending Registers, the Interrupt Mask Registers, and the Global Disable Bit. The Interrupt Pending Registers (shown in Figure 15) have one bit for each interrupt vector. If a transition occurs to trigger a particular interrupt, the associated bit in the pending register is set. When a vector to an interrupt routine is taken, the associated pending bit is cleared.

The Interrupt Mask Registers (IMASK, IMASK1) have bits to correspond to each interrupt and are set up identically to the Interrupt Pending Registers. Each mask bit can be set or cleared in software to enable or disable individual interrupts. These registers are also referred to as INT_MASK and INT_MASK1.

PSW bit 9, the global Interrupt Disable Bit, controls the entire interrupt structure. When it is cleared, all interrupts are disabled except NMI, TRAP and unimplemented opcode. When it is set and an interrupt is both pending and unmasked (Ipend.x = 1, Imask.x = 1), the interrupt service procedure begins. The highest priority interrupt which is pending and unmasked is the first to occur. Interrupt servicing involves a call to the address stored in the interrupt vector location and clearing of the interrupt pending bit.

The interrupt mask registers can be used to enable and disable specific interrupts from occurring under software control. By using this feature a programmer can determine which interrupt sources can interrupt which interrupt routines. There are 6 instructions which facilitate this by not allowing interrupts to occur immediately after them:

EI, DI — Enable and disable all interrupts by toggling the global disable bit (PSW.9).

PUSHF — PUSH Flags pushes the PSW/IMASK pair then clears it, leaving both IMASK and PSW.9 clear.

POPF — POP Flags pops the PSW/IMASK pair off the stack

	7	6	5	4	3	2	1	0
IPEND1: IMASK1:	NMI	FIFO FULL	EXT INT	T2 OVF	T2 CAP	HSI4	RI	TI

	7	6	5	4	3	2	1	0
IPEND: IMASK1:	EXT INT	SER PORT	SOFT TIMER	HSI.1 PIN	HSO PIN	HSI DATA	A2D NONE	TIMER OVF

Figure 15. Interrupt Pending Registers

PUSHA — PUSH All does a PUSHF, then pushes the IMASK1/WSR pair and clears IMASK1

POPA — POP All pops the IMASK1/WSR pair and then does a POPF

Interrupts can also not occur immediately after execution of:

Unimplemented Opcodes

TRAP — The software trap instruction

SIGND — The signed prefix for multiply and divide instructions

PUSHA, PUSHF, and DI disable interrupts until software changes either the interrupt mask, PSW.9 or both. POPA, POPF, and EI can enable interrupts and are frequently used at the end of an interrupt routine, just prior to a RETurn. By preventing interrupts from occurring between these instructions and a RETurn, the RET is always executed and the stack will not build up needlessly.

Interrupts cannot occur immediately after unimplemented opcodes or the TRAP instruction, since the interrupt routine for these operations must have time to execute a PUSHF, PUSHA or DI. The SIGND prefix and the associated multiply or divide instructions must not be separated, so interrupts cannot occur after the SIGND opcode.

Setting and clearing the IPEND and IPEND1 registers is simplified since new interrupts are stored in buffer registers while read-modify- write operations are performed on IPEND and IPEND1. To set and clear bits in the pending registers the following sequences can be used:

```
ANDB IPEND, #11110111B;    Clear IPEND.3

ORB IPEND, #00000010B;     Set IPEND.1
```

Comparing the 80C196KA to the 8096, the interrupt response time has been improved as follows:

	External Stack		Internal Stack	
	8096	80C196KA	8096	80C196KA
States	24	18	21	16
8096 @ 12 MHz	6.00	—	5.25	—
80C196KA @ 8 MHz	—	4.5	—	4.00
80C196KA @ 12 MHz	—	3.0	—	2.67

Interrupt response time is measured as the elapsed time from the end of the previous instruction to the beginning of the first instruction of the interrupt service routine. It does not include the time needed to finish the current instruction or to save values on the stack.

2.6 INSTRUCTION SET AND PSW

All the instructions in the 8096 exist in the 80C196KA and perform the same function with two exceptions. First, the PSW bits are set in a specific manner for some operations where the 8096 PSW results were undefined. Second, some instructions execute in fewer state times.

PSW Settings

The PSW bits on the 80C196KA are set as follows:

PSW:	7	6	5	4	3	2	1	0
	Z	N	V	VT	C	X	I	ST

Z: The Zero flag is set to indicate that an operation generated a result equal to zero. The instructions SUBC(B) and ADDC(B) can only clear the Z flag but can not set it. This makes it easier to perform double word arithmetic, as a zero in the high word will not set the zero flag.

N: The Negative flag is set to indicate that the operation generated a negative result. Note that the N flag will be in the algebraically correct state even if an overflow occurs. For shift operations, including the normalize operation and all three forms (SHL, SHR, SHRA) of byte, word and double word shifts, the N flag will be set to the same value as the most significant bit of the result. This will be true even if the shift count is 0.

V: The oVerflow flag is set to indicate that the operation generated a result which is outside the range for the destination data type. For divide operations, the following conditions are set:

For the operation:	V is set if Quotient is:
UNSIGNED BYTE DIVIDE	> 255(0FFH)
UNSIGNED WORD DIVIDE	> 65535(0FFFFH)
SIGNED BYTE DIVIDE	< −127(81H) or > 127(7FH)
SIGNED WORD DIVIDE	< −32767(8001H) or > 32767(7FFFH)

VT: The oVerflow Trap flag is set when the V flag is set, but it is only cleared by the CLRVT, JVT and JNVT instructions. This allows testing for overflows in a group of operations instead of after each operation.

C: The Carry flag is set to indicate the state of the arithmetic carry from the most significant bit of the ALU for an arithmetic operation, or the state of the last bit shifted out of an operand for a shift. Arithmetic Borrow after a subtract operation is the complement of the C flag (i.e. if the operation generated a borrow then C=0.)

X: Reserved for future. Should always be cleared when writing to the PSW for compatibility with future products.

I: The global Interrupt disable bit disables all interrupts except NMI when cleared.

ST: The STicky bit is set to indicate that during a right shift a one has been shifted into the Carry flag and then has been shifted out. This flag can be used with the carry flag to determine rounding.

Instruction Set Additions

Six instructions have been added to the 8096 instruction set to form the 80C196KA instruction set. The added instructions are:

PUSHA — PUSHes the PSW, INT__MASK, IMASK1, and WSR

POPA — POPs the PSW, INT__MASK, IMASK1, and WSR

IDLPD — Sets the part into IDLE or Powerdown mode

DJNZW — Decrement Jump Not Zero using a Word counter

CMPL — Compare 2 long direct values

BMOV — Block move using 2 auto-incrementing pointers and a counter

Descriptions of these new instructions follow:

1. **PUSHA** (push all): This instruction is used instead of PUSHF to support the 8 additional interrupts. It is similar to PUSHF, but pushes two words instead of one. The first word pushed is the same as for the PUSHF instruction, PSW/INT__MASK. The second word pushed is formed by the IMASK1/WSR register pair. As a result of this instruction the PSW, INT__MASK, and IMASK1 registers are cleared, and the SP is decremented by 4. Interrupts are disabled in two ways by this instruction since both PSW.9 and the interrupt masks are cleared. Interrupts cannot occur between this instruction and the one following it.

```
execution:  SP ← SP - 2
            (SP) ← PSW/INT_MASK
            PSW/INT_MASK ← 0
            SP ← SP - 2
            (SP) ← IMASK1/WSR
            IMASK1 ← 0

   assembly language format: PUSHA
   object code format: <11110100>

      bytes:  1
      states: on-chip stack:12
              off-chip stack:18
```

PSW:

Z	N	V	VT	C	X	I	ST
0	0	0	0	0	X	0	0

2. **POPA** (pop all): This instruction is used instead of POPF to support the 8 additional interrupts. It is similar to POPF, but pops two words instead of one. The first word is popped into the IMASK1/WSR register pair, while the second word is popped into the PSW/INT__MASK register pair. As a result of this instruction the SP is incremented by 4. Interrupts can not occur between this instruction and the one following it.

```
execution:  IMASK1/WSR ← (SP)
            SP ← SP + 2
            PSW/INT_MASK ← (SP)
            SP ← SP + 2

   assembly language format: POPA
   object code format: <11110101>
```

```
bytes:  1
states: on-chip stack: 12
off-chip stack: 18
```

PSW:	Z	N	V	VT	C	x	I	ST
	✓	✓	✓	✓	✓	x	✓	✓

(✓ = changed)

3. **IDLPD** (idle/powerdown): This instruction is used for entry into the idle and powerdown modes. Selecting IDLE or POWERDOWN is done using the key operand. If the operand is not a legal key, the part executes a reset sequence. The bus controller will complete any prefetch cycle in progress before the CPU stops or resets.

```
execution:  if KEY = 1 then enter IDLE
            else if KEY = 2 then enter
             POWERDOWN
            else execute reset.
```

```
assembly language format: IDLPD #key (key is 8-bit value)
object code format: <11110110> <key>
```

```
bytes:  2
states: legal key:   8
        illegal key: 25
```

PSW:	Z	N	V	VT	C	x	I	ST
Legal Key	-	-	-	-	-	x	-	-
Illegal Key	0	0	0	0	0	x	0	0

(- = Unchanged)

4. **DJNZW** (decrement and jump if not zero word): This instruction is the same as the DJNZ except that the count is a word operand. A counter word is decremented; if the result is not zero the jump is taken. The range of the jump is −128 to +127.

```
execution:  COUNT ← COUNT - 1
            if COUNT <> 0 then
            PC ← PC + disp (sign extended)
```

```
assembly language format: DJNZW wreg,cadd
object code format: <11100001> <wreg> <disp>
```

```
bytes:  3
states: jump not taken: 5
        jump taken: 9
```

PSW:	Z	N	V	VT	C	x	I	ST
	-	-	-	-	-	x	-	-

5. **CMPL** (compare long): This instruction is used to compare the magnitudes of two double word (long) operands. The operands are specified using the direct addressing mode. Five PSW flags are set following this operation, but the operands are not affected.

```
execution:  DST - SRC

                                      DST  SRC
assembly language format:    CMPL Lreg,Lreg
object code format: <11000101> <src Lreg> <dst Lreg>

   bytes:  3
   states: 7
```

PSW:

Z	N	V	VT	C	x	I	ST
✓	✓	✓	✓	✓	x	-	-

6. **BMOV** (block move): This instruction is used to move a block of word data from one location in memory to another. The source and destination addresses are calculated using the indirect with auto- increment addressing modes. A long register addresses the source and destination pointers which are stored in adjacent word registers. The number of transfers is specified by a word register. The blocks of data can reside anywhere in memory but should not overlap.

```
execution:  COUNT ← (CNTREG)
            LOOP: SRCPTR ← (PTRS)
            DSTPTR ← (PTRS + 2)
            (DSTPTR) ← (SRCPTR)
            (PTRS) ← SRCPTR + 2
            (PTRS + 2) ← DSTPTR + 2
            COUNT ← COUNT - 1
            if COUNT <> 0 then go to LOOP

                                     PTRS CNTREG
assembly language format: BMOV  Lreg,wreg
object code format: <11000001> <wreg> <Lreg>

   bytes:  3
   states: internal/internal:  8 per transfer + 6
           external/internal: 11 per transfer + 6
           external/external: 14 per transfer + 6
```

PSW:

Z	N	V	VT	C	x	I	ST
-	-	-	-	-	x	-	-

Notes:

1. CNTREG does not get decremented during the instruction
2. It is easy to unintentionally create a very long un-interruptable operation with this instruction.

To provide an interruptable version of BLKMOV for large blocks, the BLKMOV instruction can be used with the DJNZ(W) instruction. This is possible because the pointers are modified, but CNTREG is not. Consider the example:

```
LD     PTRS, SRC      ;Pointer to base of sources table
LD     PTRS+2, DST    ;Pointer to base of destination table
LD     CNTREG, #COUNT ;Number of words to move per set
LD     CNTSET, #SETS  ;Number of sets to move
BMOV   PTRS, CNTREG   ;Move one set
DJNZW  CNTSET, MOVE   ;Decrement set counters and move again
```

Addressing Modes

The instructions on the 80C196KA can be divided into 4 groups: no operand, one operand, two operand, and three operand. Two and three operand instructions, as well as the PUSH and POP instructions, can use multiple addressing modes, the remaining instructions can operate on any of the bytes in the register file or SFR space.

To indicate the address range for the operands of each instruction the letters "D", "B", and "A" are used. "D" is the destination register and must be in the register file or SFR space. "A" is the second operand. It is addressed using one of the six addressing modes and can be located anywhere in memory. "B" is the third operand for three operand instructions and must be located in the register file or SFR space. Three operand instructions reduce the number of temporary variables needed and therefore the number of move operations, speeding up the code for many applications.

The address modes usable with "A" operands are listed below:

Direct - The operand is specified by an 8-bit address field in the instruction. The operand must be in the Register File or SFR space.

Immediate - The operand itself follows the opcode in the instruction stream as immediate data. The immediate data can be either 8 or 16 bits wide.

Indirect - An 8-bit address field in the instruction contains the 7-bit address of a word in the Register File which contains the 16-bit address of the operand. The operand can be anywhere in memory

Indirect With Auto-Increment - Same as indirect, except that after the operand is referenced, the word register which contained its address is incremented by one if the operand is a byte or by two if it is a word.

Indexed (Long and Short) - The instruction contains an 8-bit address field and either an 8-bit or 16-bit displacement field. The 8-bit address field gives the 7-bit address of a word in the Register File which contains a 16-bit base address. The 8-bit or 16-bit displacement field contains a signed displacement which is added to the base address to produce the address of the operand. The operand can be anywhere in memory.

NOTE:
The indexed address mode can be used with the Zero Register to directly address any location in memory. It can also be used with the Stack Pointer to address variables on the stack.

The indexed and indirect modes of addressing on the 80C196KA operate in fewer state times than they do on the 8096 because of the extra 16-bit internal bus.

Figures 16 and 17 show a summary of the instructions available on the 80C196KA and the number of state times each requires to execute. Timing values for jumps, calls and returns include the time required to flush the instruction queue and to fetch the opcode at the destination address.

The instruction times listed are the minimum number of state times required for execution. (A state time is 2 oscillator periods.) This number could increase if wait states are used or if the opcode and its operands are not prefetched and residing in the instruction queue when they are needed. The instruction queue is almost never empty when running in the 16-bit bus mode without wait states, so the minimum number of state times is almost always the correct execution time.

As would be expected, some performance degradation occurs when using wait states or the 8-bit bus since the queue may become empty. It is very difficult to predict the exact queue status at all times, so the instruction timings can not be exactly predicted, only minimum and worst case timings can be calculated.

When adding wait-states, the number of wait-states used, multiplied by the number of instruction fetches and data accesses occurring, must be added to the instruction execution timing. This will provide the worst-case timing for an instruction sequence, the actual timing will be between the minimum timing and the worst-case timing.

In the 8-bit bus mode, the worst case timing, assuming no wait-states, can be calculated by adding the following to the minimum timings:

2 state times for each external word write

1 state time for each external word read

1 state time for each byte that is not in the queue when needed (worst case is the number of bytes in an instruction minus 1)

Instruction execution in the 8-bit mode typically takes 20 to 30 percent longer than in the 16-bit mode.

Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	✓	✓	✓	✓	↑	–	
ADD/ADDB	3	D ← B + A	✓	✓	✓	✓	↑	–	
ADDC/ADDCB	2	D ← D + A + C	↓	✓	✓	✓	↑	–	
SUB/SUBB	2	D ← D – A	✓	✓	✓	✓	↑	–	
SUB/SUBB	3	D ← B – A	✓	✓	✓	✓	↑	–	
SUBC/SUBCB	2	D ← D – A + C – 1	↓	✓	✓	✓	↑	–	
CMP/CMPB	2	D – A	✓	✓	✓	✓	↑	–	
MUL/MULU	2	D,D + 2 ← D × A	–	–	–	–	–	–	2
MUL/MULU	3	D,D + 2 ← B × A	–	–	–	–	–	–	2
MULB/MULUB	2	D,D + 1 ← D × A	–	–	–	–	–	–	3
MULB/MULUB	3	D,D + 1 ← B × A	–	–	–	–	–	–	3
DIVU	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	✓	↑	–	2
DIVUB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	✓	↑	–	3
DIV	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	✓	↑	–	
DIVB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	✓	↑	–	
AND/ANDB	2	D ← D AND A	✓	✓	0	0	–	–	
AND/ANDB	3	D ← B AND A	✓	✓	0	0	–	–	
OR/ORB	2	D ← D OR A	✓	✓	0	0	–	–	
XOR/XORB	2	D ← D (ecxl. or) A	✓	✓	0	0	–	–	
LD/LDB	2	D ← A	–	–	–	–	–	–	
ST/STB	2	A ← D	–	–	–	–	–	–	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	–	–	–	–	–	–	3,4
LDBZE	2	D ← A; D + 1 ← 0	–	–	–	–	–	–	3,4
PUSH	1	SP ← SP – 2; (SP) ← A	–	–	–	–	–	–	
POP	1	A ← (SP); SP + 2	–	–	–	–	–	–	
PUSHF	0	SP ← SP – 2; (SP) ← PSW; PSW ← 0000H; I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ✓	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LJMP	1	PC ← PC + 16-bit offset	–	–	–	–	–	–	5
BR[indirect]	1	PC ← (A)	–	–	–	–	–	–	
SCALL	1	SP ← SP – 2; (SP) ← PC; PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LCALL	1	SP ← SP – 2; (SP) ← PC; PC ← PC + 16-bit offset	–	–	–	–	–	–	5

Figure 16. Instruction Summary

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	jump if C = 0	—	—	—	—	—	—	5
JE	1	jump if Z = 1	—	—	—	—	—	—	5
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 0	—	—	—	—	—	—	5
JNV	1	Jump if V = 1	—	—	—	—	—	—	5
JVT	1	Jump if VT= 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5,6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5,6
DJNZ/ DJNZW	1	D ← D − 1; If D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 − D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	✓	✓	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb - - - - - lsb ← 0	✓	✓	✓	✓	↑	—	7
SHR/SHRB/SHRL	2	0 → msb - - - - - lsb → C	✓	✓	✓	0	—	✓	7
SHRA/SHRAB/SHRAL	2	msb → msb - - - - - lsb → C	✓	✓	✓	0	—	✓	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	

Figure 16. Instruction Summary (Continued)

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
CLRVT	0	VT ← 0	–	–	–	–	0	–	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interupts (I ← 0)	–	–	–	–	–	–	
EI	0	Enable All Interupts (I ← 1)	–	–	–	–	–	–	
NOP	0	PC ← PC + 1	–	–	–	–	–	–	
SKIP	0	PC ← PC + 2	–	–	–	–	–	–	
NORML	2	Left shift till msb = 1;D ← shift count	✓	✓	0	–	–	–	7
TRAP	0	SP ← SP – 2; (SP) ← PC; PC ← (2010H)	–	–	–	–	–	–	9

PUSHA	1	SP ← SP-2; (SP) ← PSW;
		PSW ← 0000H; SP ← SP-2;
		(SP) ← IMASK1/WSR; IMASK1 ← 00H
			0	0	0	0	0	0	

POPA	1	IMASK1/WSR ← (SP); SP ← SP+2 PSW ← (SP); SP ← SP+2	✓	✓	✓	✓	✓	✓	

IDLPD	1	IDLE MODE IF KEY=1;
		POWERDOWN MODE IF KEY =2;
		CHIP RESET OTHERWISE
			–	–	–	–	–	–	

CMPL	2	D-A	✓	✓	✓	✓	↑	–	
BMOV	2	[PTR_HI]+ ← [PTR_LOW]+ ; UNTIL COUNT=0	–	–	–	–	–	–	

NOTES:

1. If the mnemonic ends in "B" a byte operation is performed, otherwise a word operation is done. Operands is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D,D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D,D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to word.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

Figure 16. Instruction Summary (Continued)

MNEMONIC	DIRECT	IMMED	INDIRECT		INDEXED	
			NORMAL*	A-INC*	SHORT*	LONG*
ADD (3-op)	5	6	7/9	8/10	7/9	8/10
SUB (3-op)	5	6	7/9	8/10	7/9	8/10
ADD (2-op)	4	5	6/8	7/9	6/8	7/9
SUB (2-op)	4	5	6/8	7/9	6/8	7/9
ADDC	4	5	6/8	7/9	6/8	7/9
SUBC	4	5	6/8	7/9	6/8	7/9
CMP	4	5	6/8	7/9	6/8	7/9
ADDB (3-op)	5	5	7/9	8/10	7/9	8/10
SUBB (3-op)	5	5	7/9	8/10	7/9	8/10
ADDB (2-op)	4	4	6/8	7/9	6/8	7/9
SUBB (2-op)	4	4	6/8	7/9	6/8	7/9
ADDCB	4	4	6/8	7/9	6/8	7/9
SUBCB	4	4	6/8	7/9	6/8	7/9
CMPB	4	4	6/8	7/9	6/8	7/9
MUL (3-op)	16	17	18/21	19/22	19/22	20/23
MULU (3-op)	14	15	16/19	17/20	17/20	18/21
MUL (2-op)	16	17	18/21	19/22	19/22	20/23
MULU (2-op)	14	15	16/19	17/20	17/20	18/21
DIV	26	27	28/31	29/32	29/32	30/33
DIVU	24	25	26/29	27/30	27/30	28/31
MULB (3-op)	12	12	14/17	15/18	15/18	16/19
MULUB (3-op)	10	10	12/15	12/16	12/16	14/17
MULB (2-op)	12	12	14/17	15/18	15/18	16/19
MULUB (2-op)	10	10	12/15	12/16	12/16	14/17
DIVB	18	18	20/23	21/24	21/24	22/25
DIVUB	16	16	18/21	19/22	19/22	20/23
AND (3-op)	5	6	7/9	8/10	7/9	8/10
AND (2-op)	4	5	6/8	7/9	6/8	7/9
OR (2-op)	4	5	6/8	7/9	6/8	7/9
XOR	4	5	6/8	7/9	6/8	7/9
ANDB (3-op)	5	5	7/9	8/10	7/9	8/10
ANDB (2-op)	4	4	6/8	7/9	6/8	7/9
ORB (2-op)	4	4	6/8	7/9	6/8	7/9
XORB	4	4	6/8	7/9	6/8	7/9
LD/LDB	4	5	5/7	6/8	6/8	7/9
ST/STB	4	5	5/7	6/8	6/8	7/9
LDBSE	4	4	5/7	6/8	6/8	7/9
LDBZE	4	4	5/7	6/8	6/8	7/9
BMOV	6+8 per word			6 + 11/14 per word		
PUSH (int stack)	6	7	9/12	10/13	10/13	11/14
POP (int stack)	8	—	10/12	11/13	11/13	12/14
PUSH (ext stack)	8	9	11/14	12/15	12/15	13/16
POP (ext stack)	11	—	13/15	14/16	14/16	15/17

*Times for (Internal/External) Operands

Figure 17a. Instruction Execution State Times

MNEMONIC		MNEMONIC	
PUSHF (int stack)	6	PUSHF (ext stack)	8
POPF (int stack)	7	POPF (ext stack)	10
PUSHA (int stack)	12	PUSHA (ext stack)	18
POPA (int stack)	12	POPA (ext stack)	18
TRAP (int stack)	16	TRAP (ext stack)	18
LCALL (int stack)	11	LCALL (ext stack)	13
SCALL (int stack)	11	SCALL (ext stack)	13
RET (int stack)	11	RET (ext stack)	14
CMPL	7	DEC/DECB	3
CLR/CLRB	3	EXT/EXTB	4
NOT/NOTB	3	INC/INCB	3
NEG/NEGB	3		
LJMP	7		
SJMP	7		
BR [indirect]	7		
JNST, JST	4/8 jump not taken/jump taken		
JNH, JH	4/8 jump not taken/jump taken		
JGT, JLE	4/8 jump not taken/jump taken		
JNC, JC	4/8 jump not taken/jump taken		
JNVT, JVT	4/8 jump not taken/jump taken		
JNV, JV	4/8 jump not taken/jump taken		
JGE, JLT	4/8 jump not taken/jump taken		
JNE, JE	4/8 jump not taken/jump taken		
JBC, JBS	5/9 jump not taken/jump taken		
DJNZ	5/9 jump not taken/jump taken		
DJNZW	5/9 jump not taken/jump taken		
NORML	8 + 1 per shift (9 for 0 shift)		
SHRL	7+ 1 per shift (8 for 0 shift)		
SHLL	7 + 1 per shift (8 for 0 shift)		
SHRAL	7 + 1 per shift (8 for 0 shift)		
SHR/SHRB	6 + 1 per shift (7 for 0 shift)		
SHL/SHLB	6 + 1 per shift (7 for 0 shift)		
SHRA/SHRAB	6 + 1 per shift (7 for 0 shift)		
CLRC	2		
SETC	2		
DI	2		
EI	2		
CLRVT	2		
NOP	2		
RST	15 (includes fetch of configuration byte)		
SKIP	3		
IDLPD	8/25 (proper key/improper key)		

Figure 17b. Instruction Execution State Times

3.0 PERIPHERAL DESCRIPTION

3.1 OVERVIEW

There are five major peripherals on the 80C196KA: the serial port, analog to digital converter, pulse-width-modulated output, standard I/O ports and the high speed I/O unit. With the exception of the high speed I/O unit (HSIO), each of the peripherals is a single unit that can be discussed without further separation. These peripherals will be described after the HSIO unit.

Four individual sections make up the HSIO and work together to form a very flexible timer/counter based I/O system. Included in the HSIO are a 16-bit timer (TIMER1), a 16-bit up/down counter (TIMER2), a programmable high speed input unit (HSI), and a programmable high speed output unit (HSO).

With very little CPU overhead the HSIO can measure pulse widths, generate waveforms, and create periodic interrupts. Depending on the application, it can perform the work of up to 18 timer/counters and capture/compare registers. Timer1 and Timer2 are used as the time bases for the HSIO. After describing their operation, the HSI and then the HSO will be discussed.

3.2 TIMERS

Timer1

Timer1 is a free-running timer which is incremented every eight state times, just as it is on the 8096. It can be read and written, but care must be taken when writing to it if the High Speed I/O (HSIO) Subsystem is being used. The precautions necessary when writing to Timer1 are described in the HSIO section. Timer1 can cause an interrupt when it overflows from 0FFFFH to 0000H if enabled by setting IOC1.2 = 1.

Timer2

Timer2 on the 80C196KA has many enhancements over Timer2 on the 8096. It counts transitions, both positive and negative, on its input which can be either the T2CLK pin or the HSI.1 pin depending on the state of IOC0.7. The maximum transition speed is once per state time in the Fast Increment mode, and once every 8 states otherwise. Timer2 can be read and written and can be reset by hardware, software or the HSO unit.

Interrupts can be generated if Timer2 crosses the 0FFFFH/0000H boundary or the 7FFFH/8000h boundary in either direction. By having two interrupt points it is possible to have interrupts enabled even if Timer2 is counting up and down centered around one of the interrupt points. The interrupt can be set to vector through location 2038H or 2000H using the interrupt mask registers and IOC1.3.

The value in Timer2 can be captured into the T2CAPture register by a rising edge on P2.7. T2CAP is located at 0CH in register plane 15. The interrupt generated by a capture vectors through location 2036H.

Timer2 can be placed in the Fast Increment mode by setting IOC2.0. In this mode it is not synchronized to the HSO unit and may not work properly with the HSO if transitions occur faster than every 8 states. In addition, HSO events based on Timer2 may not occur as expected if a count transition occurs within 8 state times before or after the timer is reset by other than an HSO event.

Timer2 can be made to count up or down based on the Port 2.6 pin if IOC2.1 = 1. However, caution must be used when this feature is working in conjunction with the HSO. If Timer2 does not complete a full cycle it is possible to have events in the CAM which never match the timer. These events would stay in the CAM until the CAM is cleared or the chip is reset.

The following control/status bits are associated with the Timer2:

	Bit = 1	Bit = 0
IOC0.1	Reset Timer2 each write	No action
IOC0.3	Enable external reset	Disable
IOC0.5	HSI.0 is ext. reset source	T2RST is reset source
IOC0.7	HSI.1 is T2 clock source	T2CLK is clock source
IOC1.3	Enable Timer2 overflow int.	Disable overflow interrupt
IOC2.0	Enable fast increment	Disable fast increment
IOC2.1	Enable downcount feature	Disable downcount
P2.6	Count down if IOC2.1 = 1	Count up
IOC2.5	Interrupt on 7FFFH/8000H	Interrupt on 0FFFFH/0000H
P2.7	Capture Timer2 into T2CAPture on rising edge	

3.3 HIGH SPEED INPUTS (HSI)

The High Speed Input (HSI) unit can capture the value of Timer1 when an event takes place on one of four input lines. Four types of events can trigger a capture; rising edges only, falling edges only, rising or falling edges or every eighth rising edge. Whenever the every eighth rising edge mode is entered the divide-by-8 counter is reset, allowing very fast pulses to be measured and counted. The input lines are sampled for events during every Phase1. A block diagram of this unit is shown in Figure 18.

Each of the input lines can be individually programmed to select the type of event to trigger on using the HSI_MODE register (shown in Figure 19). Several bits of the IOC0 register enable and disable the HSI lines, as well as control the inputs to Timer2. The function of these bits is shown in Figure 20.

When events occur, the Timer1 value and 4 status bits indicating which line(s) had events get stored in a 7 level fifo. The next event ready to be unloaded from the fifo is placed in the HSI Holding Register, so a total of 8 pieces of data can be stored in the fifo. If events occur after the fifo is full they will not be recorded and the fifo will contain the information gathered prior to the overflow error condition.

Data is taken off the fifo by reading the HSI_STATUS register, followed by reading the HSI_TIME register. When the high byte of the time register is read the next fifo location is loaded into the holding register, so reading HSI_TIME before HSI_STATUS will result in getting the wrong status information. For convenience the HSI time register should be read as a word. The HSI unit is synchronized to Timer1 which increments every 8 state times. For this reason it is required that 8 state times elapse between reading HSI_TIME and the next HSI_STATUS. The HSI_STATUS register, shown in Figure 21, also contains bits which indicate the level of the HSI pins at the time that HSI_STATUS is read.

The HSI can generate interrupts in three ways: each time a value moves from the fifo into the holding register; when the fifo (independent of the holding register) has 4 or more events stored; when the fifo has 6 or more events stored. The first case is called FIFO_LOADED, the second is FIFO_4, and the last case is called FIFO_FULL. Either the FIFO_LOADED or the FIFO_FULL interrupts can be selected by IOC1.7 to vector through location 2004H. The FIFO_4 interrupt vectors through location 2034H, and the FIFO_LOADED interrupt vectors through location 203CH. An additional interrupt can be generated by a rising edge on the HSI.0 pin, even if the pin is not enabled to the HSI unit. This interrupt vectors through location 2008H.

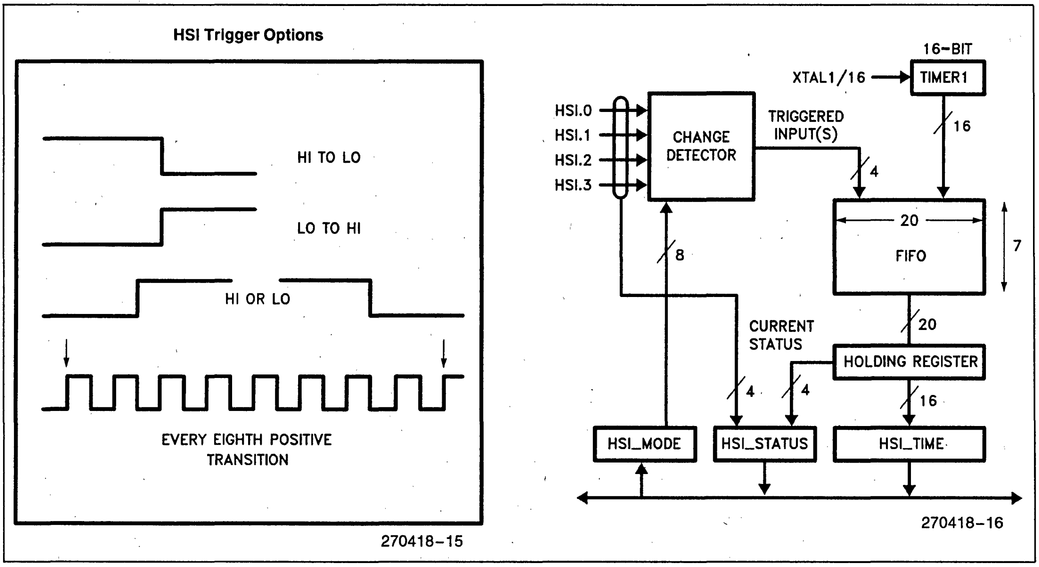


Figure 18. HSI Block Diagram

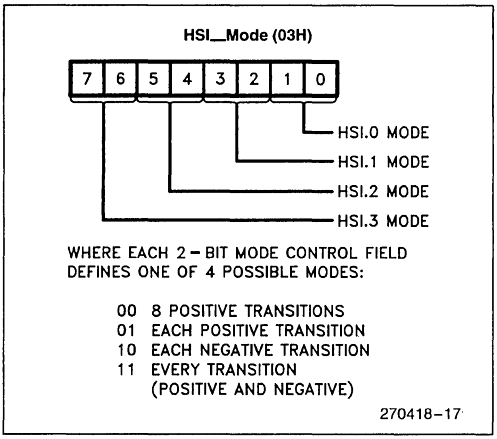

HSI_Mode (03H)
7
6
5
4
3
2
1
0
HSI.0 MODE
HSI.1 MODE
HSI.2 MODE
HSI.3 MODE
WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:
00 8 POSITIVE TRANSITIONS
01 EACH POSITIVE TRANSITION
10 EACH NEGATIVE TRANSITION
11 EVERY TRANSITION (POSITIVE AND NEGATIVE)
270418-17

Figure 19. HSI Mode Register

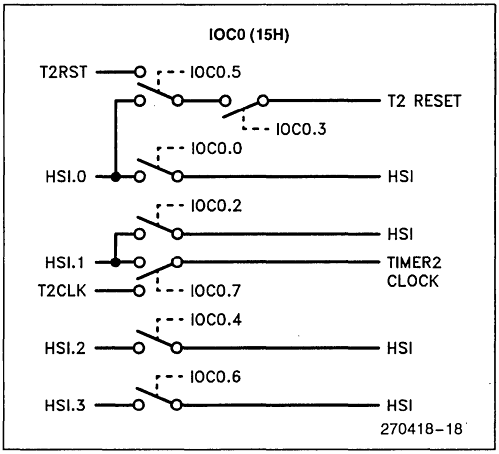


Figure 20. IOC0 Control of the HSI

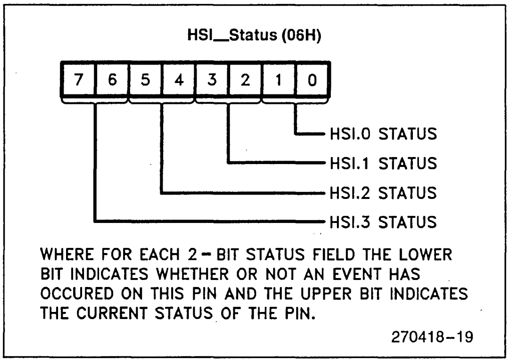

HSI_Status (06H)
7
6
5
4
3
2
1
0
HSI.0 STATUS
HSI.1 STATUS
HSI.2 STATUS
HSI.3 STATUS
WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.
270418-19

Figure 21. HSI Status Register

3.4 HIGH SPEED OUTPUTS (HSO)

The High Speed Output (HSO) unit can generate events at specified times or counts based on Timer1 or Timer2. A block diagram of the HSO unit is shown in Figure 22. Up to 8 pending events can be stored in the CAM (Content Addressable Memory) of the HSO unit at one time. Commands are placed into the HSO unit by first writing to HSO_COMMAND with the event to occur, and then to HSO_TIME with the timer match value. Although HSO_TIME is usually written as a word, it is the writing of the high byte which sends the command into the CAM. Since the HSO is synchronized to Timer1 and the HSI, 8 state times must elapse between writing to HSO_TIME and writing the next HSO_COMMAND.

Sixteen different types of events can be triggered by the HSO: 8 external and 8 internal. There are two interrupt vectors associated with the HSO. The one at 2006H is used for external events, the one at 200AH, called the Software Timer Interrupt, is used for internal events. External events consist of switching up to 6 lines, HSO.0 through HSO.5. These lines switch during Phase1. (Note that HSO.4 and HSO.5 are shared with HSI.2 and HSI.3.)

Internal events include setting up 4 Software Timers, resetting Timer2, and starting an A to D conversion. The software timers are flags that can be set by the HSO and optionally cause interrupts. The format for the HSO commands is shown in Figure 23. Note that commands 0C and 0D will act as additional software timer commands with no associated status bit. They are useful only if the interrupt bit (bit4) is set in the HSO_COMMAND.

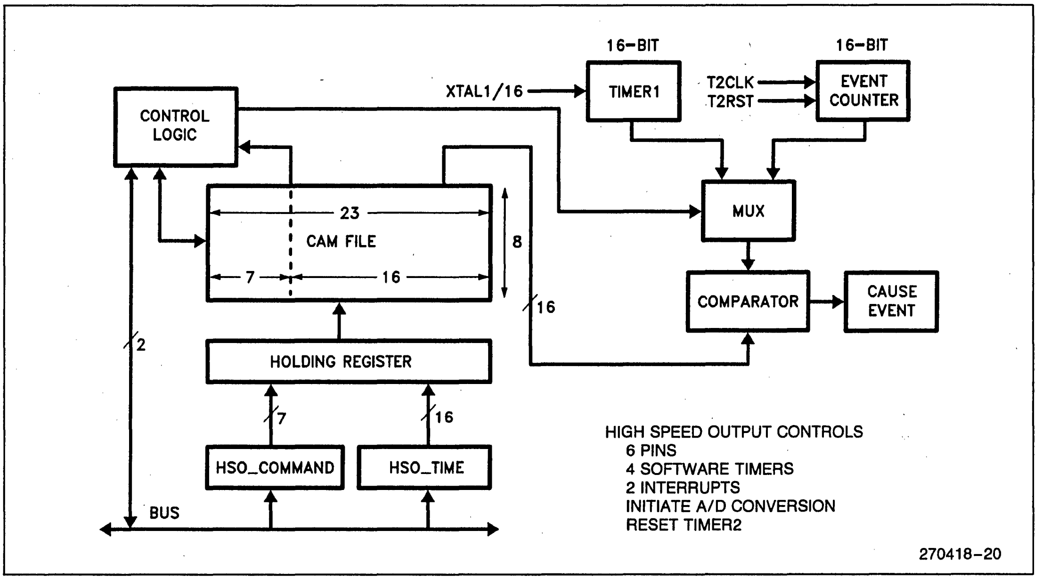


Figure 22. HSO Block Diagram

	7	6	5	4	3	2	1	1
HSO_ COMMAND	CAM LOCK	TMR2/ $\overline{\text{TMR1}}$	SET/ $\overline{\text{CLEAR}}$	INT/ $\overline{\text{INT}}$	CHANNEL			

CAM Lock — Locks event in CAM if this is enabled by IOC2.6 (ENA_LOCK)

TMR/$\overline{\text{TMR1}}$ — Events Based on Timer2 / Based on Timer1 if 0

SET/$\overline{\text{CLEAR}}$ — Set HSO line / Clear HSO line if 0

INT/$\overline{\text{INT}}$ — Cause Interrupt / No interrupt if 0

CHANNEL: (in Hex):		
	0–5:	HSO lines 0-5
	6 :	HSO lines 0 and 1
	7:	HSO lines 2 and 3
	8-B:	Software Timers 0-4
	C-D:	Unflagged Events
	E:	Reset Timer2
	F:	Start A to D conversion

Figure 23. HSO Command Register

The CAM Lock bit (HSO_Command.7) can be set to keep commands in the CAM, otherwise the commands will clear from the CAM as soon as they cause an event. This feature is best used to generate periodic events based on Timer2 and must be enabled by setting IOC2.6. To clear locked events from the CAM, the entire CAM must be cleared by writing a one to the CAM clear bit IOC2.7. A chip reset will also clear the CAM. It is possible to cancel individual external events by writing the opposite event to the CAM and setting it to occur at the same time. Both of these events will then remain in the CAM until the time tag is matched.

Since HSO events are dependent on exact matches of the timers with the values in the CAM, it is important to be very careful when using timers in any mode except continuous counting in one direction. If Timer2 is used in the Fast Count mode, the HSO should not be used if counts could occur faster than once every 8 state times.

A status register, IOS2, has been added to the 80C196KA to indicate which events have been generated by the HSO unit. IOS2 is cleared whenever it is accessed (a jump on bit is considered an access). The correspondence between the HSO events and the bits in the IOS2 is shown below.

IOS2:	7	6	5	4	3	2	1	0
	HC15	HC14	HSO.5	HSO.4	HSO.3	HSO.2	HSO.1	HSO.0

Bits 0 through 5 indicate that a command affecting the corresponding HSO pin was executed. Bits 6 and 7 indicate occurrence of HSO__CMD__14 and HSO__CMD__15 respectively (Reset Timer2 and Start A/D Converter.) This register clears on read.

The IOS0 register contains the status of the HSO lines. When WSR = 15, writing to this register changes the values on the HSO pins. However, the HSO can change this written value by executing a command. The IOS0 register format is shown below.

IOS0:	7	6	5	4	3	2	1	0
	H.REG	CAM	HSO.5	HSO.4	HSO.3	HSO.2	HSO.1	HSO.0

Bits 0 through 5 indicate the state of the I/O line. Bits 6 and 7 indicate that a space is available in the CAM and a space is available in the holding register, respectively.

3.5 SERIAL PORT

The serial port on the 80C196KA has three full-duplex asynchronous modes and one synchronous mode. All of the modes are compatible with the other MCS®-96 parts and members of the MCS®-51 product family. The synchronous mode is called Mode 0, the asynchronous modes are called Modes 1, 2 and 3. An independent baud rate generator determines the baud rate for all of the modes. The baud rate value is different than that used for the 8096.

Mode 0

Mode 0 synchronous operation uses the RXD pin to input or output data 8 bits at a time. TXD is used to output the clock signal. The low time of the clock is always two states except in the fastest mode. In the fastest mode, set by entering a 8001H into the baud register, the low and high times of the clock are each one state time. Figure 24 shows the relative timings of the serial port operating in Mode 0.

Mode 1

Mode 1 is the standard asynchronous serial communication mode. A 10-bit frame (shown in Figure 25) is transmitted or received using a start bit, 8 data bits, and a stop bit. If parity is enabled by setting PEN = 1, an even parity bit is sent instead of the 8th data bit and parity is checked on reception.

Mode 2

Mode 2 is the 9th bit recognition mode and is frequently used with Mode 3 in interprocessor communication. In this mode an 11-bit frame (shown in Figure 25) consisting of a start bit, 9 data bits, and a stop bit are sent and received. When transmitting, the 9th bit can be set using TB8. During reception the RI flag and interrupts will not be set unless the 9th data bit is high. Parity cannot be enabled in this mode.

Mode 3

Mode 3 uses the same 11-bit frame as Mode 2. When transmitting, parity can be enabled, providing 8 data bits and an even parity bit in place of the 9th data bit. When receiving, the RI bit is always set and the RB8 bit contains the value of the 9th data bit. If parity is enabled, (PEN = 1), the RB8/RPE bit will indicate a parity error if one occurs.

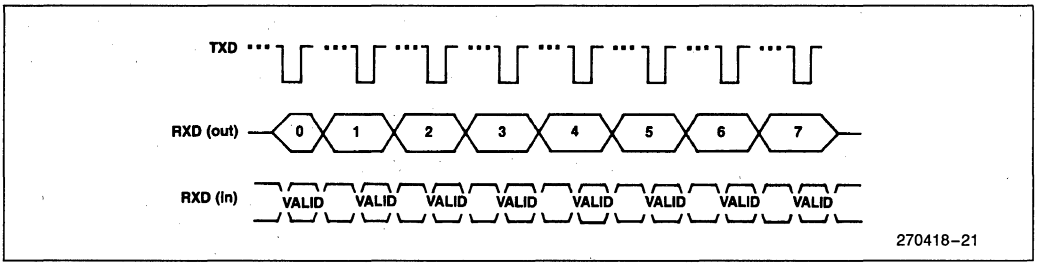


Figure 24. Serial Port Mode 0 Timings

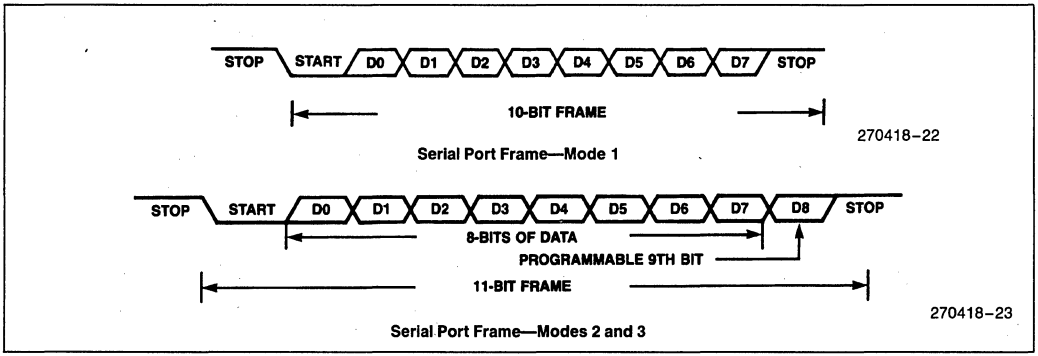


Figure 25. Serial Port Frames, Modes 1, 2 and 3

Baud Rates

Baud rates are generated based on either the T2CLK pin or XTAL1 pin. The values used are different than those used for the 8096 because the 80C196KA uses a divide-by-2 clock instead of a divide-by-3 clock to generate the internal timings. Baud rates are calculated using the following formulas where BAUD_REG is the value loaded into the baud rate register:

Asynchronous Modes 1, 2 and 3:

$$\text{BAUD\_REG} = \frac{\text{XTAL1}}{\text{Baud Rate} * 16} - 1 \quad \text{OR} \quad \frac{\text{T2CLK}}{\text{Baud Rate} * 8}$$

Synchronous Mode 0:

$$\text{BAUD\_REG} = \frac{\text{XTAL1}}{\text{Baud Rate} * 2} - 1 \quad \text{OR} \quad \frac{\text{T2CLK}}{\text{Baud Rate}}$$

The most significant bit in the baud register value is set to a one to select XTAL1 as the source. If it is a zero the T2CLK pin becomes the source. The following table shows some typical baud rate values:

BAUD RATES AND BAUD REGISTER VALUES

BAUD RATE	XTAL1 FREQUENCY		
	8.0 MHz	10.0 MHz	12.0 MHz
300	1666 / −0.02	2082 / 0.02	2499 / 0.00
1200	416 / −0.08	520 / −0.03	624 / 0.00
2400	207 / 0.16	259 / 0.16	312 / −0.16
4800	103 / 0.16	129 / 0.16	155 / 0.16
9600	51 / 0.16	64 / 0.16	77 / 0.16
19.2K	25 / 0.16	32 / 1.40	38 / 0.16

Baud Register Value / % error

A maximum baud rate of 750 Kbaud is available in the asynchronous modes with 12MHz on XTAL1. The synchronous mode has a maximum rate of 3.0 Mbaud with a 12 MHz clock. Location 0EH is the Baud Register. It is loaded sequentially in two bytes, with the low byte being loaded first. This register may not be loaded with zero in serial port Mode 0.

Serial Port Control

Reading the serial port is done through the Serial BUFfer receive (SBUF(RX)) register at location 7. This register is double buffered so data can continually be received. Writing to the serial port is done through SBUF(TX), also addressed at location 7. This register is double buffered on the 80C196KA to allow two bytes at a time to be written to the serial port.

Serial port control is done through the Serial Port CONtrol (SP_CON) register at location 11H. This register is write-only in Window 0 and has the following format:

SP_CON:	7	6	5	4	3	2	1	0
	X	X	X	TB8	REN	PEN	M2	M1

TB8 — Sets the ninth data bit for transmission. Cleared after each transmission. Not valid if parity is enabled

REN — Enables the receiver

PEN — Enables the Parity function (even parity)

M2,M1 — Sets the mode. Mode0 = 00, Mode1 = 01, Mode2 = 10; Mode3 = 11

The status of the serial port is read through the bits in the Serial Port STATus (SP_STAT) register, also at location 11H. Figure 21 shows the status bits of this register. On the 80C196KA the SP_STAT register contains new bits to indicate receive Overrun Error (OE), Framing Error (FE), and Transmiter Empty (TXE). The bits which were also present on the 8096 are the Transmit Interrupt (TI) bit, the Receive Interrupt (RI) bit, and the Received Bit 8 (RB8) or Receive Parity Error (RPE) bit. SP_STAT is read-only in Window 0 and has the following format:

SP_STAT	7	6	5	4	3	2	1	0
	RB8/ RPE	RI	TI	FE	TXE	OE	X	X

RB8 — Set if the 9th data bit is high on reception (parity disabled)

RPE — Set if parity is enabled and a parity error occurred

RI — Set at the end of the STOP bit reception

TI — Set at the beginning of the STOP bit transmission

FE — Set if no STOP bit is found at the end of a reception

TXE — Set if two bytes can be sent to SBUF(TX)

OE — Set if a byte is lost because SBUF was not read fast enough

The receiver on the 80C196KA checks for a valid stop bit. When one is detected, the data in the receive shift register is loaded into SBUF(RX). If a stop bit is not found within the appropriate time the Framing Error (FE) bit is set. In either case, the data in the receive shift register is loaded into SBUF(RX) and the RI bit is set. If this happens before the previous byte in SBUF(RX) is read, the Overflow Error (OE) bit is set. The data in SBUF(RX) will always be the latest byte received; it will never be a combination of the two bytes. When the RI bit is set it can cause an interrupt through the vectors at locations 200CH and 2032H. The RI, OE, and FE bits are reset when SP_STAT is read.

The Transmitter Empty (TXE) bit is set if the transmit FIFO is empty and ready to take up to two characters to be sent. TXE gets cleared as soon as a byte is written to SBUF. Two bytes may be written consecutively to SBUF if TXE is set. One byte may be written if TI alone is set. By definition, if TXE has just been set, a transmission has completed and TI will be set. When the TI bit is set it can cause an interrupt through the vectors at locations 200CH and 2032H. The user should not mask off this interrupt when using the double-buffered feature of the transmitter, as it could cause a missed count in the number of bytes being transmitted. The TI bit is reset when the CPU reads the SP_STAT registers.

3.6 A-TO-D CONVERTER

The 80C196KA A-to-D converter has 10 bits of resolution and can be run in modes compatible with either the 8096-90 or the 8096BH. Conversions can be performed on one of eight channels, the inputs of which share pins with port 0. The A to D includes a switchable Sample and Hold feature for the selected channel and does the conversion in as little as 91 state times.

Conversions are started by loading the AD_COMMAND register at location 02H with the channel number. The conversion can be started immediately be setting the GO bit to a one. If it is cleared the conversion will start when the HSO unit triggers it. The AD_COMMAND register has the following format:

A TO D_ COMMAND:	7	6	5	4	3	2	1	0
	X	X	X	X	GO	CHANNEL NUMBER		

The A-to-D converter can cause an interrupt to occur through the vector at location 2002H when it completes a conversion. It is also possible to use a polling method by checking the Status (S) bit in the lower byte of the AD_ RESULT register, also at location 02H. The status bit will be a 1 while a conversion is in progress. It takes 8 state times to set this bit after a conversion is started. The upper byte of the result register contains the most significant 8 bits of the conversion. The lower byte format is shown below:

A TO D_ RESULT_ LO:	7	6	5	4	3	2	1	0
	LOWEST 2 RESULT BITS		X	X	S	CHANNEL NUMBER		

At high crystal frequencies, more time is needed to allow the comparator to settle. For this reason IOC2.4 is provided to adjust the speed of the A-to-D conversion by disabling/enabling a clock prescaler. At low frequencies the leakage currents cause the sample and hold not to work, so IOC2.3 is provided to turn the sample and hold feature off.

A summary of the conversion time for the four options is shown below. The numbers represent the number of state times required for conversion, e.g., 91 states is 22.7 μs with an 8 MHz XTAL1 (providing a 250 ns state time.)

IOC2.3 1/0 = Sample and Hold off/on

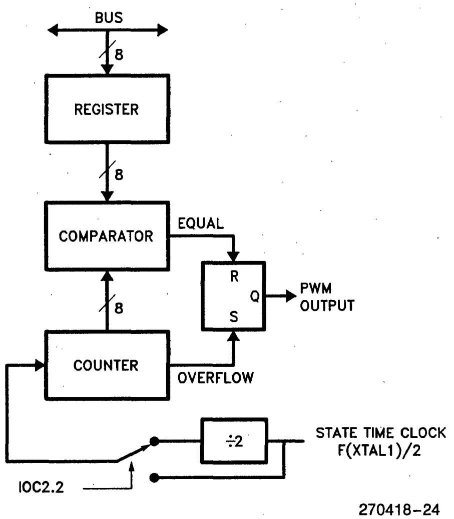
IOC2.4 1/0 = A to D Clock Prescaler off/on
10 MHz XTAL1 maximum with prescaler off

	Clock Prescaler On IOC2.4=0	Clock Prescaler Off IOC2.4=1	
IOC2.3=0 with S&H	158 states 26.33 μs @ 12 MHz	91 states 22.75 μs @ 8 MHz	91 states 18.2 μs @ 10 MHz
OC2.3=1 without S&H	293 states 48.83 μs @ 12 MHz	163 states 40.75 μs @ 8 MHz	163 states 32.6 μs @ 10 MHz

3.7 PULSE-WIDTH-MODULATION OUTPUT (PWM)

The PWM output unit is an 8-bit counter which increments every state time. When the counter equals zero the output is set high, when it equals the value in the PWM register (location 17H) the output goes low. This provides an approximation to an analog output for driving motors and other similar devices. A block diagram of the PWM unit and examples of PWM waveforms are shown in Figures 26 and 27 respectively. The 80C196KA PWM unit has a prescaler bit (divide by 2) which is enabled by setting IOC2.2=1. This allows the counter to have a period of 512 state times instead of 256. The PWM frequencies are as follows:

XTAL1 =	8 MHz	10 MHz	12 MHz
IOC2.2=0	15.6 KHz	19.6 KHz	23.6 KHz
IOC2.2=1	7.8 KHz	9.8 KHz	11.8 KHz



270418-24

• Duty Cycle Programmable in 256 Steps

Figure 26. PWM Block Diagram

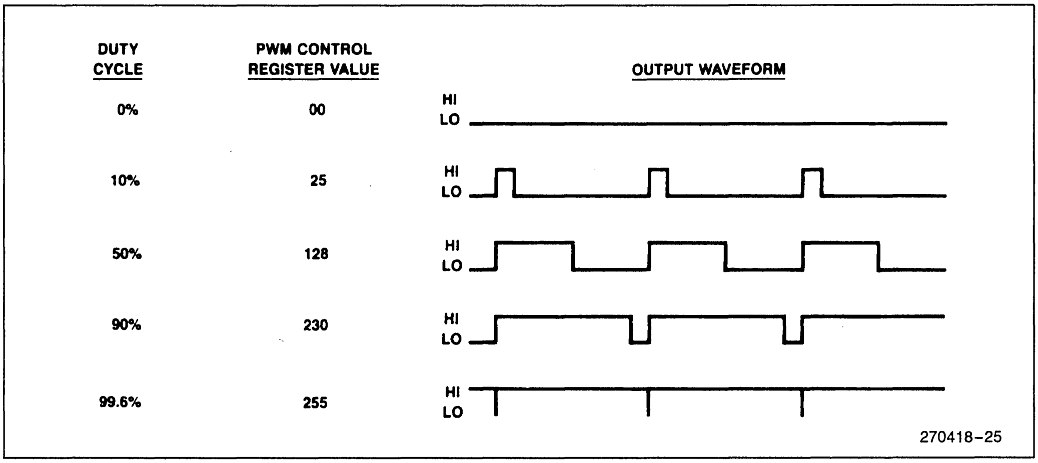


Figure 27. PWM Waveforms

3.8 STANDARD I/O PORTS

Five (5) 8-bit I/O ports are available on the 80C196KA. Port 0 (location 0EH) is an input only port which shares its pins with the A to D converter. Port 1 (location 0FH) is a quasi-bidirectional port. Port 2 (location 10H) has multiple functions on its pins as shown in Figure 28.

Quasi-bidirectional pins can be used as input and output pins without the need for a data direction register. They output a strong low value and a weak high value. The weak high value can be externally pulled low providing an input function. Figure 29 shows the configuration of a CHMOS quasi-bidirectional port. Note that it is not identical to the NMOS version.

Outputting a 0 on a quasi-bidirectional pin turns on the strong pull-down and turns off all of the pull-ups. When a 1 is output the pull-down is turned off and 3 pull-ups (strong-P1, weak-P3, very weak-P2) are turned on. Each time a pin switches from 0 to 1 transistor P1 turns on for two state times. P2 remains on until a zero is written to the pin. P3 is used as a latch, so it is turned on whenever the pin is above the threshold value (around 2 volts).

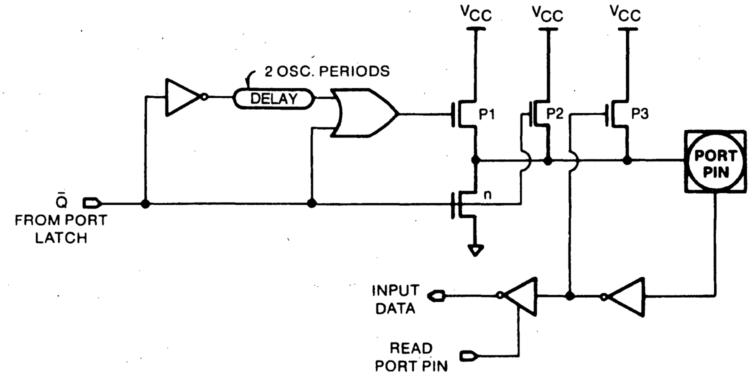
To reduce the amount of current which flows when the pin is externally pulled low, P3 is turned off when the pin voltage drops below the threshold. The current required to pull the pin from a high to a low is at its maximum just prior to the pull-up turning off. An external driver can switch these pins easily. The maximum current required occurs at the threshold voltage and is approximately 700 microamps.

Ports 3 and 4 are open drain I/O ports which share their pins with the System Bus. The port 3 and 4 pins will act as port pins if the $\overline{EA}$ pin is set for internal access and external memory is not being accessed. In all other cases the ports must be reconstructed with external hardware since the system bus uses the pins. Since external memory is always required with the 80C196KA, these ports must be reconstructed by placing latches at addresses 1FFE and 1FFFH in external memory. Future ROM and EPROM parts will be able to use the on-chip ports. By using the port reconstruction feature it is possible to build a multi-chip system which is exactly software compatible with a single-chip system.

PIN	FUNC.	ALTERNATE FUNCTION	CONTROL REG.
2.0	Output	TXD (Serial Port Transmit)	IOC1.5
2.1	Input	RXD (Serial Port Receive)	SPCON.3
2.3	Input	T2CLK (Timer2 Clock & Baud)	IOC0.7
2.4	Input	T2RST (Timer2 Reset)	IOC0.5
2.5	Output	PWM Output	IOC1.0
2.6	QBD*	Timer2 up/down select	IOC2.1
2.7	QBD*	Timer2 Capture	N/A

*QBD = Quasi-bidirectional

Figure 28. Port 2 Multiple Functions



270418-26

CHMOS Configuration. pFET 1 is turned on for 2 osc. periods after Q makes a 1-to-0 transition. During this time, pFET 1 also turns on pFET 3 through the inverter to form a latch which holds the 1. pFET 2 is also on.

Figure 29. CHMOS Quasi-bidirectional Port Circuit

3.9 USING THE ALTERNATE REGISTER WINDOW (WSR = 15)

I/O register expansion on the new CHMOS members of the MCS-96 family has been provided by making two register windows available. Switching between these windows is done using the Window Select Register (WSR). The PUSHA and POPA instructions can be used to push and pop the WSR and second interrupt mask when entering or leaving interrupts, so it is easy to change between windows.

On the 80C196KA only Window 0 and Window 15 are active. Window 0 is a true superset of the standard 8096 SFR space, while Window 15 allows the read-only registers to be written and write-only registers to be read. The only major exception to this is the Timer2 register which is the Timer2 capture register in Window 15. The writeable register for Timer2 is in Window 0. There are also some minor changes and cautions. The descriptions of the registers which have different functions in Window 15 than in Window 0 are listed below:

AD_COMMAND (02H)	— Read the last written command
AD_RESULT (02H, 03H)	— Write a value into the result register
HSI_MODE (03H)	— Read the value in HSI_MODE
HSI_TIME (04H,05H)	— Write to FIFO Holding register
HSO_TIME (04H,05H)	— Read the last value placed in the holding register
HSI_STATUS (06H)	— Write to status bits but not to HSI pin bits. (Pin bits are 1,3,5,7).
HSO_COMMAND (06H)	— Read the last value placed in the holding register
SBUF(RX) (07H)	— Write a value into the receive buffer
SBUF(TX) (07H)	— Read the last value written to the transmit buffer
WATCHDOG(0AH)	— Read the value in the upper byte of the WDT
TIMER1 (0AH,0BH)	— Write a value to Timer1
TIMER2 (0CH,0DH)	— Read/Write the Timer2 capture register. (Timer2 read/write is done with WSR = 0)
IOC2 (0BH)	— Last written value is readable, except bit 7 (note 1)
BAUD_RATE (0EH)	— No function, cannot be read
PORT0 (0EH)	— No function, no output drivers on the pins
SP_STAT (11H)	— Set the status bits, TI and RI can be set, but it will not cause an interrupt

SP_CON (11H) — Read the current control byte

IOS0 (15H) — Writing to this register controls the HSO pins. Bits 6 and 7 are inactive for writes.

IOC0 (15H) — Last written value is readable, except bit 1 (note 1)

IOS1 (16H) — Writing to this register will set the status bits, but not cause interrupts. Bits 6 and 7 are not functional

IOC1 (16H) — Last written value is readable

IOS2 (17H) — Writing to this register will set the status bits, but not cause interrupts.

PWM_CONTROL (17H) — Read the duty cycle value written to PWM_CONTROL

Note:

1. IOC2.7 (CAM CLEAR) and IOC0.1 (T2RST) are not latched and will read as a 1 (precharged bus) .

Being able to write to the read-only registers and vice-versa provides a lot of flexibility. One of the most useful advantages is the ability to set the timers and HSO lines for initial conditions other than zero.

3.10 SFR BIT SUMMARY

A summary of the SFRs which control I/O functions has been included in this section. The summary is separated into a list of those SFRs which have changed on the 80C196KA and a list of those which have remained the same.

The following 80C196KA SFRs are different than those on the 8096BH:

(The Read and Write comments indicate the register's function in Window 0 unless otherwise specified.)

SBUF(TX) — Now double buffered
07h
write

BAUD RATE — Uses new Baud Rate Values
0Eh
write

SP_STAT:

7	6	5	4	3	2	1	0
RB8/ RPE	RI	TI	FE	TXE	OE	X	X

11h
read

RPE :	Receive Parity Error
RI :	Receive Indicator
TI :	Transmit Indicator
FE :	Framing Error
TXE :	Transmitter Empty
OE :	Receive Overrun Error

IPEND1:
IMASK1:

7	6	5	4	3	2	1	0
NMI	FIFO FULL	EXT INT	T2 OVF	T2 CAP	HSI4	RI	TI

12h,13h
read/write

NMI :	Non-Maskable Interrupt
FIFO FULL :	HSIO FIFO full
EXTINT :	External Interrupt Pin
T2OVF :	Timer2 Overflow
T2CAP :	Timer2 Capture
HSI4 :	HSI has 4 or more entries in FIFO
RI :	Receive Interrupt
TI :	Transmit Interrupt

WSR:

7	6	5	4	3	2	1	0
X	X	X	X	W	W	W	W

14h
read/write

WWWW = 0 : SFRs function like a superset of 8096 SFRs

WWWW = 15 : Exchange read/write registers

WWWW = OTHER : Undefined, do not use

XXXX = 0000B : These bits must always be written as zeros to provide compatibility with future products.

IOS2:

7	6	5	4	3	2	1	0
START A2D	T2 RESET	HSO.5	HSO.4	HSO.3	HSO.2	HSO.1	HSO.0

17h
read

Indicates which HSO event occured

START A2D : HSO__CMD 15, start A to D

T2RESET : HSO__CMD 14, Timer 2 reset

HSO.0-5 : Output pins HSO.0 through HSO.5

IOC2:

7	6	5	4	3	2	1	0
CLEAR CAM	ENA LOCK	T2ALT INT	A2D CPD	NOSH	SLOW PWM	T2UD ENA	FAST T2EN

0Bh
write

CLEAR__CAM : Clear Entire CAM

ENA__LOCK : Enable lockable CAM entry feature

T2ALT INT : Enable T2 Alternate Interrupt at 8000H

A2D__CPD : Clock Prescale Disable for low XTAL frequency (A to D conversion in fewer state times)

NOSH : Disable A/D Sample and Hold

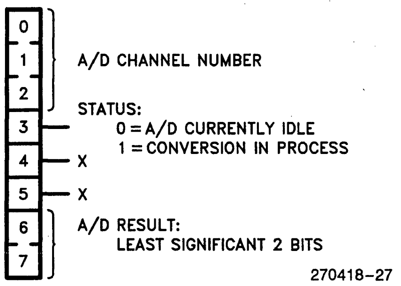
SLOW__PWM : Turn on divide by 2 Prescaler on PWM

T2UD ENA : Enable Timer 2 as up/down counter

FAST__T2EN : Enable Fast increment of T2; once per state time.

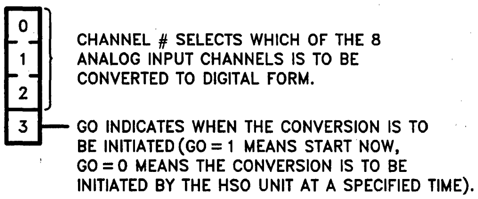
The following registers are the same on the 80C196KA as they were on the 8096BH:

A/D Result LO (02H)



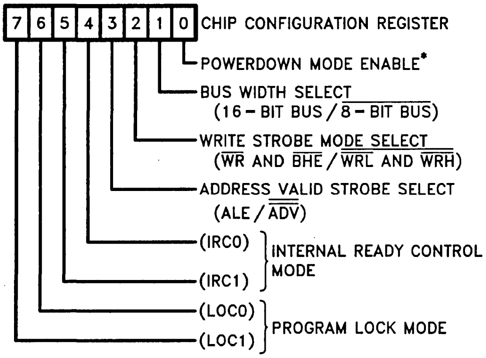
270418-27

A/D Command (02H)


0
1
2
3
CHANNEL # SELECTS WHICH OF THE 8 ANALOG INPUT CHANNELS IS TO BE CONVERTED TO DIGITAL FORM.
GO INDICATES WHEN THE CONVERSION IS TO BE INITIATED (GO = 1 MEANS START NOW, GO = 0 MEANS THE CONVERSION IS TO BE INITIATED BY THE HSO UNIT AT A SPECIFIED TIME).

270418-30

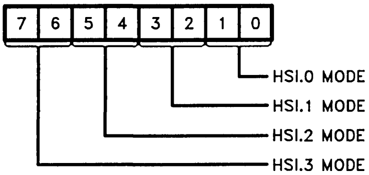
Chip Configuration (2108H)



270418-29

*Minor Change

HSI_Mode (03H)

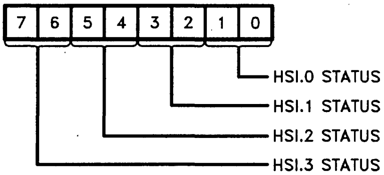


WHERE EACH 2-BIT MODE CONTROL FIELD
DEFINES ONE OF 4 POSSIBLE MODES:

00 8 POSITIVE TRANSITIONS
01 EACH POSITIVE TRANSITION
10 EACH NEGATIVE TRANSITION
11 EVERY TRANSITION
(POSITIVE AND NEGATIVE)

270418-28

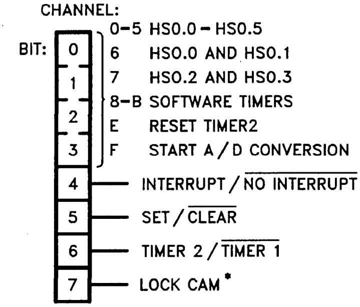
HSI_Status (06H)



WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

270418-31

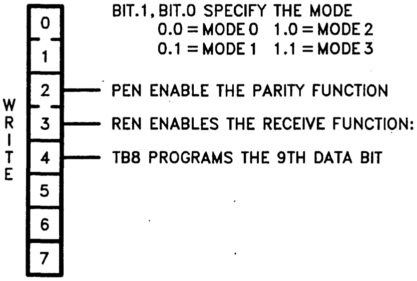
HSO Command (06H)



270418-32

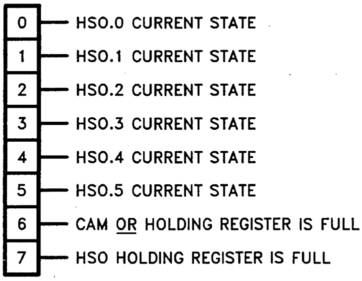
*Minor Change

SPCON (11H)

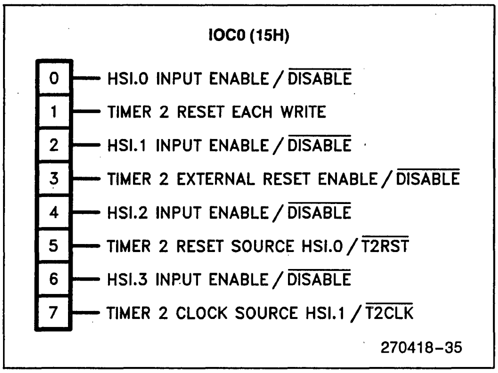

0 1 2 3 4 5 6 7
WRITE
BIT.1, BIT.0 SPECIFY THE MODE
0.0 = MODE 0 1.0 = MODE 2
0.1 = MODE 1 1.1 = MODE 3
PEN ENABLE THE PARITY FUNCTION
REN ENABLES THE RECEIVE FUNCTION:
TB8 PROGRAMS THE 9TH DATA BIT

270418-33

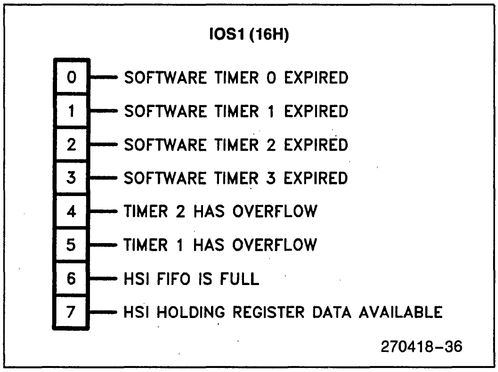
IOS0 (15H)



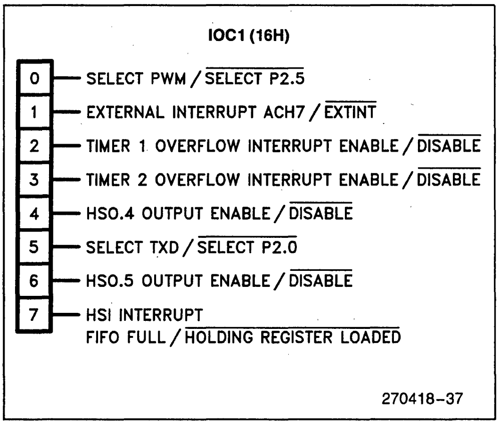
270418-34


IOC0 (15H)
0 HSI.0 INPUT ENABLE / $\overline{\text{DISABLE}}$
1 TIMER 2 RESET EACH WRITE
2 HSI.1 INPUT ENABLE / $\overline{\text{DISABLE}}$
3 TIMER 2 EXTERNAL RESET ENABLE / $\overline{\text{DISABLE}}$
4 HSI.2 INPUT ENABLE / $\overline{\text{DISABLE}}$
5 TIMER 2 RESET SOURCE HSI.0 / $\overline{\text{T2RST}}$
6 HSI.3 INPUT ENABLE / $\overline{\text{DISABLE}}$
7 TIMER 2 CLOCK SOURCE HSI.1 / $\overline{\text{T2CLK}}$

270418-35


IOS1 (16H)
0 SOFTWARE TIMER 0 EXPIRED
1 SOFTWARE TIMER 1 EXPIRED
2 SOFTWARE TIMER 2 EXPIRED
3 SOFTWARE TIMER 3 EXPIRED
4 TIMER 2 HAS OVERFLOW
5 TIMER 1 HAS OVERFLOW
6 HSI FIFO IS FULL
7 HSI HOLDING REGISTER DATA AVAILABLE

270418-36


IOC1 (16H)
0 SELECT PWM / $\overline{\text{SELECT P2.5}}$
1 EXTERNAL INTERRUPT ACH7 / $\overline{\text{EXTINT}}$
2 TIMER 1 OVERFLOW INTERRUPT ENABLE / $\overline{\text{DISABLE}}$
3 TIMER 2 OVERFLOW INTERRUPT ENABLE / $\overline{\text{DISABLE}}$
4 HSO.4 OUTPUT ENABLE / $\overline{\text{DISABLE}}$
5 SELECT TXD / $\overline{\text{SELECT P2.0}}$
6 HSO.5 OUTPUT ENABLE / $\overline{\text{DISABLE}}$
7 HSI INTERRUPT FIFO FULL / $\overline{\text{HOLDING REGISTER LOADED}}$

270418-37

4.0 OPERATING MODES

4.1 IDLE MODE

When the IDLE mode is entered, using the instruction "IDLPD #1", the CPU stops executing. The CPU clocks are frozen at logic state zero, but the peripheral clocks and CLKOUT continue to be active. CLKOUT logically equals the Phase2 signal that is supplied to the peripherals. System bus control signals ALE, $\overline{\text{RD}}$, $\overline{\text{WR}}$, INST. and $\overline{\text{BHE}}$ go to their inactive states and the bus becomes high impedance unless it was being used as ports 3 and 4. Power consumption in this mode is about 40% of that in the normal mode, since only the peripherals are running.

The interrupt controller and all peripherals, except Ports 3 and 4, continue to function during IDLE mode. If the chip was executing out of internal memory, Ports 3 and 4 will retain the data present in their data latches, otherwise these pins will be high impedance and their input buffers will be turned off. (See the Standard I/O Port section for more information about Ports 3 and 4.)

It is important to note that the Watchdog Timer continues to operate in the IDLE mode (if it was enabled after reset). This means the chip must wake up the CPU approximately every 64K state times (16 milliseconds at 8 MHz XTAL1) in order to reset this timer.

The CPU can be awakened by any enabled interrupt source or a hardware reset. Since all of the peripherals are running, this interrupt can be generated by the HSI, HSO, timer overflow, serial port, extint, or other similar interrupts. If an interrupt brings the CPU out of IDLE mode, the first action taken will be to place the program counter on the stack and jump to the interrupt service routine. When the interrupt service routine is done, the instruction executed is the one following IDLPD instruction which put the chip in the IDLE mode.

4.2 POWERDOWN MODE

When the POWERDOWN mode is entered, using the instruction "IDLPD #2", all internal clocks are frozen at logic state zero and the oscillator is turned off. All registers and most peripherals hold their values if V_{CC} is not removed from the part. The bus control signals go to their inactive states, and power is reduced to just the device leakage.

All the bidirectional or output-only port pins (including HSO, PWM, serial port, etc.) will assume values present in their respective data latches, except Ports 3 and 4. In this way the user controls the logic state of the port pins. The Port 3 and 4 pins will have values of the port latches if the chip was executing out of internal memory (future ROM and EPROM parts only), otherwise the pins will be in a high-impedance state with input buffers shut off.

All peripherals should be in an inactive state before putting the chip in powerdown. If the A to D converter is in the middle of a conversion it is aborted. The

HSIO, timers (Timer1 and Timer2), and the serial port stop in POWERDOWN mode. If the chip comes out of POWERDOWN by an external interrupt, the serial port will continue from where it left off with a chance of erroneous data transmitted or received. Therefore, the user must shut off the transmitter (not write anything to it) and the receiver (REN = 0) before putting the chip in POWERDOWN.

When the chip is in Powerdown, it is impossible to time out the Watchdog Timer or detect oscillator failure. Therefore, systems which will use Powerdown should not enable the Watchdog Timer and the systems using the Watchdog Timer should not go into Powerdown, unless the Watchdog is always reset immediately before entering and after exiting Powerdown.

To prevent accidental entry into Powerdown, the Powerdown feature can be disabled at reset by clearing bit 0 of the Chip Configuration Register (CCR). Since the default value of the Configuration Byte is 0FFH, Powerdown is normally enabled.

When in Powerdown, almost the entire state of the 80C196KA will be preserved, not just the most significant 16 bytes of register file. The V_{CC} (not V_{PD}) is used to supply power to the chip, so it must remain within specifications if the chip status is to be maintained. Certain SFRs, may contain incorrect information when the chip comes out of Powerdown. SFRs which could do this are the A/D result and serial port registers since the functions of these registers are real-time dependent and CPU-time stops in Powerdown mode. A/D commands in progress are aborted when coming out of Powerdown. It is the users responsibility to handle the serial port.

The Powerdown mode can be exited using either $\overline{\text{RESET}}$ or an external interrupt pin. If the $\overline{\text{RESET}}$ pin is used, it must externally be held low long enough for the oscillator to stabilize, plus 4 states for the reset sequence.

When exiting Powerdown using an external interrupt, a positive level on the pin mapped to INT7 (either EXTINT pin or Port0.7 pin) will bring the part out of Powerdown mode. This procedure is not affected by either the interrupt disable bit or the interrupt mask register. An internal timing circuit is used to ensure that the oscillator has stabilized before the internal clocks are turned on. Figure 30 shows the power down and powerup sequence in such a case.

During normal operation, before the chip goes into powerdown, the V_{PP} pin will rise to V_{CC} through an internal pullup. The user must connect a capacitor between V_{PP} and V_{SS}. A positive level on the pin mapped to INT7 (external interrupt) will start discharging this capacitor if the chip was in Powerdown when this edge occurred. The internal current source used to discharge this capacitor is approximately 100 μA. A threshold detector will detect 1 V or lower on the V_{PP} pin and mark the end of the time-out period. A 1 μF capacitor will provide about 4 ms startup time.

If the external interrupt is used to bring the part out of Powerdown, that bit will be set in the interrupt pending register when the chip starts to run. If the interrupt is not masked off, the first section of code executed will be the interrupt service routine, otherwise execution will begin with the code following the IDLPD instruction. If the interrupt is not serviced the interrupt pending bit will remain set.

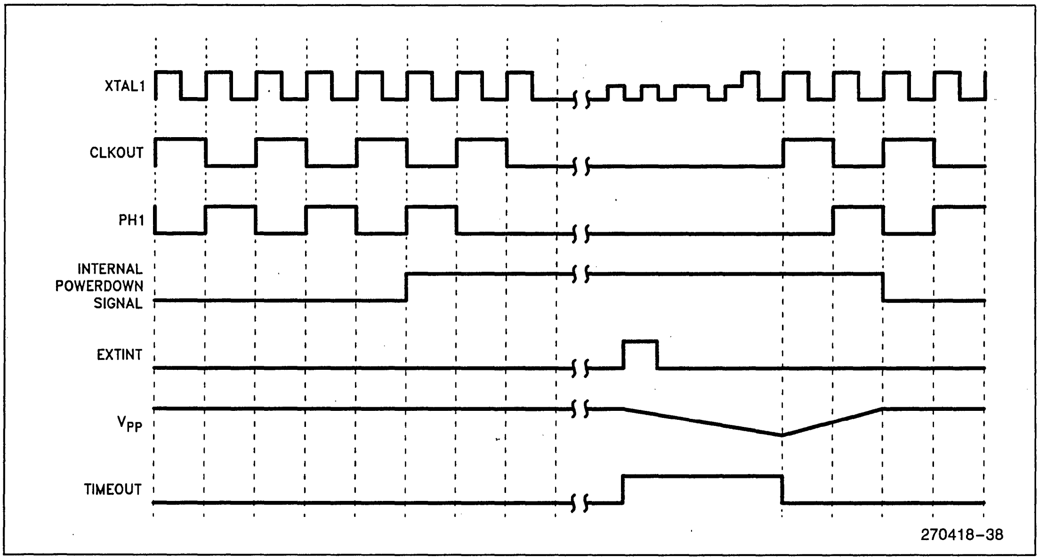


Figure 30. Powerdown/Up Sequence

4.3 RESET SEQUENCE AND STATUS

The reset sequence on the 80C196KA is slightly different than that of the 8096BH. Figure 31 shows the sequence used on the 80C196KA.

As soon as the RESET line is pulled low the I/O and control lines will go into their reset condition. The state of these lines is shown below:

Pin Name	Multiplexed Port Pins	Value of the Pin on Reset
$\overline{\text{RESET}}$		Mid-sized Pullup
$\overline{\text{ALE}}$		Weak Pullup
$\overline{\text{RD}}$		Weak Pullup
$\overline{\text{BHE}}$		Weak Pullup
$\overline{\text{WR}}$		Weak Pullup
INST		Weak Pull-up
$\overline{\text{EA}}$		Undefined Input *
READY		Undefined Input *
NMI		Undefined Input *
BUSWIDTH		Undefined Input *
CLKOUT		Phase 2 of Clock
System Bus	P3.0-P4.7	Weak Pullups

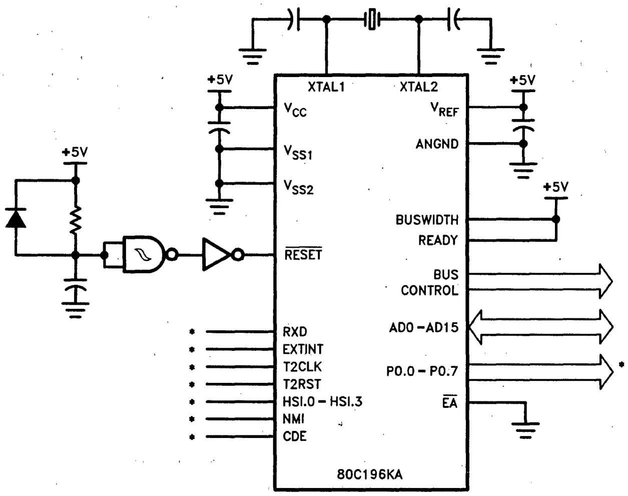
Pin Name	Multiplexed Port Pins	Value of the Pin on Reset
ACH0-7	P0.0-P0.7	Undefined Input *
PORT1	P1.0-P1.7	Weak Pullups
TXD	P2.0	Weak Pullup
RXD	P2.1	Undefined Input *
EXTINT	P2.2	Undefined Input *
T2CLK	P2.3	Undefined Input *
T2RST	P2.4	Undefined Input *
PWM	P2.5	Weak Pulldown
—	P2.6-P2.7	Weak Pullups
HSI0-HSI1		Undefined Input *
HSI2/HSO4		Undefined Input *
HSI3/HSO5		Undefined Input *
HSO0-HSO3		Weak Pulldown

The weak pullups and pulldowns are sufficient to hold a line in one position or another. Pins listed as undefined inputs (*) must be tied or driven externally, otherwise the part may not function properly. Reset must be held low for 4 state times.

In order for the part to function, the following pins must be connected:

V_{CC}, V_{SS1}, V_{SS2}, V_{REF}, ANGND, XTAL1, XTAL2

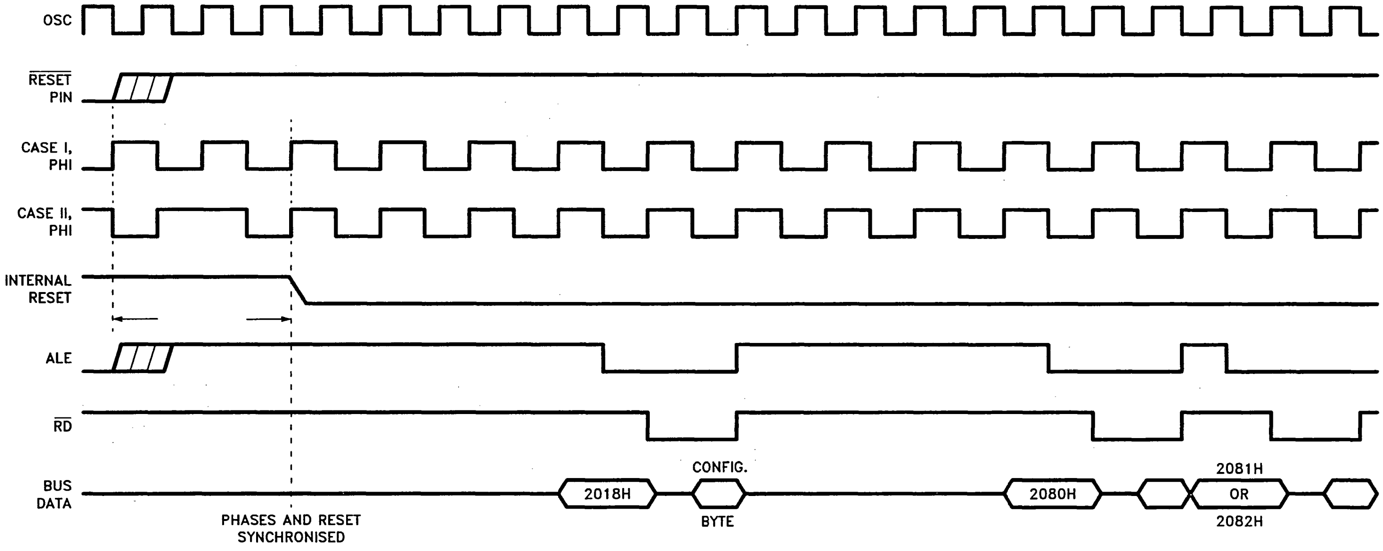
Minimum Connections for 16-Bit Bus Mode



* MUST BE DRIVEN HIGH OR LOW

270418-45

80C196KA Reset Sequence



270418-39

Figure 31. Reset Sequence

After the reset sequence, the internal registers are at the following values:

REGISTER NAME	VALUE
AD_RESULT	0000H
HSI_STATUS	x0x0x0x0B
SBUF(RX)	00H
INT_MASK	00000000B
INT_PENDING	00000000B
TIMER1	0000H
TIMER2	0000H
IOPORT1	11111111B
IOPORT2	11000001B
SP_STAT/SP_CON	00000000B
IMASK1	00000000B
IPEND1	00000000B
WSR	XXXX0000B
HSI_MODE	11111111B
IOC2	X0000000B
IOC0	000000X0B
IOC1	00100001B
PWM_CONTROL	00H
IOPORT3	11111111B
IOPORT4	11111111B
IOS0	00000000B
IOS1	00000000B
IOS2	00000000B

4.4 PROGRAM PROTECTION FEATURES

Software protection

Several features to assist in recovery from hardware and software errors are available on the 80C196KA. Protection is also provided against executing unimplemented opcodes by making use of the unimplemented opcode interrupt. In addition, the hardware reset instruction (RST) can be used in software to cause a reset if the program counter goes out of bounds. This instruction has an opcode of 0FFH, so if the processor reads in bus lines which have been pulled high it will reset itself.

Clock Failure Detect

A clock failure detection circuit is provided to recover from hardware problems. When triggered by too slow of a clock on XTAL1, it pulls the $\overline{\text{RESET}}$ line low. The switch frequency is V_{CC} dependent. At a V_{CC} of 6 volts, detection occurs at some point below 250KHz. When V_{CC} is at 4 volts, the detection point is below 28KHz. This feature can be disabled by holding the CDE pin (Clock Detect Enable) at a low level. It should be disabled when using the Powerdown mode. CDE uses the same pin as V_{PD} on the 8096 since the V_{PD} pin function is not needed for the 80C196KA.

Watchdog Timer

The Watchdog Timer can be enabled to cause a hardware reset every 64K state times unless the timer is cleared periodically. The timer is started by writing the sequence "1Eh", "E1h" to the Watchdog Register. Once started it can only be turned off by resetting the chip. To clear the watchdog the sequence "1Eh", "E1h" must be written to the register.

When any of the protection methods are used to reset the chip, the external $\overline{\text{RESET}}$ line will be pulled low by an internal pulldown transistor. It will keep pulling the line down until the part resets itself. Writing to the watchdog timer will not turn the transistor off. The $\overline{\text{RESET}}$ line can also be used as an output to reset other circuitry. If a capacitor is used on the $\overline{\text{RESET}}$ line for reset timing, the line may never be pulled below 0.8 volts. This could cause other external circuitry not to be reset.

4.5 ONCE™ AND TEST MODES

Test modes are entered on the 80C196KA by externally holding ALE, INST or $\overline{\text{RD}}$ in their active state while the RESET pin is taken high. By using combinations of pins different test modes can be selected. The only test mode which is not reserved for Intel use is ONCE. This mode is entered by driving ALE high and $\overline{\text{RD}}$ and INST low while RESET is taken high.

ONCE is the ON-Circuit-Emulation mode. In this mode all of the pins, except XTAL1 and XTAL2, are floated. Some of the pins are not truly high impedance as they have weak pullups or pulldowns.

5.0 DIFFERENCES BETWEEN THE 80C196KA AND THE 8096BH

5.1 CONVERTING FROM OTHER MCS®-96 PRODUCTS TO THE 80C196KA

The following list of suggestions for designing an 8X9XBH system will yield a design that is easily converted to the 80C196KA.

1. Do not base critical timing loops on instruction or peripheral execution times.
2. Use equate statements to set all timing parameters, including the baud rate.
3. Do not base hardware timings on CLKOUT or XTAL1. The timings of the 80C196KA are different than those of the 8X9XBH, but they will function with standard ROM / EPROM / Peripheral type memory systems.
4. Make sure all inputs are tied high or low and not left floating.
5. On the 8X9XBH, the $\overline{\text{WRL}}/\overline{\text{WR}}$ and $\overline{\text{WRH}}/\overline{\text{BHE}}$ signals both go low for byte writes to odd addresses in eight bit write strobe mode. On the 80C196KA, only the $\overline{\text{WRH}}/\overline{\text{BHE}}$ signal goes low for this type of operation.
6. Indexed and indirect operations relative to the stack pointer (SP) work differently on the 80C196KA than on the 8096. On the 8096, the address is calculated based on the un-updated version of the stack pointer. The 80C196KA uses the updated version. The offset for PUSH[SP], POP[SP], PUSH nn[SP] and POP nn[SP] instructions may need to be changed by a count of 2.

5.2 NEW FEATURE SUMMARY

CPU FEATURES

Divide by 2 instead of divide by 3 clock for 1.5X performance

Faster instructions, especially indexed/indirect data operations

2.33 µSec 16×16 multiply with 12MHz clock (was 6.25 µSec)

Faster interrupt response (almost twice as fast)

Different Reset Sequence

Powerdown and Idle Modes

Clock Failure Detect

6 new instructions including Compare Long and Block Move

8 new interrupt vectors

PERIPHERAL FEATURES

SFR Window switching allows read-only registers to be written and vice-versa

Timer2 can count up and down by external selection

Timer2 has an independent capture register

HSO lines which transitioned are saved

HSO lines can be written directly

HSO has CAM Lock and CAM Clear commands

A to D has a selectable sample and hold and speed control

New Baud Rate values are needed for serial port, higher speeds possible in all modes

Double buffered serial port transmit register

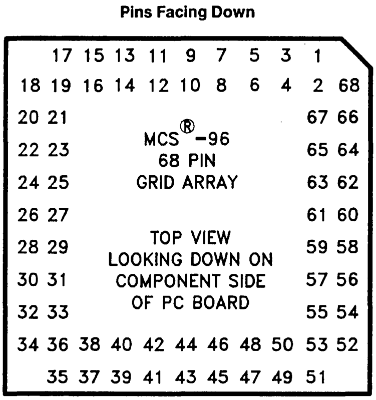
Serial Port Receive Overrun and Framing Error Detection

PWM has a Divide-by-2 Prescaler

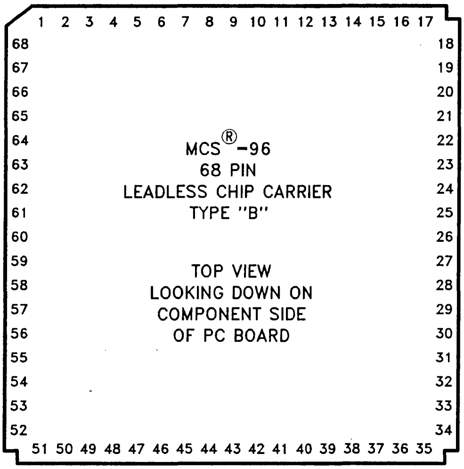
6.0 PACKAGES, PINOUTS, PIN DEFINITIONS

PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description
1	9	ACH7/P0.7	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{EA}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	V_{CC}	32	46	AD14/P4.6	55	23	P1.4
10	68	V_{SS}	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	BUSWIDTH	37	41	$\overline{BHE}/\overline{WRH}$	60	18	TXD/P2.0
15	63	INST	38	40	$\overline{WR}/\overline{WRL}$	61	17	RXD/P2.1/PALE
16	62	ALE/$\overline{ADV}$	39	39	PWM/P2.5	62	16	$\overline{RESET}$
17	61	$\overline{RD}$	40	38	P2.7/T2CAPTURE	63	15	EXTINT/P2.2
18	60	AD0/P3.0	41	37	V_{PP}	64	14	CDE
19	59	AD1/P3.1	42	36	V_{SS}	65	13	V_{REF}
20	58	AD2/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4
22	56	AD4/P3.4	45	33	P2.6/T2UP-DN	68	10	ACH5/P0.5
23	55	AD5/P3.5	46	32	P1.7			

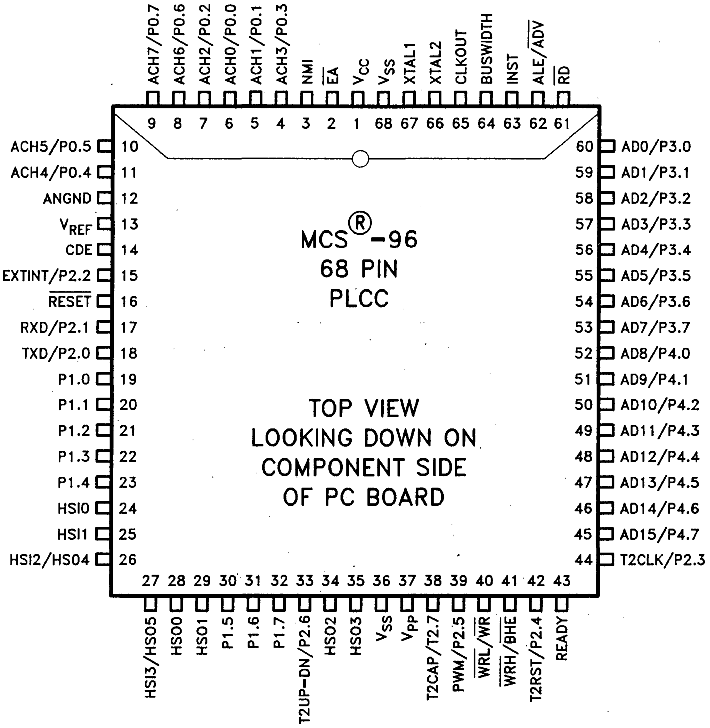
PGA packages will be available on future parts.



270418–40



270418–44



270418–41

PIN DESCRIPTIONS

Symbol	Name and Function
V_{CC}	Main supply voltage (5V).
V_{SS}	Digital circuit ground (0V). There are two V_{SS} pins, both of which must be connected.
CDE	Clock Detect Enable - When pulled high enables the clock failure detection circuit. If the XTAL1 frequency falls below a specified limit the $\overline{\text{RESET}}$ pin will be pulled low. This pin was the V_{PD} pin on the 8096.
V_{REF}	Reference voltage for the A/D converter (5V). V_{REF} is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0. Must be connected for A/D and Port 0 to function.
ANGND	Reference ground for the A/D converter. Must be held at nominally the same potential as V_{SS}.
V_{PP}	Timing pin for the return from powerdown circuit. Connect this pin with a 1 μF capacitor to V_{SS} and a 1 mΩ resistor to V_{CC}. If this function is not used V_{PP} may be tied to V_{CC}. This pin was V_{BB} on the 8X9X-90 parts and will be programming voltage on future EPROM parts.
XTAL1	Input of the oscillator inverter and of the internal clock generator.
XTAL2	Output of the oscillator inverter.
CLKOUT	Output of the internal clock generator. The frequency of CLKOUT is ½ the oscillator frequency. It has a 50% duty cycle.
$\overline{\text{RESET}}$	Reset input to the chip. Input low for at least 4 state times to reset the chip. The subsequent low-to-high transition re- synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to locations 2080H is executed. Input high for normal operation. $\overline{\text{RESET}}$ has an internal pullup.
BUSWIDTH	Input for buswidth selection. If CCR bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is a 1, a 16-bit bus cycle occurs. If BUSWIDTH is a 0 an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. This pin is the $\overline{\text{TEST}}$ pin on 8X9X-90 parts. Systems with $\overline{\text{TEST}}$ tied to V_{CC} do not need to change.
NMI	A positive transition causes a vector through 203EH.
INST	Output high during an external memory read indicates the read is an instruction fetch. INST is valid throughout the bus cycle. INST is activated only during external memory accesses.
$\overline{\text{EA}}$	Input for memory select (External Access). $\overline{\text{EA}}$ equal to a TTL-high causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM/EPROM. $\overline{\text{EA}}$ equal to a TTL-low causes accesses to these locations to be directed to off-chip memory.
ALE/$\overline{\text{ADV}}$	Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is $\overline{\text{ADV}}$, it goes inactive high at the end of the bus cycle. $\overline{\text{ADV}}$ can be used as a chip select for external memory. ALE/$\overline{\text{ADV}}$ is activated only during external memory accesses.
$\overline{\text{RD}}$	Read signal output to external memory. $\overline{\text{RD}}$ is activated only during external memory reads.
$\overline{\text{WR}}$/$\overline{\text{WRL}}$	Write and Write Low output to external memory, as selected by the CCR. $\overline{\text{WR}}$ will go low for every external write, while $\overline{\text{WRL}}$ will go low only for external writes where an even byte is being written. $\overline{\text{WR}}$/$\overline{\text{WRL}}$ is activated only during external memory writes.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
$\overline{\text{BHE}}$/$\overline{\text{WRH}}$	Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{\text{BHE}}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{\text{BHE}}$ = 1), to the high byte only (A0 = 1, $\overline{\text{BHE}}$ = 0), or both bytes (A0 = 0, $\overline{\text{BHE}}$ = 0). If the $\overline{\text{WRH}}$ function is selected, the pin will go low if the bus cycle is writing to an odd memory location. $\overline{\text{BHE}}$/$\overline{\text{WRH}}$ is valid only during 16-bit external memory write cycles.
READY	Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the memory controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle held not ready is available through configuration of CCR.
HSI	Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as inputs by future EPROM parts in Programming Mode.
HSO	Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.
Port 0	8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins are also a mode input to future EPROM parts in the Programming Mode.
Port 1	8-bit quasi-bidirectional I/O port.
Port 2	8-bit multi-functional port. All of its pins are shared with other functions in the 80C196KA.
Ports 3 and 4	8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups. Available only on future ROM and EPROM parts.

MCS®-96 Data Sheets, Application Notes, Development Support Tools and Index

ADVANCE INFORMATION

MCS®-96 809XBH, 839XBH, 879XBH ADVANCED 16-BIT MICROCONTROLLER WITH 8- OR 16-BIT EXTERNAL BUS

- 879XBH: an 809XBH with 8K Bytes of On-Chip EPROM
- 839XBH: an 809XBH with 8K Bytes of On-Chip ROM

- 232 Byte Register File
- Register-to-Register Architecture
- 10-Bit A/D Converter with S/H
- Five 8-Bit I/O Ports
- 20 Interrupt Sources
- Pulse-Width Modulated Output
- ROM/EPROM Lock
- Run-Time Programmable EPROM

- High Speed I/O Subsystem
- Full Duplex Serial Port
- Dedicated Baud Rate Generator
- 6.25 μs 16 x 16 Multiply
- 6.25 μs 32/16 Divide
- 16-Bit Watchdog Timer
- Four 16-Bit Software Timers
- Two 16-Bit Counter/Timers

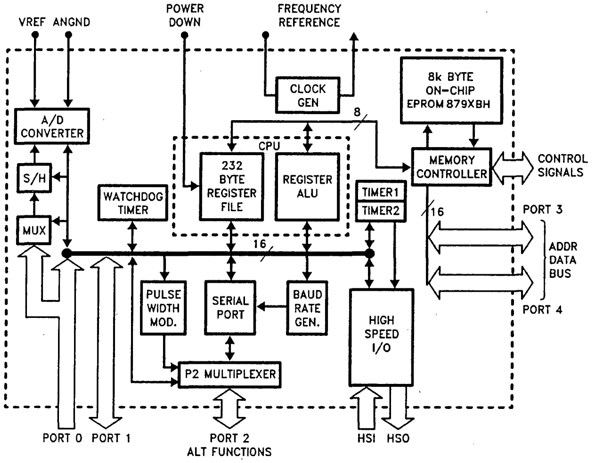
The MCS®-96 family of 16-bit microcontrollers consists of many members, all of which are designed for high-speed control functions. The MCS-96 family members produced using Intel's HMOS-III process are described in this data sheet.

The CPU supports bit, byte, and word operations. Thirty-two bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096BH can do a 16-bit addition in 1.0 μs and a 16 x 16-bit multiply or 32/16 divide in 6.25 μs. Instruction execution times average 1 to 2 μs in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at preset times. The high-speed output unit can simultaneously perform software timer functions. Up to four 16-bit software timers can be in operation at once.

The on-chip A/D converter includes a Sample and Hold, and converts up to 8 multiplexed analog input channels to 10-bit digital values. With a 12 MHz crystal, each conversion takes 22 μs. This feature is only available on the 8X95BHs and 8X97BHs, with the 8X95BHs having 4 multiplexed analog inputs.

Also provided on-chip are a serial port, a Watchdog Timer, and a pulse-width modulated output signal.



270090-50

Figure 1. MCS®-96 Block Diagram

FUNCTIONAL OVERVIEW

The following section is an overview of the 8X9XBH devices, generally referred to as the 8096BH. Additional information is available in the Embedded Controller Handbook, order number 210918.

CPU Architecture

The 8096BH uses the same address space for both program and data memory, except in the address range from 00H through 0FFH. Data fetches in this range are always to the Register File, while instruction fetches from these locations are directed to external memory. (Locations 00H through 0FFH in external memory are reserved for Intel development systems).

Within the Register File, locations 00H through 17H are register mapped I/O control registers, also referred to as Special Function Registers (SFRs). The rest of the Register File (018H through 0FFH) contains 232 bytes of RAM, which can be referenced as bytes, words, or double-words. This register space allows the user to keep the most frequently-used variables in on-chip RAM, which can be accessed faster than external memory. Locations 0F0H through 0FFH can be preserved during power down via a separate power down pin (V_{PD}).

Outside of the Register File, program memory, data memory, and peripherals can be intermixed. The addresses with special significance are:

0000H-	0017H	Register Mapped I/O (SFRs)
0018H-	0019H	Stack Pointer
1FFEH-	1FFFH	Ports 3 and 4
2000H-	2011H	Interrupt Vectors
2012H-	2017H	Reserved
2018H		Chip Configuration Byte
2019H		Reserved
201AH-	201BH	"Jump to Self" Opcode (27 FE)
201CH-	201FH	Reserved
2020H-	202FH	Security Key
2030H-	207FH	Reserved
2080H		Reset Location

The 839XBH carries 8K bytes of ROM, while the 879XBH has 8K bytes of EPROM. With ROM and EPROM parts, the internal program memory occupies addresses 2000H through 3FFFH. Instruction or data fetches from these addresses access the on-chip memory if the $\overline{EA}$ pin is externally held at 5V. If the $\overline{EA}$ pin is at 0V, these addresses access off-chip memory. On the 879XBH parts, holding $\overline{EA}$ at +12.75V puts the part in Programming Mode, which is described in the EPROM Characteristics Section of this data sheet.

A memory map for the MCS-96 product family is shown in Figure 2.

The RALU (Register/ALU) section consists of a 17-bit ALU, the Program Status Word, the Program Counter, and several temporary registers. A key feature of the 8096BH is that it does not use an accumulator. Rather, it operates directly on any register in the Register File. Being able to operate directly on data in the Register File without having to move it into and out of an accumulator results in a significant improvement in execution speed.

In addition to the normal arithmetic and logical functions, the MCS-96 instruction set provides the following special features:

6.25 μs Multiply and Divide
Multiple Shift Instruction
3 Operand Instructions
Normalize Instruction
Software Reset Instruction

All operations on the 8096BH take place in a set number of "State Times." The 8096BH uses a three phase internal clock, so each state time is 3 oscillator periods. With a 12 MHz clock, each state time requires 0.25 μs, based on a T_{OSC} of 83 ns.

Operating Modes

The 8096BH supports a variety of options to simplify memory systems, interfacing requirements and ready control. Bus flexibility is provided by allowing selection of bus control signal definitions and runtime selection of the external bus width. In addition, several ready control modes are available to simplify the external hardware requirements for accessing slow devices. The Chip Configuration Register is used to store the operating mode information.

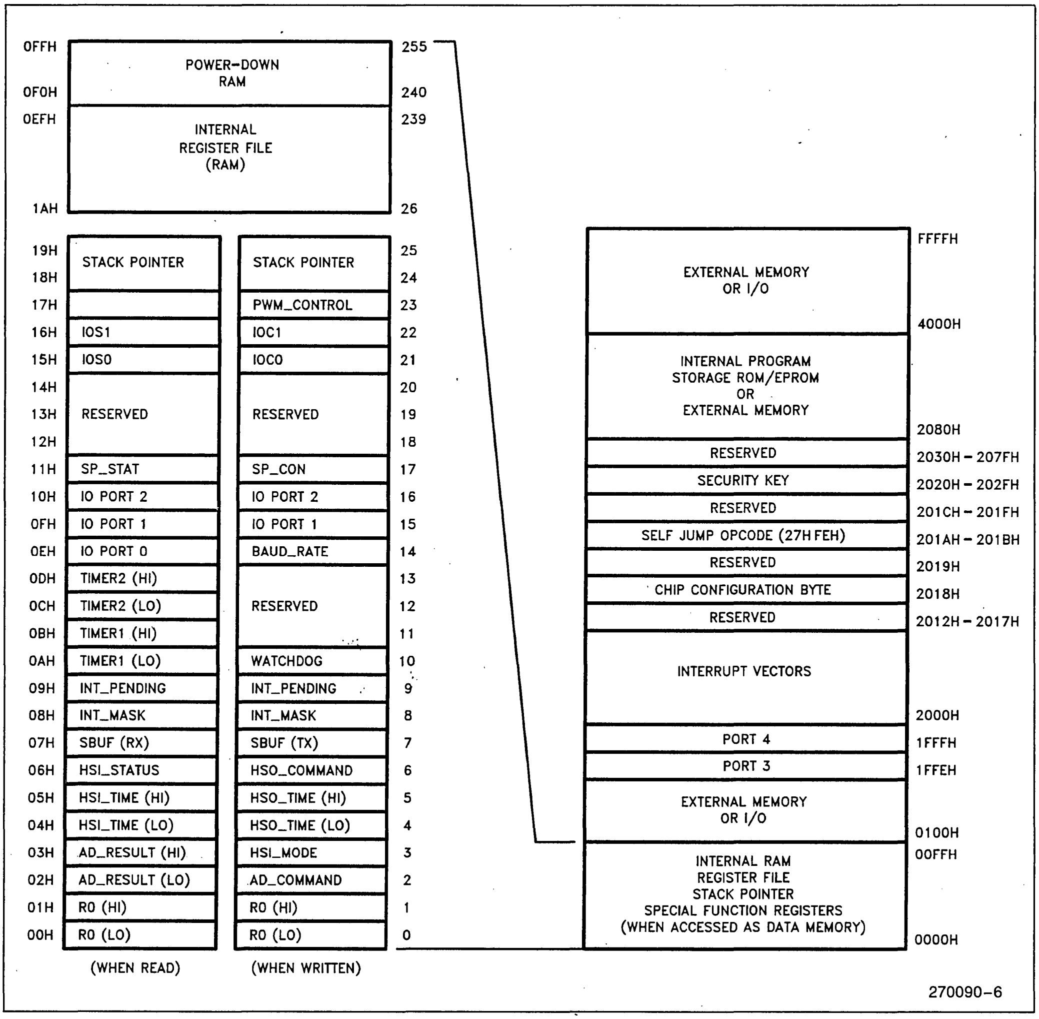


Figure 2. Memory Map

CHIP CONFIGURATION REGISTER (CCR)

Configuration information is stored in the Chip Configuration Register (CCR). Four of the bits in the register specify the bus control mode and ready control mode. Two bits also govern the level of ROM/EPROM protection and one bit is NANDed with the BUSWIDTH pin every bus cycle to determine the bus size. The CCR bit map is shown in Figure 3, and the functions associated with each bit are described later.

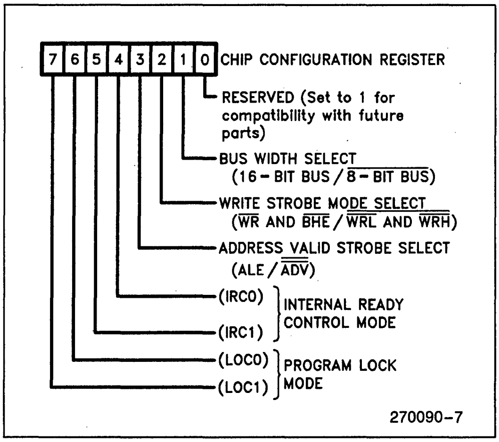

7 6 5 4 3 2 1 0 CHIP CONFIGURATION REGISTER
RESERVED (Set to 1 for compatibility with future parts)
BUS WIDTH SELECT (16-BIT BUS / $\overline{\text{8-BIT BUS}}$)
WRITE STROBE MODE SELECT ($\overline{\text{WR}}$ AND $\overline{\text{BHE}}$ / $\overline{\text{WRL}}$ AND $\overline{\text{WRH}}$)
ADDRESS VALID STROBE SELECT (ALE / $\overline{\text{ADV}}$)
(IRC0) (IRC1) INTERNAL READY CONTROL MODE
(LOC0) (LOC1) PROGRAM LOCK MODE
270090-7

Figure 3. Chip Configuration Register

The CCR is loaded on reset with the Chip Configuration Byte, located at address 2018H. The CCR register is a non-memory mapped location that can only be written to during the reset sequence; once it is loaded it cannot be changed until the next reset occurs. The 8096BH will correctly read this location in every bus mode.

In order to work properly with an 8-bit only system, it is necessary to hold the upper address byte on the address bus throughout the CCB read cycle since an address latch may not be present. However, in a 16-bit system, the 8X9XBH must float the high half of the bus to avoid contention with the high data byte during the CCB read. To accomplish a correct read on either 8- or 16-bit buses, the upper address lines are current sensed (during CCB read only) and will be floated if a current of approximately 1 mA or more is detected, indicating a bus contention.

If the $\overline{\text{EA}}$ pin is set to a logical 0, the access to 2018H comes from external memory. If $\overline{\text{EA}}$ is a logical 1, the access comes from internal ROM/EPROM. If $\overline{\text{EA}}$ is +12.5V, the CCR is loaded with a byte from a separate non-memory-mapped location called PCCB (Programming CCB). The Programming Mode is described in the EPROM Characteristics Section.

BUS WIDTH

The 8096BH external bus width can be run-time configured to operate as a standard 16-bit multiplexed address/data bus, or as an 8088 minimum mode type 16-bit address/ 8-bit data bus.

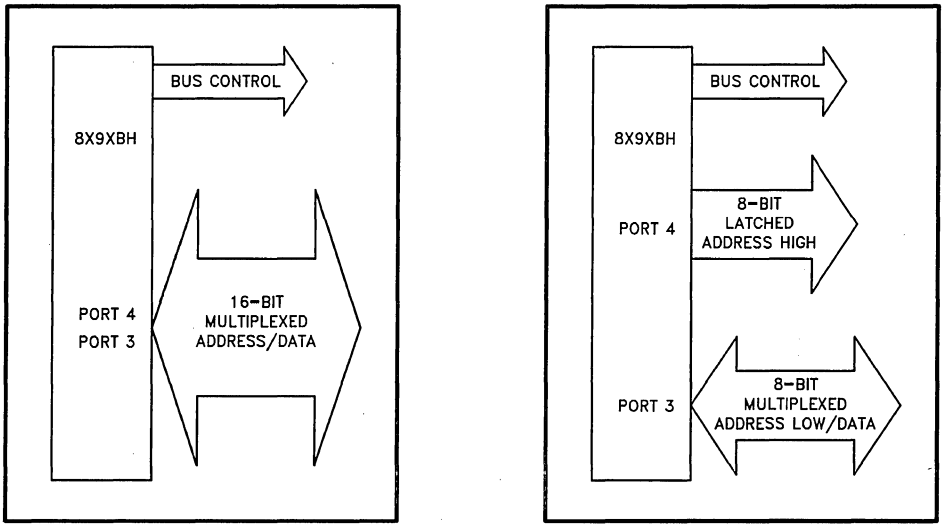
During 16-bit bus cycles, Ports 3 and 4 contain the address multiplexed with data using ALE to latch the address. In 8-bit bus cycles, Port 3 is multiplexed address/data while Port 4 is address bits 8 through 15. The address bits on Port 4 are valid throughout an 8-bit bus cycle. Figure 4 shows the two options.

The bus width can be changed each bus cycle and is controlled using bit 1 of the CCR with the BUSWIDTH pin. If either CCR.1 or BUSWIDTH is a 0, external accesses will be over a 16-bit address/8-bit data bus. If both CCR.1 and BUSWIDTH are 1s, external accesses will be over a 16-bit address/16-bit data bus. Internal accesses are always 16-bits wide.

The bus width can be changed every external bus cycle if a 1 was loaded into CCR bit 1 at reset. If this is the case, changing the value of the BUSWIDTH pin at run-time will dynamically select the bus width. For example, the user could feed the INST line into the BUSWIDTH pin, thus causing instruction accesses to be word wide from EPROMs while data accesses are byte wide to and from RAMs. A second example would be to place an inverted version of address bit 15 on the BUSWIDTH pin. This would make half of external memory word wide, while half is byte wide.

Since BUSWIDTH is sampled after address decoding has had time to occur, even more complex memory maps could be constructed. See the timing specifications for an exact description of BUSWIDTH timings. The bus width will be determined by bit 1 of the CCR alone on 48-pin parts since they do not have a BUSWIDTH pin.

When using an 8-bit bus, some performance degradation is to be expected. On the 8096BH, instruction execution times with an 8-bit bus will slow down if any of three conditions occur. First, word writes to external memory will cause the executing instruction to take two extra state times to complete. Second, word reads from external memory will cause a one state time extension of instruction execution time. Finally, if the prefetch queue is empty when an instruction fetch is requested, instruction execution is lengthened by one state time for each byte that must be externally acquired (worst case is the number of bytes in the instruction minus one).



270090-8

16-Bit Bus

270090-9

8-Bit Bus

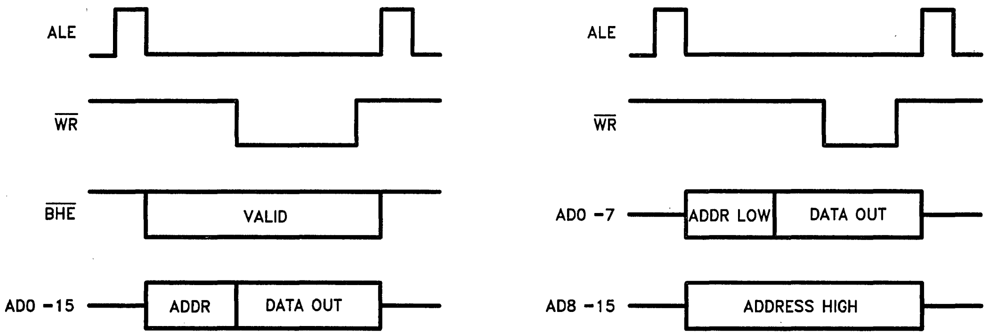
Figure 4. Bus Width Options

BUS CONTROL

The 8096BH can be made to provide bus control signals of several types. Three control lines have dual functions designed to reduce external hardware. Bits 2 and 3 of the CCR specify the functions performed by these control lines.

Standard Bus Control

If CCR bits 2 and 3 are 1s, then the standard 8096BH control signals $\overline{\text{WR}}$, $\overline{\text{BHE}}$ and ALE are provided (Figure 5). $\overline{\text{WR}}$ will come out for every write. $\overline{\text{BHE}}$ will be valid throughout the bus cycle and can be combined with $\overline{\text{WR}}$ and address line 0 to form $\overline{\text{WRL}}$ and $\overline{\text{WRH}}$. ALE will rise as the address starts to come out, and will fall to provide the signal to externally latch the address.



270090-10

16-Bit Bus Cycle

270090-11

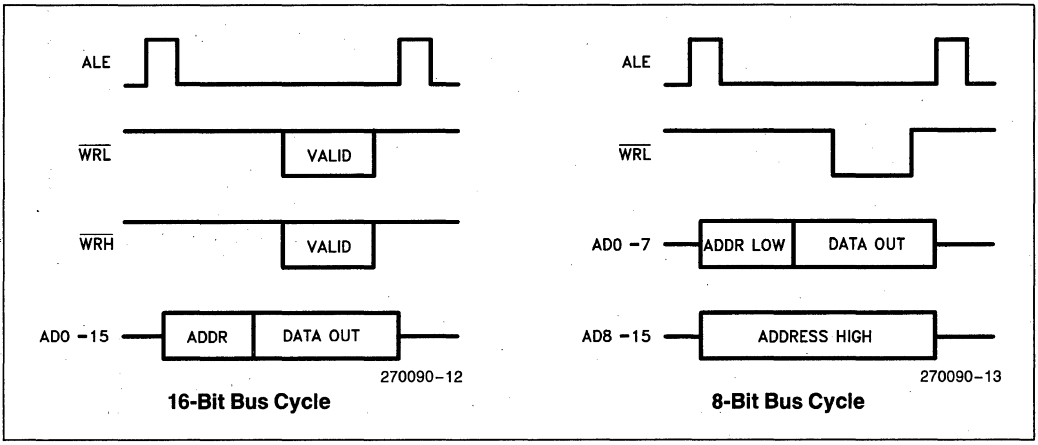
8-Bit Bus Cycle

Figure 5. Standard Bus Control

Write Strobe Mode

The Write Strobe Mode eliminates the necessity to externally decode for odd or even byte writes. If CCR bit 2 is a 0, and the bus is in a 16-bit cycle, $\overline{\text{WRL}}$ and $\overline{\text{WRH}}$ signals are provided in place of $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ (Figure 6). $\overline{\text{WRL}}$ will go low for all byte writes to an even address and all word writes. $\overline{\text{WRH}}$ will go low for all byte writes to an odd address and all word writes.

In an 8-bit bus cycle $\overline{\text{WRL}}$ will go active for all writes.



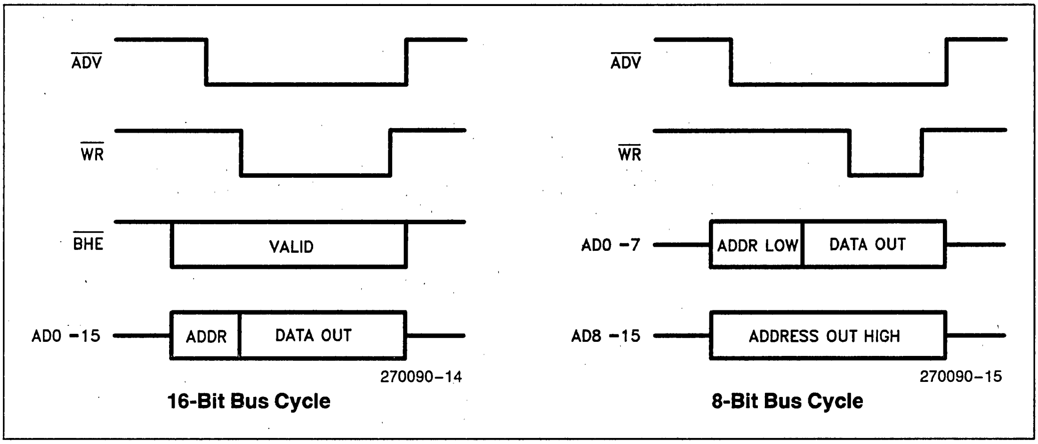
16-Bit Bus Cycle

8-Bit Bus Cycle

Figure 6. Write Strobe Mode

Address Valid Strobe Mode

If CCR bit 3 is a 0, then an Address Valid Strobe is provided in the place of ALE (Figure 7). When the Address Valid Mode is selected, $\overline{\text{ADV}}$ will go low after an external address is set up. It will stay low until the end of the bus cycle, where it will go inactive high. This can be used to provide a chip select for external memory.



270090-14

16-Bit Bus Cycle

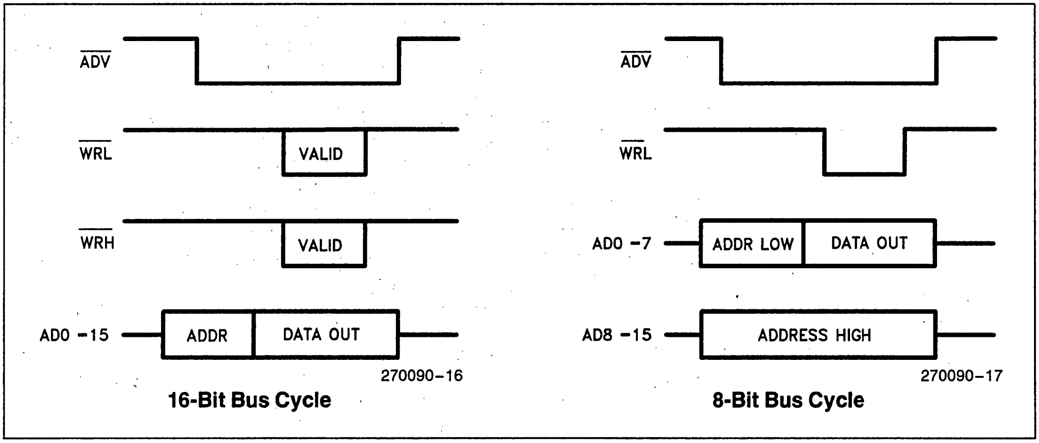
270090-15

8-Bit Bus Cycle

Figure 7. Address Valid Strobe Mode

Address Valid with Write Strobe

If both CCR bits 2 and 3 are 0s, both the Address Valid Strobe and the Write Strobes will be provided for bus control. Figure 8 shows these signals.



270090-16

16-Bit Bus Cycle

270090-17

8-Bit Bus Cycle

Figure 8. Write Strobe with Address Valid Strobe

READY CONTROL

To simplify ready control, four modes of internal ready control logic have been provided. The modes are chosen by properly configuring bits 4 and 5 of the CCR.

The internal ready control logic can be used to limit the number of wait states that slow devices can insert into the bus cycle. When the READY pin is pulled low, wait states will be inserted into the bus cycle until the READY pin goes high, or the number of wait states equals the number specified by CCR bits 4 and 5, whichever comes first. Table 1 shows the number of wait states that can be selected. Internal ready control can be disabled by loading 11 into bits 4 and 5 of the CCR.

Table 1. Internal Ready Control

IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

This feature provides for simple ready control. For example, every slow memory chip select line could be ORed together and be connected to the READY pin with CCR bits 4 and 5 programmed to give the proper number of wait states to the slow devices.

ROM/EPROM LOCK

Four modes of program memory lock are available on the 839XBH and 879XBH parts. CCR bits 6 and 7 (LOC0, LOC1) select whether internal program memory can be read (or written in EPROM parts) by a program executing from external memory. The modes are shown in Table 2. Internal ROM/EPROM addresses 2020H through 3FFFH are protected from reads while 2000H through 3FFFH are protected from writes, as set by the CCR.

Table 2. Program Lock Modes

LOC1	LOC0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

Only code executing from internal memory can read protected internal memory, while a write protected memory can not be written to, even from internal execution. As a result of 8096BH prefetching of instructions, however, accesses to protected memory are not allowed for instructions located above 3FFAH. Note that the interrupt vectors and the CCR are not protected.

To provide ROM/EPROM lock while allowing verification and testing, the 839XBH and 879XBH require security key verification before programming or test modes are allowed to read protected memory. More information on ROM/EPROM Lock can be found in the EPROM Characteristics section.

High Speed I/O Unit (HSIO)

The HSIO unit consists of the High Speed Input Unit (HSI), the High Speed Output Unit (HSO), one counter and one timer. "High Speed" denotes that the units can perform functions related to the timers without CPU intervention. The HSI records times when events occur and the HSO triggers events at pre-programmed times.

All actions within the HSIO unit are synchronized to the timers. The two 16-bit timer/counter registers in the HSIO unit are cleared on chip reset and can be programmed to generate an interrupt on overflow. The Timer 1 register is automatically incremented every 8 state times (every 2.0 μs, with a 12 MHz clock). The Timer 2 register can be programmed to count transitions on either the T2CLK pin or HSI.1 pin. It is incremented on both positive and negative edges of the selected input line. In addition to being cleared by reset, Timer 2 can also be cleared in software or by signals from input pins T2RST or HSI.0. Neither of these timers is required for either the Watchdog Timer or the serial port.

The High Speed Input (HSI) unit can detect transitions on any of its 4 input lines. When one occurs it records the time (from Timer 1) and which input lines made the transition. This information is recorded with 2 μs (12 MHz system) resolution and stored in an 8-level FIFO. The unit can be programmed to look for four types of events, as shown in Figure 9. It can activate the HSI Data Available interrupt either when the Holding Register is loaded or the 6th FIFO entry has been made. Each input line can be individually enabled or disabled to the HSI unit by software.

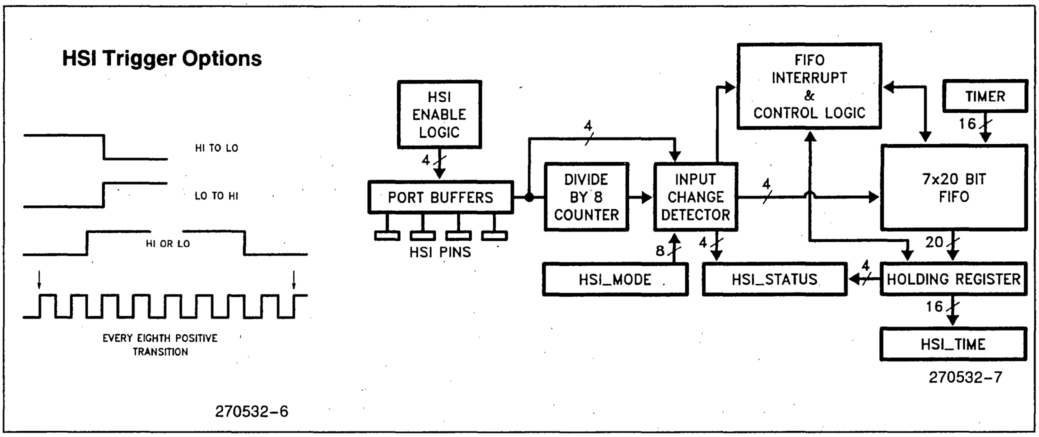


Figure 9. High Speed Input Unit

The High Speed Output (HSO) unit is shown in Figure 10. It can be programmed to set or clear any of its 6 output lines, reset Timer 2, trigger an A/D conversion, or set one of 4 Software Timer flags at a programmed time. An interrupt can be enabled for any of these events. Either Timer 1 or Timer 2 can be referenced for the programmed time value and up to 8 commands for preset actions can be stored in the CAM (Content Addressable Memory) file at any one time. As each action is carried out at its preset time that command is removed from the CAM making space for another command. HSO.4 and HSO.5 are shared with the HSI unit as HSI.2 and HSI.3, and can be individually enabled or disabled as outputs.

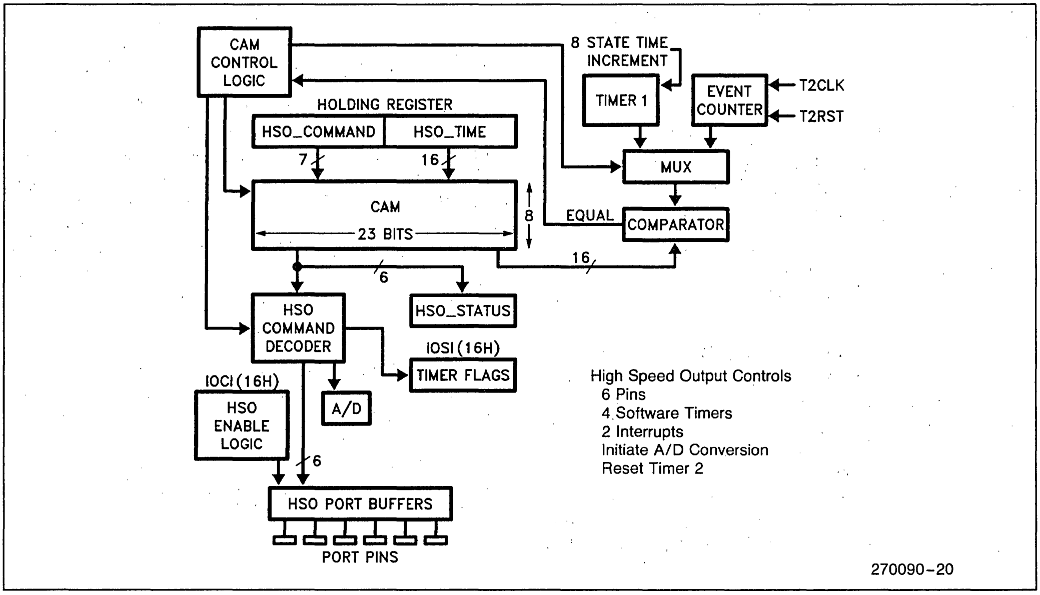


Figure 10. High Speed Output Unit

Standard I/O Ports

There are 5 8-bit I/O ports on the 8096BH in addition to the High Speed I/O lines.

Port 0 is an input-only port which shares its pins with the analog inputs to the A/D converter. The port can be read digitally and/or, by writing to the A/D Command Register, one of the lines can be selected as the input to the A/D converter. Port 0 is also used to input mode information on EPROM parts operating in the Programming Mode.

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). This configuration allows the pin to be used as either an input or an output without using a data direction register. In parallel with the weak internal pullup is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time.

Port 2 is a multi-functional port. Two of the pins (P2.6, 2.7) are quasi-bidirectional while the remaining six are shared with other functions in the 8096BH, as shown in Table 3. Port 2 is also used for control signals by EPROM parts operating in the Programming Mode.

Table 3. Port 2 Pin Functions

Port	Function	Alternate Function
P2.0	Output	TXD (Serial Port Transmit)
P2.1	Input	RXD (Serial Port Receive)
P2.2	Input	EXTINT (External Interrupt)
P2.3	Input	T2CLK (Timer 2 Clock)
P2.4	Input	T2RST (Timer 2 Reset)
P2.5	Output	PWM (Pulse Width Modulation)

Ports 3 and 4 are bi-directional I/O ports with open drain outputs. These pins are also used as the multiplexed address/data bus when accessing external memory, in which case they have strong internal pullups. The internal pullups are only used during external memory read or write cycles when the pins are outputting address or data bits. At any other time, the internal pullups are disabled. When used as a system bus, Ports 3 and 4 can be configured to be either a multiplexed 16-bit address/data bus or a multiplexed 16-bit address/ 8-bit data bus. EPROM parts also use Ports 3 and 4 to pass programming commands, addresses, data and status.

Serial Port

The serial port is compatible with the MCS-51 family, (8051, 8031 etc.), serial port. It is full duplex, and double-buffered on receive. There are 3 asynchronous modes and 1 synchronous mode of operation for the serial port. The asynchronous modes allow for 8 or 9 bits of data with even parity optionally inserted for one of the data bits. Selective interrupts based on the 9th data bit are available to support interprocessor communication.

Baud rates in all modes are determined by an independent 16-bit on-chip baud rate generator. Either the XTAL1 pin or the T2CLK pin can be used as the input to the baud rate generator. The maximum baud rate in the asynchronous mode is 187.5 KBaud. The maximum baud rate in the synchronous mode is 1.5 MBaud.

Pulse Width Modulator (PWM)

The PWM output shares a pin with port bit P2.5. When the PWM output is selected, this pin outputs a pulse train having a fixed period of 256 state times, and a programmable width of 0 to 255 state times. The width is programmed by loading the desired value, in state times, to the PWM Control Register.

A/D Converter with Sample and Hold

The analog-to-digital converter is a 10-bit, successive approximation converter with internal sample and hold. It has a fixed conversion time of 88 state times which includes the 4 state acquisition time of the internal Sample/Hold. With a 12 MHz clock, the conversion takes 22 μs, including the 1 μs sample for the Sample and Hold. The Sample acquisition begins 4 state times after the conversion is triggered. A 2 pF capacitance is charged from the input signal during acquisition.

The analog input must be in the range of 0 to V_{REF} (nominally, $V_{REF} = 5V$). This input can be selected from 8 analog input lines, which connect to the same pins as Port 0. A conversion can be initiated either by setting a control bit in the A/D Command register, or by programming the HSO unit to trigger the conversion at some specified time.

Interrupts

The 8096BH has 20 interrupt sources which vector through 8 interrupt vectors. A 0-to-1 transition from any of the sources sets a corresponding bit in the Interrupt Pending register. The content of the Interrupt Mask register determines if a pending interrupt will be serviced or not. If it is to be serviced, the CPU pushes the current Program Counter onto the stack and reloads it with the vector corresponding to the desired interrupt. The interrupt vectors are located in addresses 2000H through 2011H, as shown in Figure 11.

Vector	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 11. Interrupt Vectors

At the end of the interrupt routine the RET instruction pops the program counter from the stack and execution continues where it left off. It is not necessary to store and replace registers during interrupt routines as each routine can be set up to use a different section of the Register File. This feature of the architecture provides for very fast context switching. While the 8096BH has a single priority level in the sense that any interrupt may itself be interrupted, a priority structure exists for resolving simultaneously pending interrupts, as indicated in Figure 11. Since the interrupt pending and interrupt mask registers can be manipulated in software, it is possible to dynamically alter the interrupt priorities to suit the users software.

Watchdog Timer

The Watchdog Timer is a 16-bit counter which, once started, is incremented every state time. If not cleared before it overflows, the $\overline{\text{RESET}}$ pin will be pulled down for two state times, causing the system to be reinitialized. In a 12 MHz system, the Watchdog Timer overflows after 16 ms.

This feature is provided as a means of graceful recovery from a software upset. The counter must be cleared by the software before it overflows, or else the system assumes an upset has occurred and activates $\overline{\text{RESET}}$. Once the Watchdog Timer is started it cannot be turned off by software. The flip-flop which enables the Watchdog Timer has been designed to maintain its state through V_{CC} glitches to as low as 0V or as high as 7V for 1 μs to 1 ms.

To start the Watchdog Timer, or to clear it, one writes 1EH followed by 0E1H to the WDT address (000AH). The Watchdog cannot be stopped once it is started unless the system is reset.

PACKAGING

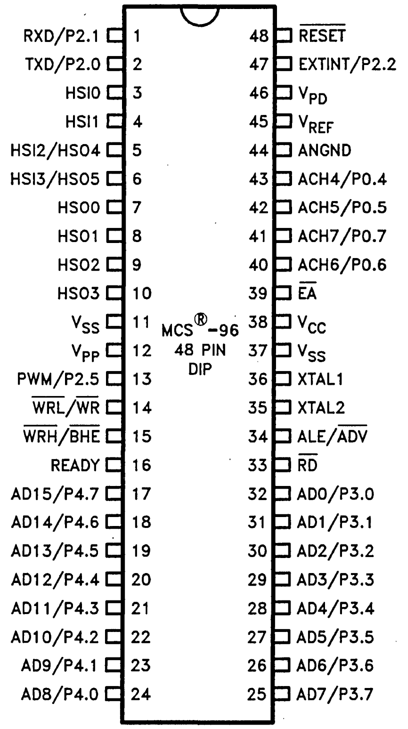
The 8096BH is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM or EPROM. The MCS-96 numbering system is shown in Figure 12. Figures 13–17 show the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin versions come in a Plastic Leaded Chip Carrier (PLCC), a Pin Grid Array (PGA) or a Type "B" Leadless Chip Carrier.

		Without A/D	With A/D
ROMless	48 Pin		C8095CH - Ceramic DIP P8095BH - Plastic DIP
	68 Pin	A8096BH - Ceramic PGA N8096BH - PLCC	A8097BH - Ceramic PGA N8097BH - PLCC
ROM	48 Pin		C8395BH - Ceramic DIP P8395BH - Plastic DIP
	68 Pin	A8396BH - Ceramic PGA N8396BH - PLCC	A8397BH - Ceramic PGA N8397BH - PLCC
EPROM	48 Pin		C8795BH - Ceramic DIP
	68 Pin	A8796BH - Ceramic PGA R8796BH - Ceramic LCC	A8797BH - Ceramic PGA R8797BH - Ceramic LCC

Figure 12. The MCS-96® Family Nomenclature

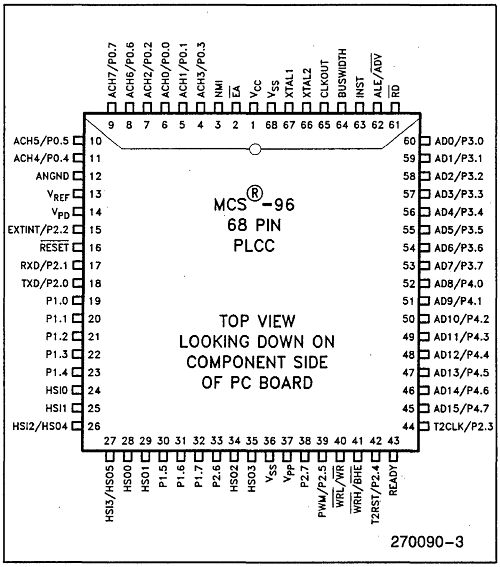
PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description
1	9	ACH7/P0.7/PMOD.3	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6/PMOD.2	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{\text{EA}}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	VCC	32	46	AD14/P4.6	55	23	P1.4
10	68	VSS	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	BUSWIDTH	37	41	$\overline{\text{BHE}}/\overline{\text{WRH}}$	60	18	TXD/P2.0/PVER/SALE
15	63	INST	38	40	$\overline{\text{WR}}/\overline{\text{WRL}}$	61	17	RXD/P2.1/PALE
16	62	ALE/$\overline{\text{ADV}}$	39	39	PWM/P2.5/$\overline{\text{PDO}}/\overline{\text{SPROG}}$	62	16	$\overline{\text{RESET}}$
17	61	$\overline{\text{RD}}$	40	38	P2.7	63	15	EXTINT/P2.2/$\overline{\text{PROG}}$
18	60	AD0/P3.0	41	37	VPP	64	14	VPD
19	59	AD1/P3.1	42	36	VSS	65	13	VREF
20	58	AD2/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4/PMOD.0
22	56	AD4/P3.4	45	33	P2.6	68	10	ACH5/P0.5/PMOD.1
23	55	AD5/P3.5	46	32	P1.7			

Figure 13. PGA, PLCC and LCC Function Pinouts



270090-2

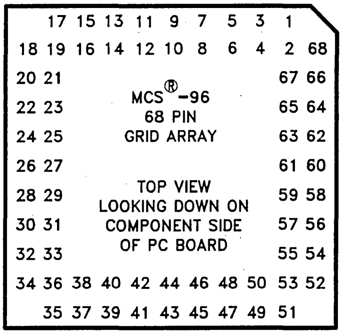
Figure 14. 48-Pin Package



270090-3

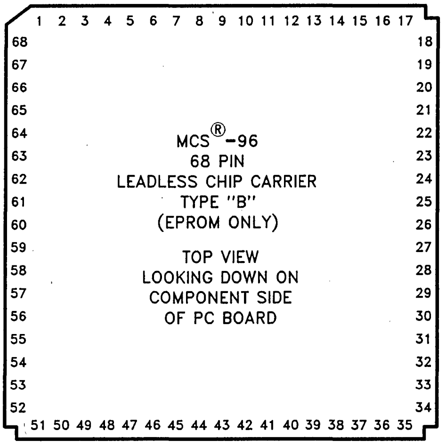
Figure 15. 68-Pin Package (PLCC - Top View)

Pins Facing Down



270090-4

Figure 16. 68-Pin Package (Pin Grid Array - Top View)



270090-5

Figure 17. 68-Pin Package (LCC - Top View)

PIN DESCRIPTIONS

Symbol	Name and Function
V_{CC}	Main supply voltage (5V).
V_{SS}	Digital circuit ground (0V). There are two V_{SS} pins, both of which must be connected.
V_{PD}	RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e. V_{CC} drops to zero), if $\overline{\text{RESET}}$ is activated before V_{CC} drops below spec and V_{PD} continues to be held within spec., the top 16 bytes in the Register File will retain their contents. $\overline{\text{RESET}}$ must be held low during the Power Down and should not be brought high until V_{CC} is within spec and the oscillator has stabilized.
V_{REF}	Reference voltage for the A/D converter (5V). V_{REF} is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0. Must be connected for A/D and Port 0 to function.
ANGND	Reference ground for the A/D converter. Must be held at nominally the same potential as V_{SS}.
V_{PP}	Programming voltage for the EPROM parts. It should be + 12.75V for programming. This pin is V_{BB} on 8X9X-90 parts. Systems that have this pin connected to ANGND through a capacitance (required on 8X9X-90 parts) do not need to change. Otherwise, tie to V_{CC}.
XTAL1	Input of the oscillator inverter and of the internal clock generator.
XTAL2	Output of the oscillator inverter.
CLKOUT	Output of the internal clock generator. The frequency of CLKOUT is ⅓ the oscillator frequency. It has a 33% duty cycle.
$\overline{\text{RESET}}$	Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to location 2080H is executed. Input high for normal operation. $\overline{\text{RESET}}$ has an internal pullup.
BUSWIDTH	Input for bus width selection. If CCR bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is a 1, a 16-bit bus cycle occurs. If BUSWIDTH is a 0 an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. This pin is the $\overline{\text{TEST}}$ pin on 8X9X-90 parts. Systems with $\overline{\text{TEST}}$ tied to V_{CC} do not need to change. If this pin is left unconnected, it will rise to V_{CC}.
NMI	A positive transition causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.
INST	Output high during an external memory read indicates the read is an instruction fetch. INST is valid throughout the bus cycle. INST is activated only during external memory accesses.
$\overline{\text{EA}}$	Input for memory select (External Access). $\overline{\text{EA}}$ equal to a TTL-high causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM/EPROM. $\overline{\text{EA}}$ equal to a TTL-low causes accesses to these locations to be directed to off-chip memory. $\overline{\text{EA}}$ = + 12.5V causes execution to begin in the Programming Mode. $\overline{\text{EA}}$ has an internal pulldown, so it goes to 0 unless driven otherwise. $\overline{\text{EA}}$ is latched at reset.
ALE/$\overline{\text{ADV}}$	Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is $\overline{\text{ADV}}$, it goes inactive high at the end of the bus cycle. $\overline{\text{ADV}}$ can be used as a chip select for external memory. ALE/$\overline{\text{ADV}}$ is activated only during external memory accesses.
$\overline{\text{RD}}$	Read signal output to external memory. $\overline{\text{RD}}$ is activated only during external memory reads.
$\overline{\text{WR}}$/$\overline{\text{WRL}}$	Write and Write Low output to external memory, as selected by the CCR. $\overline{\text{WR}}$ will go low for every external write, while $\overline{\text{WRL}}$ will go low only for external writes where an even byte is being written. $\overline{\text{WR}}$/$\overline{\text{WRL}}$ is activated only during external memory writes.
$\overline{\text{BHE}}$/$\overline{\text{WRH}}$	Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{\text{BHE}}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{\text{BHE}}$ = 1), to the high byte only (A0 = 1, $\overline{\text{BHE}}$ = 0), or both bytes (A0 = 0, $\overline{\text{BHE}}$ = 0). If the $\overline{\text{WRH}}$ function is selected, the pin will go low if the bus cycle is writing to an odd memory location. $\overline{\text{BHE}}$/$\overline{\text{WRH}}$ is activated only during external memory writes.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
READY	Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the memory controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. The bus cycle can be lengthened by up to 1 μs. When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle held not ready is available through configuration of CCR. READY has a weak internal pullup, so it goes to 1 unless externally pulled low.
HSI	Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as inputs by EPROM parts in Programming Mode.
HSO	Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.
Port 0	8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins are also a mode input to EPROM parts in the Programming Mode.
Port 1	8-bit quasi-bidirectional I/O port.
Port 2	8-bit multi-functional port. Six of its pins are shared with other functions in the 8096BH, the remaining 2 are quasi-bidirectional. These pins are also used to input and output control signals on EPROM parts in Programming Mode.
Ports 3 and 4	8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups. Ports 3 and 4 are also used as a command, address and data path by EPROM parts operating in the Programming Mode.

INSTRUCTION SET

The 8096BH instruction set makes use of six addressing modes as described below:

DIRECT—The operand is specified by an 8-bit address field in the instruction. The operand must be in the Register File or SFR space (locations 0000H through 00FFH).

IMMEDIATE—The operand itself follows the opcode in the instruction stream as immediate data. The immediate data can be either 8-bits or 16-bits as required by the opcode.

INDIRECT—An 8-bit address field in the instruction gives the word address of a word register in the Register File which contains the 16-bit address of the operand. The operand can be anywhere in memory.

INDIRECT WITH AUTO-INCREMENT—Same as Indirect, except that, after the operand is referenced, the word register that contains the operand's address is incremented by 1 if the operand is a byte, or by 2 if the operand is a word.

INDEXED (LONG AND SHORT)—The instruction contains an 8-bit address field and either an 8-bit or a 16-bit displacement field. The 8-bit address field gives the word address of a word register in the Register File which contains a 16-bit base address. The 8- or 16-bit displacement field contains a signed displacement that will be added to the base address to produce the address of the operand. The operand can be anywhere in memory.

The 8096BH contains a zero register at word address 0000H (and which contains 0000H). This register is available for performing comparisons and for use as a base register in indexed addressing. This effectively provides direct addressing to all 64K of memory.

In the 8096BH, the Stack Pointer is at word address 0018H in the Register File. If the 8-bit address field contains 18H, the Stack Pointer becomes the base register. This allows direct accessing of variables in the stack.

The following tables list the MCS-96 instructions, their opcodes, and execution times.

Instruction Summary

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	✓	✓	✓	✓	↑	—	
ADD/ADDB	3	D ← B + A	✓	✓	✓	✓	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	✓	✓	✓	↑	—	
SUB/SUBB	2	D ← D − A	✓	✓	✓	✓	↑	—	
SUB/SUBB	3	D ← B − A	✓	✓	✓	✓	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	✓	✓	✓	↑	—	
CMP/CMPB	2	D − A	✓	✓	✓	✓	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	✓	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	✓	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	
AND/ANDB	2	D ← D and A	✓	✓	0	0	—	—	
AND/ANDB	3	D ← B and A	✓	✓	0	0	—	—	
OR/ORB	2	D ← D or A	✓	✓	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	✓	✓	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ✓	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR [indirect]	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 − D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign(D)	✓	✓	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — — lsb ← 0	✓	?	✓	✓	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — — lsb → C	✓	?	✓	0	—	✓	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — — lsb → C	✓	✓	✓	0	—	✓	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left shift till msb = 1; D ← shift count	✓	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

Opcode and State Time Listing

		DIRECT			IMMEDIATE			INDIRECT③					INDEXED③				
								NORMAL			AUTO-INC.		SHORT			LONG	
MNEMONIC	OPERANDS	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES③	BYTES	STATE① TIMES③
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29

270090–45

NOTES:

*Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any Indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

③ State times shown for 16-bit bus.

Opcode and State Time Listing (Continued)

		DIRECT			IMMEDIATE			INDIRECT⑥					INDEXED⑥				
								NORMAL			AUTO-INC.		SHORT			LONG	
MNEMONIC	OPERANDS	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES⑥	BYTES	STATE① TIMES⑥
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	——	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	——	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	——	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	——	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
BR[ ]	E3	2	8	RET	F0	1	12/16⑤
				TRAP③	F7	1	21/24

270090-46

NOTES:

① Number of state times shown for internal/external operands.

③ The assembler does not accept this mnemonic.

④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.

⑤ State times for stack located internal/external.

⑥ State times shown for 16-bit bus.

CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.(8)

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.(8)

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

MNEMONIC	OPCODE	BYTES	STATE TIMES
DJNZ	EO	3	5/9 STATE TIME (NOT TAKEN/TAKEN)(8)

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES(8)	MNEMONIC	OPCODE	BYTES	STATES(8)
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR MNEMONIC	WORD		INSTR MNEMONIC	BYTE		INSTR MNEMONIC	DBL WD		STATE TIMES(8)
	OP	B		OP	B		OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT(7)
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT(7)
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT(7)

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES(8)	MNEMONIC	OPCODE	BYTES	STATES(8)
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST(6)	FF	1	166	SKIP	00	2	4

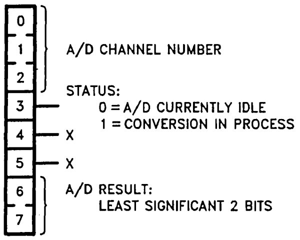
NORMALIZE

MNEMONIC	OPCODE	BYTES	STATE TIMES
NORML	0F	3	11 + 1 PER SHIFT

NOTES:

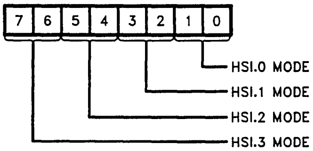
6. This instruction takes 2 states to pull RESET low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H. If a capacitor is tied to RESET, the pin may take longer to go low and may never reach the V_{OL} specification.
7. Execution will take at least 8 states, even for 0 shift.
8. State times shown for 16-bit bus.

A/D Result LO (02H)



270090-21

HSI_Mode (03H)

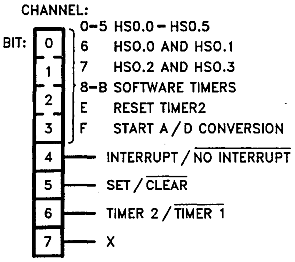


WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00	8 POSITIVE TRANSITIONS
01	EACH POSITIVE TRANSITION
10	EACH NEGATIVE TRANSITION
11	EVERY TRANSITION (POSITIVE AND NEGATIVE)

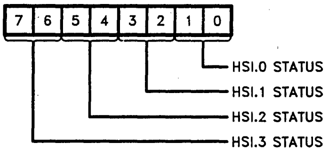
270090-22

HSO Command (06H)



270090-23

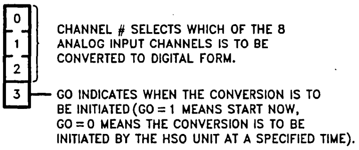
HSI_Status (06H)



WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

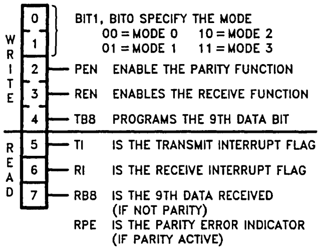
270090-25

A/D Command (02H)


0
1
2
3
CHANNEL # SELECTS WHICH OF THE 8 ANALOG INPUT CHANNELS IS TO BE CONVERTED TO DIGITAL FORM.
GO INDICATES WHEN THE CONVERSION IS TO BE INITIATED (GO = 1 MEANS START NOW, GO = 0 MEANS THE CONVERSION IS TO BE INITIATED BY THE HSO UNIT AT A SPECIFIED TIME).

270090-24

SPCON/SPSTAT (11H)


WRITE
READ
0
1
2
3
4
5
6
7
BIT1, BIT0 SPECIFY THE MODE
00 = MODE 0 10 = MODE 2
01 = MODE 1 11 = MODE 3
PEN ENABLE THE PARITY FUNCTION
REN ENABLES THE RECEIVE FUNCTION
TB8 PROGRAMS THE 9TH DATA BIT
TI IS THE TRANSMIT INTERRUPT FLAG
RI IS THE RECEIVE INTERRUPT FLAG
RB8 IS THE 9TH DATA RECEIVED (IF NOT PARITY)
RPE IS THE PARITY ERROR INDICATOR (IF PARITY ACTIVE)

270090-26

Baud Rate Calculations

Using XTAL1:

Mode 0: $$\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{4^*(B+1)}; B \neq 0$$

Others: $$\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{64^*(B+1)}$$

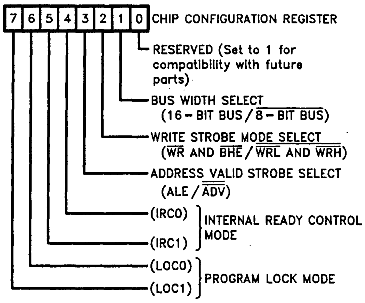
Using T2CLK:

Mode 0: $$\text{Baud Rate} = \frac{\text{T2CLK frequency}}{B}; B \neq 0$$

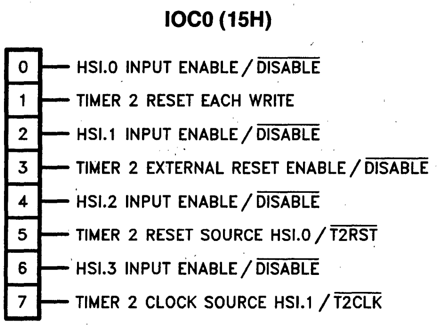
Others: $$\text{Baud Rate} = \frac{\text{T2CLK frequency}}{16^*B}; B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than Mode 0.

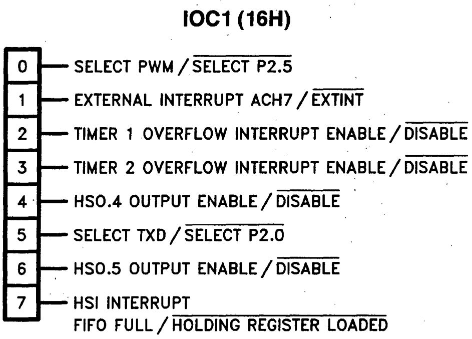
Chip Configuration


7
6
5
4
3
2
1
0
CHIP CONFIGURATION REGISTER
RESERVED (Set to 1 for compatibility with future parts)
BUS WIDTH SELECT (16-BIT BUS/$\overline{\text{8-BIT BUS}}$)
WRITE STROBE MODE SELECT ($\overline{\text{WR}}$ AND $\overline{\text{BHE}}$/$\overline{\text{WRL}}$ AND $\overline{\text{WRH}}$)
ADDRESS VALID STROBE SELECT (ALE/$\overline{\text{ADV}}$)
(IRC0)
(IRC1)
INTERNAL READY CONTROL MODE
(LOC0)
(LOC1)
PROGRAM LOCK MODE

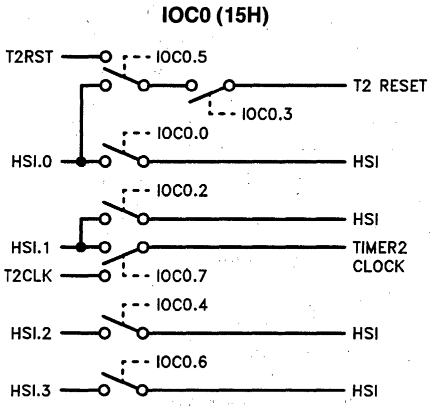
270090-32


IOC0 (15H)
0 HSI.0 INPUT ENABLE / DISABLE
1 TIMER 2 RESET EACH WRITE
2 HSI.1 INPUT ENABLE / DISABLE
3 TIMER 2 EXTERNAL RESET ENABLE / DISABLE
4 HSI.2 INPUT ENABLE / DISABLE
5 TIMER 2 RESET SOURCE HSI.0 / T2RST
6 HSI.3 INPUT ENABLE / DISABLE
7 TIMER 2 CLOCK SOURCE HSI.1 / T2CLK

270090-30

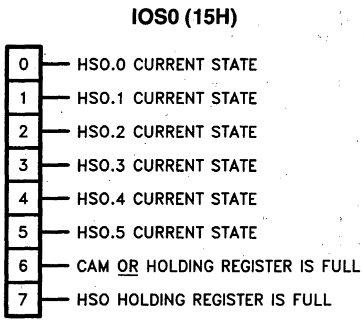

IOC1 (16H)
0 SELECT PWM / SELECT P2.5
1 EXTERNAL INTERRUPT ACH7 / EXTINT
2 TIMER 1 OVERFLOW INTERRUPT ENABLE / DISABLE
3 TIMER 2 OVERFLOW INTERRUPT ENABLE / DISABLE
4 HSO.4 OUTPUT ENABLE / DISABLE
5 SELECT TXD / SELECT P2.0
6 HSO.5 OUTPUT ENABLE / DISABLE
7 HSI INTERRUPT FIFO FULL / HOLDING REGISTER LOADED

270090-31

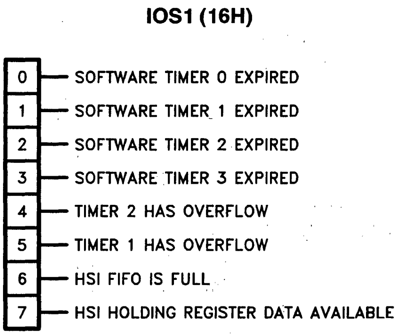


270090-29

Vector	Vector Location (High Byte)	(Low Byte)	Priority
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)



270090-27


IOS1 (16H)
0 SOFTWARE TIMER 0 EXPIRED
1 SOFTWARE TIMER 1 EXPIRED
2 SOFTWARE TIMER 2 EXPIRED
3 SOFTWARE TIMER 3 EXPIRED
4 TIMER 2 HAS OVERFLOW
5 TIMER 1 HAS OVERFLOW
6 HSI FIFO IS FULL
7 HSI HOLDING REGISTER DATA AVAILABLE

270090-28

ELECTRICAL CHARACTERISTICS ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .... 0°C to +70°C

Storage Temperature .... −40°C to +150°C

Voltage from $\overline{EA}$ or V_{PP}
to V_{SS} or ANGND .... −0.3V to +13.0V

Voltage from Any Other Pin to
V_{SS} or ANGND .... −0.3V to +7.0V*

Average Output Current from Any Pin .... 10 mA

Power Dissipation.......................... 1.5W

*This includes V_{PP} on ROM and CPU only devices.

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	0	+70	C
V_{CC}	Digital Supply Voltage	4.50	5.50	V
V_{REF}	Analog Supply Voltage	4.50	5.50	V
f_{OSC}	Oscillator Frequency	6.0	12	MHz
V_{PD}	Power-Down Supply Voltage	4.50	5.50	V

NOTE:
ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS (Test Conditions: V_{CC}, V_{REF}, V_{PD}, V_{PP}, V_{EA} = 5.0V ±0.5V; F_{OSC} = 6.0 MHz; T_A = 0°C to 70°C; V_{SS}, ANGND = 0V)

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{CC}	V_{CC} Supply Current (0°C ≤ T_A ≤ 70°C)		240	mA	All Outputs Disconnected.
I_{CC1}	V_{CC} Supply Current (T_A = 70°C)		185	mA	
I_{PD}	V_{PD} Supply Current		1	mA	Normal operation and Power-Down.
I_{REF}	V_{REF} Supply Current		8	mA	
V_{IL}	Input Low Voltage (Except $\overline{RESET}$)	−0.3	+0.8	V	
V_{IL1}	Input Low Voltage, $\overline{RESET}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage (Except $\overline{RESET}$, NMI, XTAL1)	2.0	V_{CC} +0.5	V	
V_{IH1}	Input High Voltage, $\overline{RESET}$ Rising	2.4	V_{CC} +0.5	V	
V_{IH2}	Input High Voltage, $\overline{RESET}$ Falling Hysteresis	2.1	V_{CC} +0.5	V	
V_{IH3}	Input High Voltage, NMI, XTAL1	2.2	V_{CC} +0.5	V	
I_{LI}	Input Leakage Current to each pin of HSI, P3, P4, and to P2.1.		±10	μA	V_{in} = 0 to V_{CC}
I_{LI1}	D.C. Input Leakage Current to each pin of P0		+3	μA	V_{in} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{EA}$		100	μA	V_{IH} = 2.4V
I_{IL}	Input Low Current to each pin of P1, and to P2.6, P2.7.		−125	μA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{RESET}$	−0.25	−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4, READY, BUSWIDTH		−50	μA	V_{IL} = 0.45V
V_{OL}	Output Low Voltage on Quasi-Bidirectional port pins and P3, P4 when used as ports		0.45	V	I_{OL} = 0.8 mA (Note 1)
V_{OL1}	Output Low Voltage on Quasi-Bidirectional port pins and P3, P4 when used as ports		0.75	V	I_{OL} = 2.0 mA (Notes 1, 2, 3)
V_{OL2}	Output Low Voltage on Standard Output pins, $\overline{RESET}$ and Bus/Control Pins		0.45	V	I_{OL} = 2.0 mA (Notes 1, 2, 3, 4)

D.C. CHARACTERISTICS (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{OH}	Output High Voltage on Quasi-Bidirectional pins	2.4		V	$I_{OH} = -20\ \mu A$ (Note 1)
V_{OH1}	Output High Voltage on Standard Output pins and Bus/Control pins	2.4		V	$I_{OH} = -200\ \mu A$ (Note 1)
I_{OH3}	Output High Current on $\overline{RESET}$	−50		μA	$V_{OH} = 2.4V$
C_S	Pin Capacitance (Any Pin to V_{SS})		10	pF	fTEST = 1.0 MHz

NOTES:

1. Quasi-bidirectional pins include those on P1, for P2.6 and P2.7. Standard Output Pins include TXD, RXD (Mode 0 only), PWM, and HSO pins. Bus/Control pins include CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, INST and AD0–15.
2. Maximum current per pin must be externally limited to the following values if V_{OL} is held above 0.45V.
   I_{OL} on quasi-bidirectional pins and Ports 3 and 4 when used as ports: 4.0 mA
   I_{OL} on standard output pins and $\overline{RESET}$: 8.0 mA
   I_{OL} on Bus/Control pins: 2.0 mA
3. During normal (non-transient) operation the following limits apply:
   Total I_{OL} on Port 1 must not exceed 8.0 mA.
   Total I_{OL} on P2.0, P2.6, $\overline{RESET}$ and all HSO pins must not exceed 15 mA.
   Total I_{OL} on Port 3 must not exceed 10 mA.
   Total I_{OL} on P2.5, P2.7, and Port 4 must not exceed 20 mA.
4. I_{OL} on HSO.X (X = 0, 4, 5) = 1.6 mA @ 0.5V.

A.C. CHARACTERISTICS V_{CC}, V_{PD} = 4.5 to 5.5V; T_A = 0°C to 70°C; f_{OSC} = 6.0 to 12.0 MHz

Test Conditions: Load Capacitance on Output Pins = 80 pF
Oscillator Frequency = 10 MHz

TIMING REQUIREMENTS (Other system components must meet these specs.)

Symbol	Parameter	Min	Max	Units
T_{CLYX}(4)	READY Hold after CLKOUT Edge	0(1)		ns
T_{LLYV}	End of ALE/$\overline{ADV}$ to READY Valid		2Tosc−70	ns
T_{LLYH}	End of ALE/$\overline{ADV}$ to READY High	2Tosc+40	4Tosc−80	ns
T_{YLYH}	Non-Ready Time		1000	ns
T_{AVDV}(6)	Address Valid to Input Data Valid		5Tosc−120	ns
T_{RLDV}	$\overline{RD}$ Active to Input Data Valid		3Tosc−100	ns
T_{RHDX}	Data Hold after $\overline{RD}$ Inactive	0		ns
T_{RHDZ}	$\overline{RD}$ Inactive to Input Data Float	0	Tosc−25	ns
T_{AVGV}(4)(6)	Address Valid to BUSWIDTH Valid		2 Tosc −125	ns
T_{LLGX}(4)	BUSWIDTH Hold after ALE/$\overline{ADV}$ Low	Tosc +40		ns
T_{LLGV}(4)	ALE/$\overline{ADV}$ Low to BUSWIDTH Valid		Tosc −75	ns

NOTES:

1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at 2Tosc+55 (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
4. Pins not bonded out on 48-pin parts.
6. The term "Address Valid" applies to AD0–15, $\overline{BHE}$ and INST.

A.C. CHARACTERISTICS (Continued)

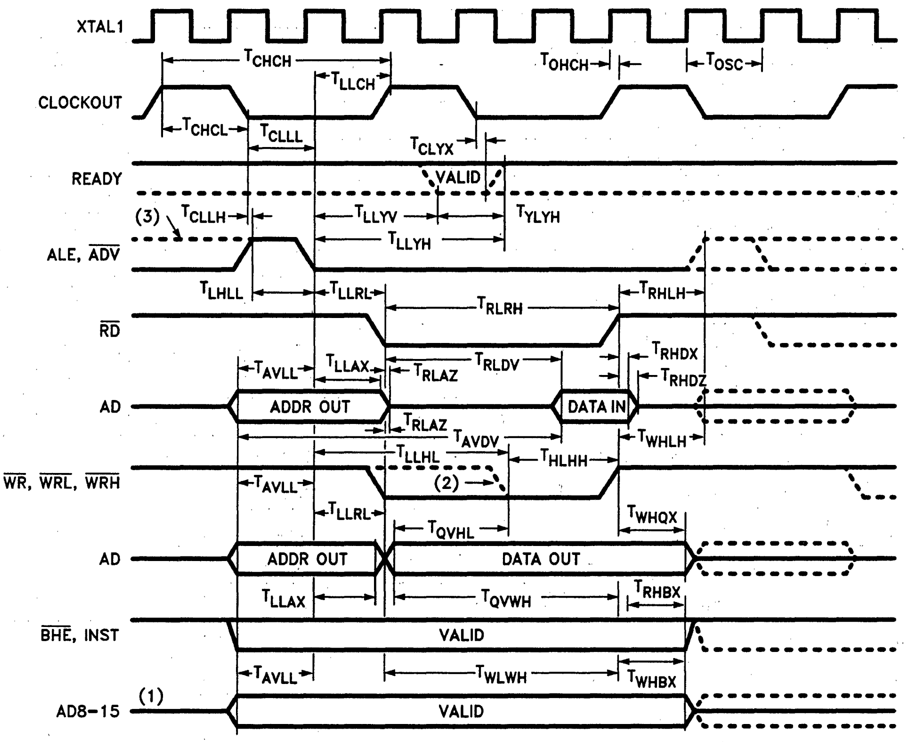
TIMING RESPONSES (MCS-96 parts meet these specs.)

Symbol	Parameter	Min	Max	Units
F_{XTAL}	Oscillator Frequency	6.0	12.0	MHz
T_{OSC}	Oscillator Period	83	166	ns
T_{OHCH}	XTAL1 Rising Edge to Clockout Rising Edge	0[4]	120[4]	ns
T_{CHCH}[4]	CLKOUT Period[3]	3Tosc[3]	3Tosc[3]	ns
T_{CHCL}[4]	CLKOUT High Time	Tosc−35	Tosc+10	ns
T_{CLLH}[4]	CLKOUT Low to ALE High	−20	+25	ns
T_{LLCH}[4]	ALE/$\overline{\text{ADV}}$ Low to CLKOUT High	Tosc−25	Tosc+45	ns
T_{LHLL}	ALE/$\overline{\text{ADV}}$ High Time	Tosc−30	Tosc+35[5]	ns
T_{AVLL}[6]	Address Setup to End of ALE/$\overline{\text{ADV}}$	Tosc−50		ns
T_{RLAZ}[7]	$\overline{\text{RD}}$ or $\overline{\text{WR}}$ Low to Address Float		25	ns
T_{LLRL}	End of ALE/$\overline{\text{ADV}}$ to $\overline{\text{RD}}$ or $\overline{\text{WR}}$ Active	Tosc−40		ns
T_{LLAX}[7]	Address Hold after End of ALE/$\overline{\text{ADV}}$	Tosc−40		ns
T_{WLWH}	$\overline{\text{WR}}$ Pulse Width	3Tosc−35		ns
T_{QVWH}	Output Data Valid to End of $\overline{\text{WR}}$/$\overline{\text{WRL}}$/$\overline{\text{WRH}}$	3Tosc−60		ns
T_{WHQX}	Output Data Hold after $\overline{\text{WR}}$/$\overline{\text{WRL}}$/$\overline{\text{WRH}}$	Tosc−50		ns
T_{WHLH}	End of $\overline{\text{WR}}$/$\overline{\text{WRL}}$/$\overline{\text{WRH}}$ to ALE/$\overline{\text{ADV}}$ High	Tosc−75		ns
T_{RLRH}	$\overline{\text{RD}}$ Pulse Width	3Tosc−30		ns
T_{RHLH}	End of $\overline{\text{RD}}$ to ALE/$\overline{\text{ADV}}$ High	Tosc−45		ns
T_{CLLL}[4]	CLOCKOUT Low to ALE/$\overline{\text{ADV}}$ Low	Tosc−40	Tosc+35	ns
T_{RHBX}[4]	$\overline{\text{RD}}$ High to INST, $\overline{\text{BHE}}$, AD8-15 Inactive	Tosc−25	Tosc+30	ns
T_{WHBX}[4]	$\overline{\text{WR}}$ High to INST, $\overline{\text{BHE}}$, AD8-15 Inactive	Tosc−50	Tosc+100	ns
T_{HLHH}	$\overline{\text{WRL}}$, $\overline{\text{WRH}}$ Low to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ High	2Tosc−35	2Tosc+40	ns
T_{LLHL}	ALE/$\overline{\text{ADV}}$ Low to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ Low	2Tosc−30	2Tosc+55	ns
T_{QVHL}	Output Data Valid to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ Low	Tosc−60		ns

NOTES:

2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ± 10 ns if Tosc is constant and the rise and fall times on XTAL1 are less than 10 ns.
4. Pins not bonded out on 48-pin parts.
5. Max spec applies only to ALE. Min spec applies to both ALE and $\overline{\text{ADV}}$.
6. The term "Address Valid" applies to AD0-15, $\overline{\text{BHE}}$ and INST.
7. The term " Address" in this definition applies to AD0-7 for 8-bit cycles, and AD0-15 for 16-bit cycles.

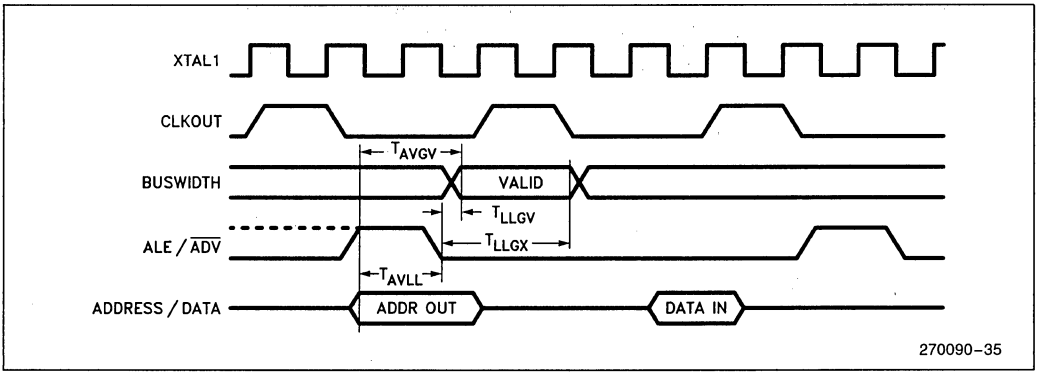
WAVEFORM



270090-47

NOTES:
(1) 8-bit bus only.
(2) 8-bit bus; or when write strobe mode selected.
(3) When ADV selected.

WAVEFORM—BUSWIDTH PIN



A.C. CHARACTERISTICS—SERIAL PORT—SHIFT REGISTER MODE

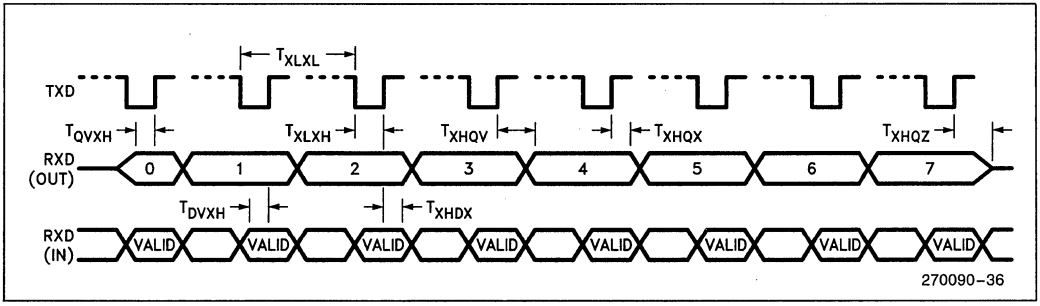
SERIAL PORT TIMING—SHIFT REGISTER MODE

Test Conditions: T_A = 0°C to +70°C; V_{CC} = 5V ±10%; V_{SS} = 0V; Load Capacitance = 80 pF

Symbol	Parameter	Min	Max	Units
T_{XLXL}	Serial Port Clock Period	$8T_{OSC}$		ns
T_{XLXH}	Serial Port Clock Falling Edge to Rising Edge	$4T_{OSC} - 50$	$4T_{OSC} + 50$	ns
T_{QVXH}	Output Data Setup to Clock Rising Edge	$3T_{OSC}$		ns
T_{XHQX}	Output Data Hold After Clock Rising Edge	$2T_{OSC} - 50$		ns
T_{XHQV}	Next Output Data Valid After Clock Rising Edge		$2T_{OSC} + 50$	ns
T_{DVXH}	Input Data Setup to Clock Rising Edge	$2T_{OSC} + 200$		ns
T_{XHDX}	Input Data Hold After Clock Rising Edge	0		ns
T_{XHQZ}	Last Clock Rising to Output Float		$5T_{OSC}$	ns

WAVEFORM—SERIAL PORT—SHIFT REGISTER MODE

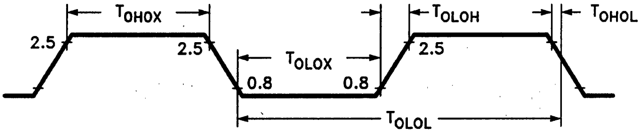
SERIAL PORT WAVEFORM—SHIFT REGISTER MODE



EXTERNAL CLOCK DRIVE

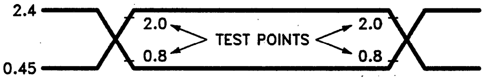
Symbol	Parameter	Min	Max	Units
$1/T_{OLOL}$	Oscillator Frequency	6	12	MHz
T_{OHOX}	High Time	25		ns
T_{OLOX}	Low Time	25		ns
T_{OLOH}	Rise Time		15	ns
T_{OHOL}	Fall Time		15	ns

EXTERNAL CLOCK DRIVE WAVEFORMS



270090-48

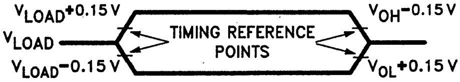
A.C. TESTING INPUT, OUTPUT WAVEFORM



270090-49

A.C. Testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

FLOAT WAVEFORM



270090-51

For Timing Purposes a Port Pin is no Longer Floating when a 100 mV change from Load Voltage Occurs, and Begins to Float when a 100 mV change from the Loaded V_{OH}/V_{OL} Level occurs $I_{OL}/I_{OH} \geq \pm 15$ mA.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097BH, 8397BH, 8095BH, 8395BH, 8797BH, 8795BH.

The absolute conversion accuracy is dependent on the accuracy of V_{REF}. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at V_{REF} = 5.120V.

OPERATING CONDITIONS

V_{CC}, V_{PD}, V_{REF} .... 4.5V to 5.5V
V_{SS}, ANGND 0.0V
T_A .... 0°C to 70°C
F_{OSC} .... 6.0 to 12.0 MHz
Test Conditions:
V_{REF} 5.120V

Parameter	Typical*(1)	Minimum	Maximum	Units**	Notes
Resolution		1024 10	1024 10	Levels Bits	
Absolute Error		0	±4	LSBs	
Full Scale Error	−0.5 ±0.5			LSBs	
Zero Offset Error	±0.5			LSBs	
Non-Linearity		0	±4	LSBs	
Differential Non-Linearity		0	±2	LSBs	
Channel-to-Channel Matching		0	±1	LSBs	
Repeatability	±0.25			LSBs	1
Temperature Coefficients: Offset Full Scale Differential Non-Linearity	 0.009 0.009 0.009			 LSB/°C LSB/°C LSB/°C	 1 1 1
Off Isolation		−60		dB	1, 2, 4
Feedthrough	−60			dB	1, 2
V_{CC} Power Supply Rejection	−60			dB	1, 2
Input Resistance		1K	5K	Ω	1
D.C. Input Leakage		0	3.0	μA	
Sample Delay		$3T_{OSC} - 50$	$3T_{OSC} + 50$	ns	1, 3
Sample Time		$12T_{OSC} - 50$	$12T_{OSC} + 50$	ns	1
Sampling Capacitor			2	pF	

NOTES:
* These values are expected for most parts at 25°C.
** An "LSB", as used here, is defined in the glossary which follows and has a value of approximately 5 mV.
1. These values are not tested in production and are based on theoretical estimates and laboratory tests.
2. DC to 100 KHz.
3. For starting the A/D with an HSO Command.
4. Multiplexer Break-Before-Make Guaranteed.

A/D GLOSSARY OF TERMS

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An actual characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversions under the same conditions.

BREAK-BEFORE-MAKE—The property of a multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g. the converter will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE CENTER—The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION—The point at which the converter changes from an output code of Q, to a code of Q + 1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK—See "Off-Isolation".

D.C. INPUT LEAKAGE—Leakage current to ground from an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D Converter after the sample window closes.

FULL SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at V_{IN} = 0.5 LSB, its last code transition at V_{IN} = (V_{REF} − 1.5 LSB) and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB—Least Significant Bit: The voltage corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For a 10-bit converter with a reference voltage of 5.12V, one LSB is 5.0 mV. Note that this is different than digital LSBs, since an uncertainty of two LSB, when referring to an A/D converter, equals 10 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 20 mV.)

MONOTONIC—The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES—For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY—The maximum deviation of code transitions of the terminal-based characteristic from the corresponding code transitions of the ideal characteristic.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY—The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY—The variation in the sample delay.

SAMPLE TIME—The time that the sample window is open.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION—An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An actual characteristic which has been rotated and translated to remove zero offset and full scale error.

V_{CC} REJECTION—Attenuation of noise on the V_{CC} line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

EPROM CHARACTERISTICS

The 879XBH contains 8K bytes of ultraviolet Erasable and Electrically Programmable Read Only Memory (EPROM) for internal storage. This memory can be programmed in a variety of ways—including at run-time under software control.

The EPROM is mapped into memory locations 2000H through 3FFFH if $\overline{EA}$ is a TTL high. However, applying +12.75V to $\overline{EA}$ when the chip is reset will place the 879XBH in EPROM Programming Mode. The Programming Mode has been implemented to support EPROM programming and verification.

When an 879XBH is in Programming Mode, special hardware functions are available to the user. These functions include algorithms for slave, gang and auto EPROM programming.

Programming the 879XBH

Three flexible EPROM programming modes are available on the 879XBH—auto, slave and run-time. These modes can be used to program 879XBHs in a gang, stand alone or run-time environment.

The Auto Programming Mode enables an 879XBH to program itself, and up to 15 other 879XBHs, with the 8K bytes of code beginning at address 4000H on its external bus. The Slave Mode provides a standard interface that enables any number of 879XBHs to be programmed by a master device such as an EPROM programmer. The Run-Time Mode allows individual EPROM locations to be programmed at run-time under complete software control.

In the Programming Mode, some I/O pins have been renamed. These new pin functions are used to determine the programming function that is performed, provide programming ALEs, provide slave ID numbers and pass error information. Figure 19 shows how the pins are renamed. Figure 20 describes each new pin function.

While in Programming Mode, PMODE selects the programming function that is performed (see Figure 18). When not in the Programming Mode, Run-Time programming can be done at any time.

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming Mode
0DH	Program Configuration Byte
0EH–0FH	Reserved

Figure 18. Programming Function PMODE Values

To guarantee proper execution, the pins of PMODE and SID must be in their desired state before the $\overline{RESET}$ pin is allowed to rise and reset the part. Once the part is reset, it is in the selected mode and should not be switched to another mode without a new reset sequence.

When $\overline{EA}$ selects the Programming Mode, the chip reset sequence loads the CCR from the Programming Chip Configuration Byte (PCCB). This is a separate EPROM location that is not mapped under normal operation. PCCR is only important when programming in the Auto Programming Mode. In this mode, the 879XBH that is being programmed gets the data to be programmed from external memory over the system bus. Therefore, PCCR must correctly correspond to the memory system in the programming setup, which is not necessarily the memory organization of the application.

The following sections describe 879XBH programming in each programming mode.

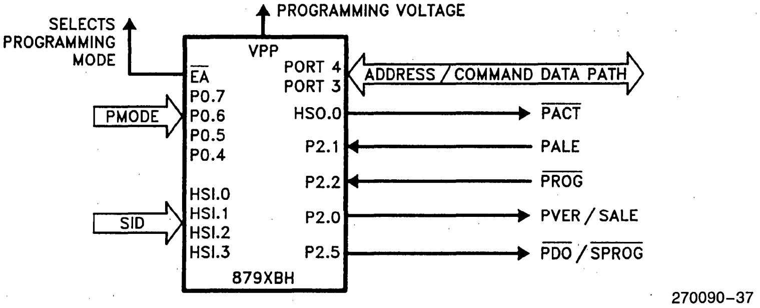


Figure 19. Programming Mode Pin Functions

Name	Function
PMODE	Programming Mode Select. Determines the EPROM programming algorithm that is performed. PMODE is sampled after a chip reset and should be static while the part is operating.
SID	Slave ID Number. Used to assign each slave a pin of Port 3 or 4 to use for passing programming verification acknowledgement. For example, if gang programming in the Slave Programming Mode, the slave with SID = 0001 will use Port 3.1 to signal correct or incorrect program verification.
PALE	Programming ALE input. Accepted by an 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain a command/address.
$\overline{\text{PROG}}$	Programming Pulse. Accepted by an 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain the data to be programmed. A falling edge on $\overline{\text{PROG}}$ signifies data valid and starts the programming cycle. A rising edge on $\overline{\text{PROG}}$ will halt programming in the slaves.
$\overline{\text{PACT}}$	Programming Active. Used in the Auto Programming Mode to indicate when programming activity is complete.
PVER	Program Verified. A signal output after a programming operation by parts in the Slave Programming Mode.
$\overline{\text{PDO}}$	Programming Duration Overflowed. A signal output by parts in the Slave Programming Mode. Used to signify that the $\overline{\text{PROG}}$ pulse applied for a programming operation was longer than allowed.
SALE	Slave ALE. Output signal from an 879XBH in the Auto Programming Mode. A falling edge on SALE indicates that Ports 3 and 4 contain valid address/command information for slave 879XBHs that may be attached to the master.
$\overline{\text{SPROG}}$	Slave Programming Pulse. Output from an 879XBH in the Auto Programming Mode. A falling edge on $\overline{\text{SPROG}}$ indicates that Ports 3 and 4 contain valid data for programming into slave 879XBHs that may be attached to the master.
PORTS 3 and 4	Address/Command/Data Bus. Used to pass commands, addresses and data to and from slave mode 879XBHs. Used by chips in the Auto Programming Mode to pass command, addresses and data to slaves. Also used in the Auto Programming Mode as a regular system bus to access external memory. Should have pullups to V_{CC} (15 KΩ).

Figure 20. Programming Mode Pin Definitions

AUTO PROGRAMMING MODE

The Auto Programming Mode provides the ability to program the internal 879XBH EPROM without having to use a special EPROM programmer. In this mode, the 879XBH simply programs itself with the data found at external locations 4000H through 5FFFH. All that is required is that some sort of external memory reside at these locations, that $\overline{EA}$ selects the Programming Mode and that V_{PP} is applied. Figure 21 shows a minimum configuration for using an 8K × 8 EPROM to program one 879XBH in the Auto Programming Mode.

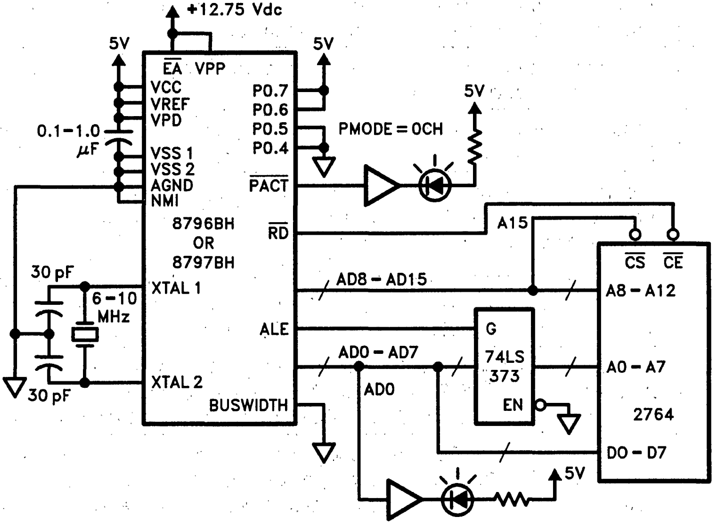
The 879XBH first reads a word from external memory, then the Modified Quick-Pulse Programming™ Algorithm (described later) is used to program the appropriate EPROM location. Since the erased state of a byte is 0FFH, the Auto Programming Mode will skip locations where the data to be programmed is 0FFH. When all 8K has been programmed, $\overline{PACT}$ goes high and the part outputs a 0 on Port 2.0 if it programmed correctly and a 1 if it failed.

Gang Programming with the Auto Programming Mode

An 879XBH in the Auto Programming Mode can also be used as a programmer for up to 15 other 879XBHs that are configured in the Slave Programming Mode. To accomplish this, the 879XBH acting as the master outputs the slave command/data pairs on Ports 3 and 4 necessary to program slave parts with the same data it is programming itself with. Slave ALE (SALE) and Slave $\overline{PROG}$ ($\overline{SPROG}$) signals are provided by the master to the slaves to demultiplex the commands from the data. Figure 22 is a block diagram of a gang programming system using one 879XBH in the Auto Programming Mode. The Slave Programming Mode is described in the next section.

The master 879XBH first reads a word from the external memory controlled by ALE, $\overline{RD}$ and $\overline{WR}$. It then drives Ports 3 and 4 with a Data Program command using the appropriate address and alerts the slaves with a falling edge on SALE. Next, the data to be programmed is driven onto Ports 3 and 4 and slave programming begins with a falling edge on $\overline{SPROG}$. At the same time, the master begins to program its own EPROM location with the data read in. Intel's Modified Quick-Pulse Programming™ Algorithm is used, with Data Verify commands being given to the slaves after each programming pulse.

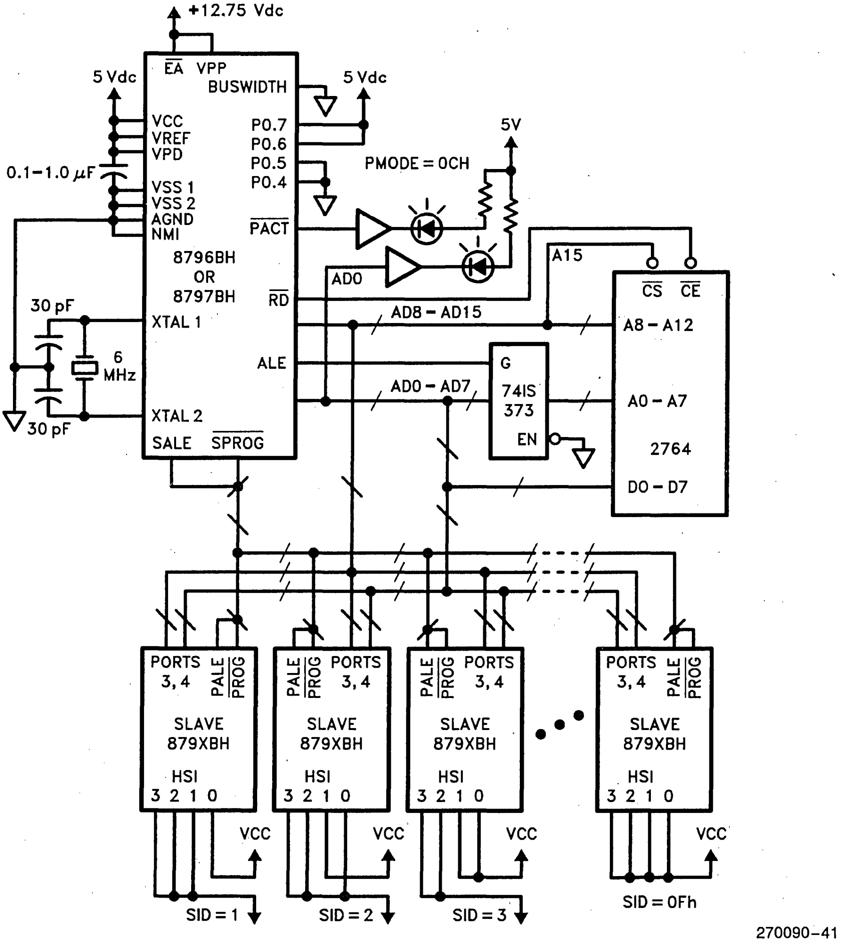
When programming is complete, $\overline{PACT}$ goes high and Ports 3 and 4 are driven with all 1s if all parts programmed correctly. Individual bits of Port 3 and 4 will be driven to 0 if the slave with that bit number as an SID did not program correctly. The 879XBH used as the master assigns itself an SID of 0.



NOTE:
Ports 3 and 4 should have pullups to V_{CC}

270090–40

Figure 21. The Auto Programming Mode



270090–41

NOTE:
EA and V_{PP} on slaves must be at +12.75 V_{dc}. Each slave's PMODE must equal 05H. Ports 3 and 4 should have pullups to V_{CC}. Minimum configuration connections must also be made for slaves. A 10 MHz clock is recommended for the slaves.

Figure 22. Gang Programming with the Auto Programming Mode

SLAVE PROGRAMMING MODE

Any number of 879XBHs can be programmed by a master programmer through the Slave Programming Mode.

The programming device uses Ports 3 and 4 of the parts being programmed as a command/data path. The slaves accept signals on PALE (Program ALE) and $\overline{\text{PROG}}$ (Program Enable) to demultiplex the commands and data. The slaves also use PVER, $\overline{\text{PDO}}$ and Ports 3 and 4 to pass error information to the programmer. Support for gang programming of up to 16 879XBHs is provided. If each part is given a unique SID (Slave ID Number) an 879XBH in the Auto Programming Mode can be used as a master to program itself and up to 15 other slave 879XBHs. There is, however, no 879XBH dependent limit to the number of parts that can be gang programmed in the slave mode.

It is important to note that the interface to an 879XBH in the slave mode is similar to a multiplexed bus. Attempting to issue consecutive PALE pulses without a corresponding $\overline{\text{PROG}}$ pulse will produce unexpected results. Similarly, issuing consecutive $\overline{\text{PROG}}$ pulses without the corresponding $\overline{\text{PALE}}$ pulses immediately preceding is equally unpredictable.

Slave Programming Commands

The commands sent to the slaves are 16-bits wide and contain two fields. Bits 14 and 15 specify the action that the slaves are to perform. Bits 0 through 13 specify the address upon which the action is to take place. Commands are sent via Ports 3 and 4 and are available to cause the slaves to program a word, verify a word, or dump a word (Table 4). The address part of the command sent to the slaves ranges from 2000H to 3FFFH and refers to the internal EPROM memory space. The following sections describe each Slave Programming Mode command.

Table 4. Slave Programming Mode Commands

P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

DATA PROGRAM COMMAND—After a Data Program Command has been sent to the slaves, $\overline{\text{PROG}}$ must be pulled low to cause the data on Ports 3 and 4 to be programmed into the location specified during the command. The falling edge of $\overline{\text{PROG}}$ is not only used to indicate data valid, but also triggers the hardware programming of the word specified. The slaves will begin programming 48 states after $\overline{\text{PROG}}$ falls, and will continue to program the location until $\overline{\text{PROG}}$ rises.

After the rising edge of $\overline{\text{PROG}}$, the slaves automatically perform a verification of the address just programmed. The result of this verification is then output on PVER (Program Verify) and $\overline{\text{PDO}}$ (Program Duration Overflowed). Therefore, verification information is available following the Data Program Command for programming systems that cannot use the Data Verify command.

If PVER and $\overline{\text{PDO}}$ of all slaves are 1s after $\overline{\text{PROG}}$ rises then the data program was successful everywhere. If PVER is a 0 in any slave, then the data programmed did not verify correctly in that part. If $\overline{\text{PDO}}$ is a 0 in any slave, then the programming pulse in those parts was terminated by an internal safety feature rather than the rising edge of $\overline{\text{PROG}}$. The safety feature prevents over-programming in the slave mode. Figure 23 shows the relationship of PALE, $\overline{\text{PROG}}$, PVER and $\overline{\text{PDO}}$ to the Command/Data Path on Ports 3 and 4 for the Data Program Command.

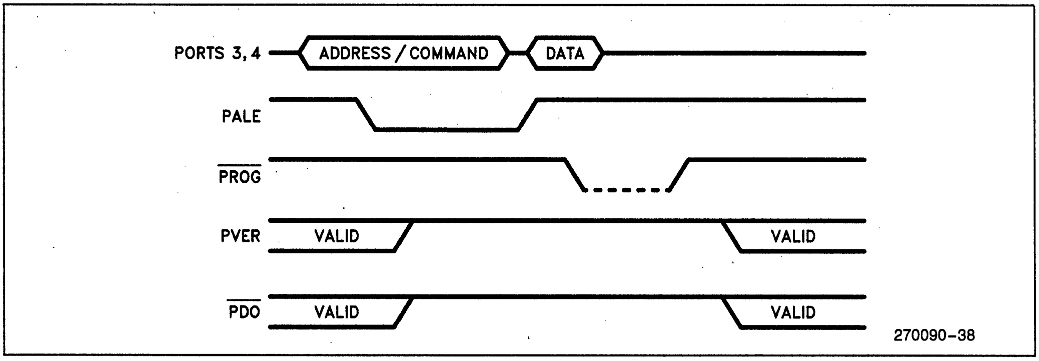


Figure 23. Data Program Signals in Slave Programming Mode

DATA VERIFY COMMAND—When the Data Verify Command is sent, the slaves respond by driving one bit of Port 3 or 4 to indicate correct or incorrect verification of the previous Data Program. A 1 indicates correct verification, while a 0 indicates incorrect verification. The SID (Slave ID Number) of each slave determines which bit of the command/data path is driven. $\overline{\text{PROG}}$ from the programmer governs when the slaves drive the bus. Figure 24 shows the relationship of Ports 3 and 4 to PALE and $\overline{\text{PROG}}$.

This command is always preceded by a Data Program Command in a programming system with as many as 16 slaves. However, a Data Verify Command does not have to follow every Data Program Command.

WORD DUMP COMMAND — When the Word Dump Command is issued, the 879XBH being programmed adds 2000H to the address field of the command and places the value found at the new address on Ports 3 and 4. For example, sending the command #0100H to a slave will result in the slave placing the word found at location 2100H on Ports 3 and 4. $\overline{\text{PROG}}$ from the programmer governs when the slave drives the bus. The signals are the same as shown in Figure 24.

Note that this command will work only when just one slave is attached to the bus, and that there is no restriction on commands that precede or follow a Word Dump Command.

Gang Programming with the Slave Programming Mode

Gang programming of 879XBHs can be done using the Slave Programming Mode. There is no 879XBH based limit on the number of chips that may be hooked to the same Port 3/Port 4 data path for gang programming.

If more than 16 chips are being gang programmed, the PVER and $\overline{\text{PDO}}$ outputs of each chip could be used for verification. The master programmer could issue a data program command then either watch every chip's error signals, or AND all the signals together to get a system PVER and $\overline{\text{PDO}}$.

If 16 or fewer 879XBHs are to be gang programmed at once, a more flexible form of verification is available. By giving each chip being programmed a unique SID, the master programmer could then issue a data verify command after the data program command. When a verify command is seen by the slaves, each will drive one pin of Port 3 or 4 with a 1 if the programming verified correctly or a 0 if programming failed. The SID is used by each slave to determine which Port 3, 4 bit it is assigned. An 879XBH in the auto programming mode could be the master programmer if 15 or fewer slaves need to be programmed (See Gang Programming with the Auto Programming Mode).

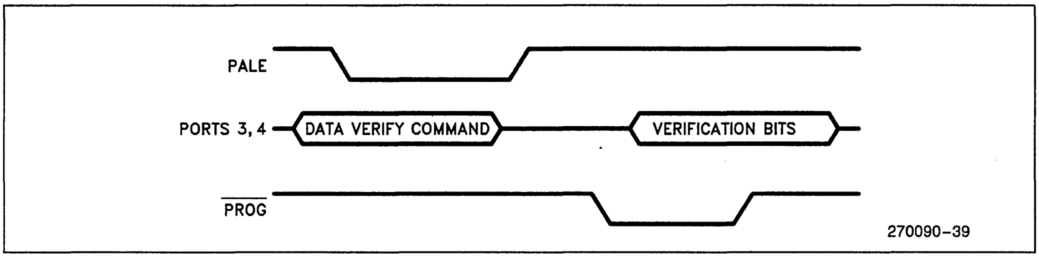


Figure 24. Data Verify Command Signals

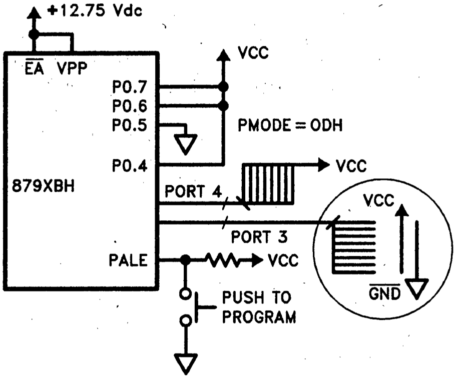
AUTO CONFIGURATION BYTE PROGRAMMING MODE

The CCB (location 2018H) can be treated just like any other EPROM location, and programmed using any programming mode. But to provide for simple programming of the CCB when no other locations need to be programmed, the Auto Configuration Byte Programming Mode is provided. Programming in this mode also programs PCCB. Figure 25 shows a block diagram for using the Auto Configuration Byte Programming Mode.

With PMODE = 0DH and 0FF on Port 4, CCB and PCCB will be programmed the value on Port 3 when a logic 0 is placed on PALE. After programming is complete, PVER will be driven to a 1 if the bytes programmed correctly, and a 0 if the programming failed.

This method of programming is the only way to program PCCB. PCCB is a non-memory mapped EPROM location that gets loaded into CCR during the reset sequence when the voltage on EA puts the 879XBH in Programming Mode. If PCCB is not programmed using the Auto Configuration Byte Programming Mode, every time the 879XBH is put into Programming Mode the CCR will be loaded with 0FFH (the value of the erased PCCB location).

However, if programming of the CCB and PCCB is done using this programming mode, the PCCB will take on the value programmed into CCB. This means that until the part is erased, programming activities that use the system will employ the bus width and controls selected by the user's CCB.



NOTE:
Tie Port 3 to the value desired to be programmed into CRB, and PCCB.
Make all necessary minimum connections for power, ground and clock.

270090-42

Figure 25. The Auto CCR Programming Mode

RUN-TIME PROGRAMMING

Run-Time Programming of the 879XBH is provided to allow the user complete flexibility in the ways in which the internal EPROM is programmed. That flexibility includes the ability to program just one byte or one word instead of the whole EPROM, and extends to the hardware necessary to program. The only additional requirement of a system is that a programming voltage is applied to V_{PP}. Run-Time Programming is done with $\overline{EA}$ at TTL-high (normal operation - internal/external access).

To Run-Time program, the user writes a byte or word to the location to be programmed. Once this is done, the 879XBH will continue to program that location until another data read from or data write to the EPROM occurs. The user can therefore control the duration of the programming pulse to within a few microseconds. An inteligent algorithm should be implemented in software. It is recommended that the Modified Quick-Pulse Programming™ Algorithm be implemented.

After the programming of a location has started, care must be taken to insure that no program fetches (or pre-fetches) occur from internal memory. This is of no concern if the program is executing from external memory. However, if the program is executing from internal memory when the write occurs, it will be necessary to use the built in "Jump to Self" located at 201AH.

"Jump to Self" is a two byte instruction in the Intel test ROM which can be CALLed after the user has started programming a location by writing to it. A software timer interrupt could then be used to escape from the "Jump to Self" when the proper programming pulse duration has elapsed. Figure 26 is an example of how to program an EPROM location while execution is entirely internal.

Upon entering the PROGRAM routine, the address and data are retrieved from the STACK and a Software Timer is set to expire one programming pulse later. The data is then written to the EPROM location and a CALL to location 201AH is made. Location 201AH is in Intel reserved test ROM, and contains the two byte opcode for a "Jump to Self." The minimum interrupt service routine would remove the 201AH return address from the STACK and return.

```
PROGRAM:
     POP   temp
     POP   address_temp                        ;take parameters from the STACK
     POP   data--temp
     PUSH  temp

     PUSHF                                     ;save current status
     LDB   int_mask ,#enable_swt_only          ;enable only swt interrupts

     LDB   HSO_COMMAND ,#SWTO_ovf              ;load swt command to interrupt
     ADD   HSO_TIME,TIMER1,#program_pulse      ;when program pulse time
                                               ;has elasped

     EI
     ST    data_temp, [address_temp]
     CALL  201AH

     POPF
     RET

SWT_ISR:
     • • •

swt0_expired:
     POP   0
     RET
     • • •
```

Figure 26. Programming the EPROM from Internal Memory Execution

ROM/EPROM PROGRAM LOCK

Protection mechanisms have been provided on the ROM and EPROM versions of the 809XBH to inhibit unauthorized accesses of internal program memory. However, there must always be a way to allow authorized program memory dumps for testing purposes. The following describes 839XBH, 879XBH program lock features and the mode provided for authorized memory dumps.

PROGRAM LOCK FEATURES

Write protection is provided for EPROM parts, while READ protection is provided for both ROM and EPROM parts.

Write protection is enabled by causing the LOC0 bit in the CCR to take the value 0. When WRITE protection is selected, the bus controller will cycle through the write sequence, but will not actually drive data to the EPROM and will not enable V_{PP} to the EPROM. This protects the entire EPROM 2000H-3FFFH from inadvertant or unauthorized programming.

READ protection is selected by causing the LOC1 bit in the CCR to take the value 0. When READ protection is enabled, the bus controller will only perform a data read from the address range 2020H-3FFFH if the slave program counter is in the range 2000H-3FFFH. Note that since the slave PC can be many bytes ahead of the CPU program counter, an instruction that is located after address 3FFAH may not be allowed to access protected memory, even through the instruction is itself protected.

If the bus controller receives a request to perform a READ of protected memory, the READ sequence occurs with indeterminant data being returned to the CPU.

Other enhancements were also made to the 8096BH for program protection. For example, the value of $\overline{EA}$ is latched on reset so that the device cannot be switched from external to internal execution mode at run-time. In addition, if READ protection is selected, an NMI event will cause the device to switch to external only execution mode. Internal execution can only resume by resetting the chip.

AUTHORIZED ACCESS OF PROTECTED MEMORY

To provide a method of dumping the internal ROM/EPROM for testing purposes a "Security Key" mechanism and ROM dump mode have been implemented.

The security key is a 128 bit number, located in internal memory, that must be matched before a ROM dump will occur. The application code contains the security key starting at location 2020H.

The ROM dump mode is entered just like any programming mode ($\overline{EA}$ = 12.75V), except that a special PMODE strapping is used. The PMODE for ROM dump is 6H (0110b).

The ROM dump sequence begins with a security key verification. Users must place at external locations 4020H–402FH the same 16 byte key that resides inside the chip at locations 2020H-202FH. Before doing a ROM dump, the chip checks that the keys match.

After a successful key verification, the chip dumps data to external locations 1000H-11FFH and 4000H-5FFFH. Unspecified data appears at the low addresses. Internal EPROM/ROM is dumped to 4000H-5FFFH beginning with internal address 2000H.

If a security key verification is not successful, the chip will put itself into an endless loop of internal execution.

NOTE:
Substantial effort has been expended to provide an excellent program protection scheme. However, Intel cannot, and does not guarantee that the protection methods that we have devised will prevent unauthorized access.

MODIFIED QUICK-PULSE PROGRAMMING™ ALGORITHM

The Modified Quick-Pulse Programming™ Algorithm calls for each EPROM location to receive 25 separate 100 μs (±5 μs) programming cycles. Verification of correct programming is done after the 25 pulses. If the location verifies correctly, the next location is programmed. If the location fails to verify, the location has failed.

Once all locations are programmed and verified, the entire EPROM is again verified.

Programming of 879XBH parts is done with V_{PP} = 12.75V ±0.25V and V_{CC} = 5.0V ±0.5V.

SIGNATURE WORD

The 879XBH contains a signature word at location 2070H. The word can be accessed in the slave mode by executing a word dump command.

Table 5. 8X9XBH Signature Word

Device	Signature Word
879XBH	896FH
839XBH	896EH
809XBH	Undefined

Erasing the 879XBH EPROM

Initially, and after each erasure, all bits of the 879XBH are in the "1" state. Data is introduced by selectively programming "0s" into the desired bit locations. Although only "0s" will be programmed, both "1s" and "0s" can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The erasure characteristics of the 879XBH are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Constant exposure to room level fluorescent lighting could erase the typical 879XBH in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 879XBH is to be exposed to light for extended periods of time, opaque labels must be placed over the EPROM's window to prevent unintentional erasure.

The recommended erasure procedure for the 879XBH is exposure to shortwave ultraviolet light which has a wavelength of 2537 Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15 Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 μW/cm^2 power rating. The 879XBH should be placed within 1 inch of the lamp tubes during erasure. The maximum integrated dose an 879XBH can be exposed to without damage is 7258 Wsec/cm^2 (1 week @ 12000 μW/cm^2). Exposure of the 879XBH to high intensity UV light for long periods may cause permanent damage.

POWER SUPPLY SEQUENCE WHILE PROGRAMMING

For any 879XBH that is in any programming mode, high voltages must be applied to the device. To avoid damaging the parts, the following rules must not be violated.

RULE #1—V_{PP} must not have a low impedance path to ground when V_{CC} is above 4.5V.

RULE #2—V_{CC} must be above 4.5V before V_{PP} can be higher than 5.0V.

RULE #3—V_{PP} must be within 1V of V_{CC} while V_{CC} is below 4.5V.

RULE #4—All voltages must be within tolerance and the oscillator stable before $\overline{\text{RESET}}$ rises.

RULE #5—$\overline{\text{EA}}$ must be brought high to place the part in programming mode before V_{PP} is brought high.

To adhere to these rules, the following power up and power down sequences can be followed.

POWER UP

$\overline{\text{RESET}}$ = 0;
CLOCK ON; if using an external clock
; instead of an oscillator
$V_{CC} = V_{PP} = V_{EA}$ = 5V;
PALE = $\overline{\text{PROG}}$ = PORT 34 = V_{IH};*
SID AND PMODE VALID;
$\overline{\text{EA}}$ = 12.75V;
V_{PP} = 12.75V;
WAIT; wait for supplies and clock to
; settle
$\overline{\text{RESET}}$ = 5V;
WAIT Tshll; See Data Sheet
BEGIN;

POWER DOWN

$\overline{\text{RESET}}$ = 0;
V_{PP} = 5V;
$\overline{\text{EA}}$ = 5V;
PALE = $\overline{\text{PROG}}$ = SID = PMODE = PORT34 = 0V;
$V_{CC} = V_{PP} = V_{EA}$ = 0V;
CLOCK OFF;

*V_{IH} = Logical "1", 2.4V Minimum

One final note on power up, power down. The maximum limit on V_{PP} must never be violated, even for an instant. Therefore, an RC rise to the desired V_{PP} is recommended. V_{PP} is also sensitive to instantaneous voltage steps. This also can be avoided by using an RC ramp on V_{PP}.

EPROM SPECIFICATIONS

A.C. EPROM PROGRAMMING CHARACTERISTICS

Operating Conditions: Load Capacitance = 150 pF, T_A = 25°C ±5°C, V_{CC}, V_{PD}, V_{REF} = 5.0V ± 0.5V, V_{SS}, AGND = 0V, V_{PP} = 12.75V ± 0.25V, $\overline{EA}$ = 11V ± 2.0V, f_{OSC} = 6.0 MHz

Symbol	Parameter	Min	Max	Units
T_{AVLL}	ADDRESS/COMMAND Valid to PALE Low	0		T_{OSC}
T_{LLAX}	ADDRESS/COMMAND Hold After PALE Low	80		T_{OSC}
T_{DVPL}	Output Data Setup Before $\overline{PROG}$ Low	0		T_{OSC}
T_{PLDX}	Data Hold After $\overline{PROG}$ Falling	80		T_{OSC}
T_{LLLH}	PALE Pulse Width	180		T_{OSC}
T_{PLPH}	$\overline{PROG}$ Pulse Width	250 T_{OSC}	100 μS + 144 T_{OSC}	
T_{LHPL}	PALE High to $\overline{PROG}$ Low	250		T_{OSC}
T_{PHLL}	$\overline{PROG}$ High to Next PALE Low	600		T_{OSC}
T_{PHDX}	Data Hold After $\overline{PROG}$ High	30		T_{OSC}
T_{PHVV}	$\overline{PROG}$ High to PVER/$\overline{PDO}$ Valid	500		T_{OSC}
T_{LLVH}	PALE Low to PVER/$\overline{PDO}$ High	100		T_{OSC}
T_{PLDV}	$\overline{PROG}$ Low to VERIFICATION/DUMP Data Valid	100		T_{OSC}
T_{SHLL}	RESET High to First PALE Low (not shown)	2000		T_{OSC}

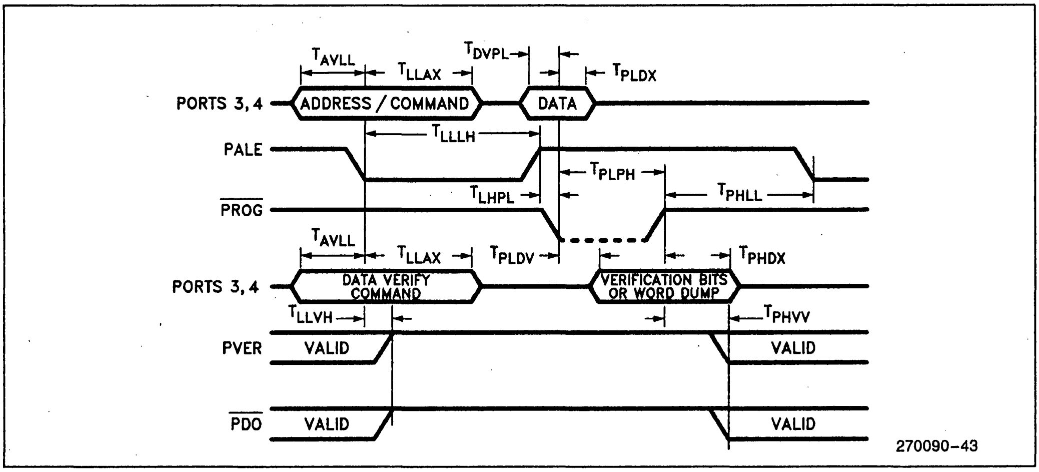
NOTE:
Run-time programming is done with F_{OSC} = 6.0 MHz to 12.0 MHz, V_{CC}, V_{PD}, V_{REF} = 5V ± 0.5V, T_A = 25°C to ±5°C and V_{PP} = 12.75V ± 0.25V. For run-time programming over a full operating range, contact the factory. All windowed devices should be covered after programming.

D.C. EPROM PROGRAMMING CHARACTERISTICS

Symbol	Parameter	Min	Max	Units
I_{PP}	V_{PP} Supply Current (Whenever Programming)		100	mA
V_{PP}	Programming Supply Voltage	12.75 ±0.25		V
V_{EA}	$\overline{EA}$ Programming Voltage	11 ±2.0		V

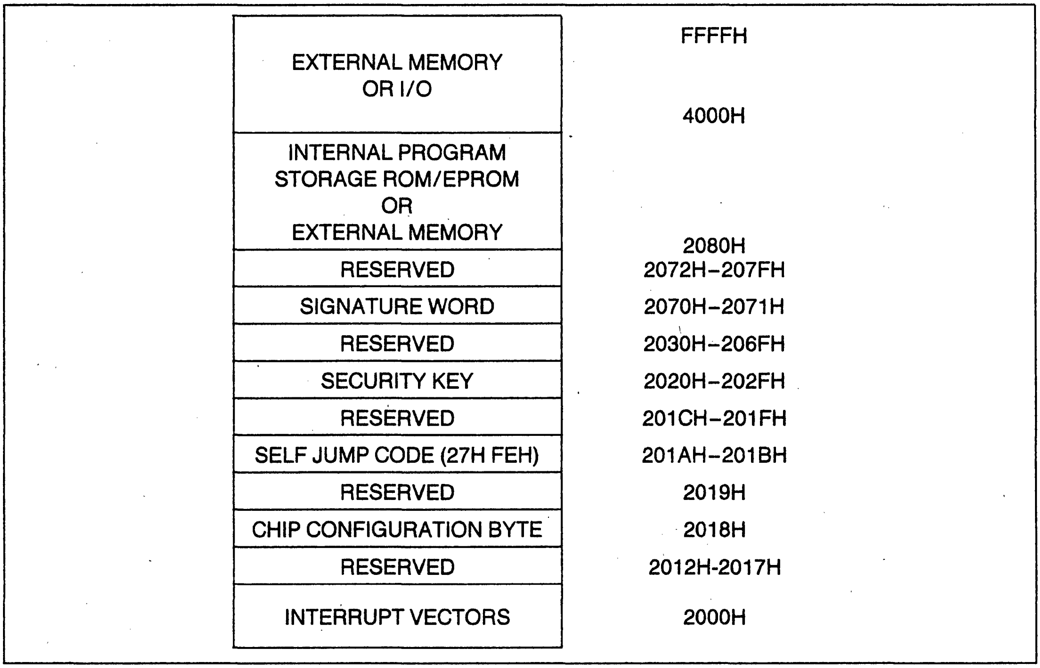
NOTE:
V_{PP} must be within 1V of V_{CC} while V_{CC} < 4.5V. V_{PP} must not have a low impedance path to ground or V_{SS} while V_{CC} > 4.5V.

WAVEFORM—EPROM PROGRAMMING



Reserved location warning: Intel Reserved addresses can not be used by applications which use 8X9XBH internal ROM/EPROM. The data read from a reserved location is not guaranteed, and a write to any reserved location could cause unpredictable results. When attempting to program Intel Reserved addresses, the data must be 0FFFFH to ensure a harmless result. A memory map indicating reserved locations on the 8X9XBH is shown in Figure 27.

Intel Reserved locations, when mapped to external memory, must be filled with 0FFFFH to ensure compatibility with future parts.



	FFFFH
EXTERNAL MEMORY OR I/O	4000H
INTERNAL PROGRAM STORAGE ROM/EPROM OR EXTERNAL MEMORY	2080H
RESERVED	2072H–207FH
SIGNATURE WORD	2070H–2071H
RESERVED	2030H–206FH
SECURITY KEY	2020H–202FH
RESERVED	201CH–201FH
SELF JUMP CODE (27H FEH)	201AH–201BH
RESERVED	2019H
CHIP CONFIGURATION BYTE	2018H
RESERVED	2012H-2017H
INTERRUPT VECTORS	2000H

Figure 27. Reserved Locations

MCS®-96
809XBH-10
ADVANCED 16-BIT MICROCONTROLLER WITH 8- OR 16-BIT EXTERNAL BUS

- **232 Byte Register File**
- **Register-to-Register Architecture**
- **A/D Converter with S/H**
- **Five 8-Bit I/O Ports**
- **20 Interrupt Sources**
- **Pulse-Width Modulated Output**
- **High Speed I/O Subsystem**
- **Full Duplex Serial Port**
- **Dedicated Baud Rate Generator**
- **Hardware 16 x 16 Multiply, 32/16 Divide**
- **16-Bit Watchdog Timer**
- **Four 16-Bit Software Timers**
- **Two 16-Bit Counter/Timers**
- **Available in 48-Pin Ceramic DIP and 68 Pin Ceramic PGA**
  (See Packaging Specification Order #231369)

The MCS®-96 family of 16-bit microcontrollers consists of many members, all of which are designed for high-speed control functions. The MCS-96 family members operating with 10 MHz clocks are described in this data sheet.

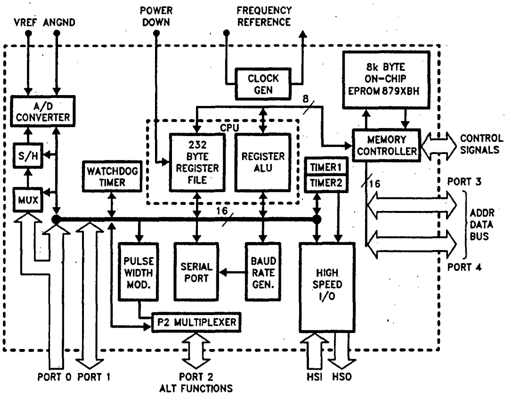
The CPU supports bit, byte, and word operations. Thirty-two bit double-words are supported for a subset of the instruction set. Hardware multiplication and division, along with three-operand instructions and flexible addressing modes provide for efficient use of the chip's 232 bytes of general purpose registers.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at preset times. The high-speed output unit can simultaneously perform software timer functions. Up to four 16-bit software timers can be in operation at once.

The on-chip A/D converter includes a Sample and Hold, and converts up to 8 multiplexed analog input channels to digital values. This feature is only available on the 8X95BHs and 8X97BHs, with the 8X95BHs having 4 multiplexed analog inputs.

Also provided on-chip are a serial port, a Watchdog Timer, and a pulse-width modulated output signal.

The specifications in this datasheet supplement the information contained in the Embedded Controller Handbook (Order Number 210918) and pertain only to suffix -10 of the MCS-96 family.



270278-1

Figure 1. MCS®-96 Block Diagram

PACKAGING

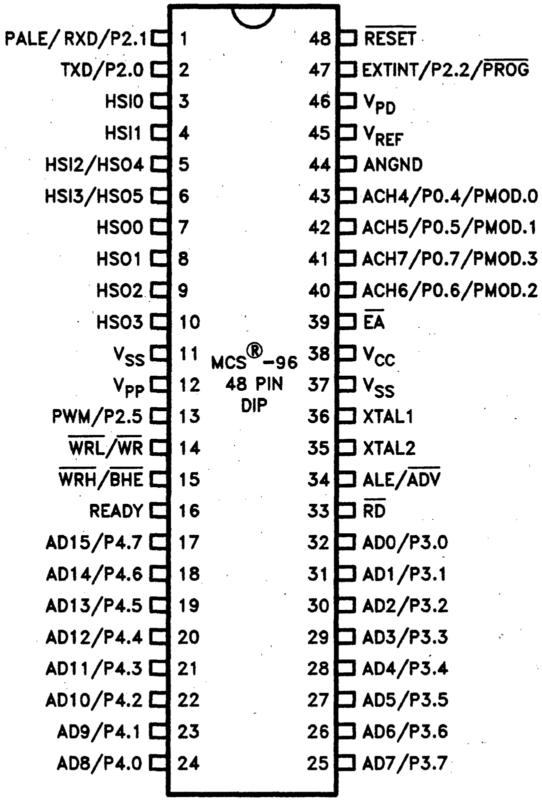
The 809XBH-10 is available in 48-pin and 68-pin packages with A/D. The MCS-96 numbering system is shown in Figure 2. Figures 3–5 show the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin version is in a Pin Grid Array (PGA).

		With A/D
ROMless	48 Pin	C8095BH-10 - Ceramic DIP
	68 Pin	A8097BH-10 - Ceramic PGA

Figure 2. The MCS®-96 Family Nomenclature

PGA	Description	PGA	Description	PGA	Description
1	ACH7/P0.7/PMOD.3	24	AD6/P3.6	47	P1.6
2	ACH6/P0.6/PMOD.2	25	AD7/P3.7	48	P1.5
3	ACH2/P0.2	26	AD8/P4.0	49	HSO.1
4	ACH0/P0.0	27	AD9/P4.1	50	HSO.0
5	ACH1/P0.1	28	AD10/P4.2	51	HSO.5/HSI.3
6	ACH3/P0.3	29	AD11/P4.3	52	HSO.4/HSI.2
7	NMI	30	AD12/P4.4	53	HSI.1
8	$\overline{\text{EA}}$	31	AD13/P4.5	54	HSI.0
9	VCC	32	AD14/P4.6	55	P1.4
10	VSS	33	AD15/P4.7	56	P1.3
11	XTAL1	34	T2CLK/P2.3	57	P1.2
12	XTAL2	35	READY	58	P1.1
13	CLKOUT	36	T2RST/P2.4	59	P1.0
14	BUSWIDTH	37	$\overline{\text{BHE}}/\overline{\text{WRH}}$	60	TXD/P2.0
15	INST	38	$\overline{\text{WR}}/\overline{\text{WRL}}$	61	RXD/P2.1/PALE
16	ALE/$\overline{\text{ADV}}$	39	PWM/P2.5	62	$\overline{\text{RESET}}$
17	$\overline{\text{RD}}$	40	P2.7	63	EXTINT/P2.2/$\overline{\text{PROG}}$
18	AD0/P3.0	41	VPP	64	VPD
19	AD1/P3.1	42	VSS	65	VREF
20	AD2/P3.2	43	HSO.3	66	ANGND
21	AD3/P3.3	44	HSO.2	67	ACH4/P0.4/PMOD.0
22	AD4/P3.4	45	P2.6	68	ACH5/P0.5/PMOD.1
23	AD5/P3.5	46	P1.7		

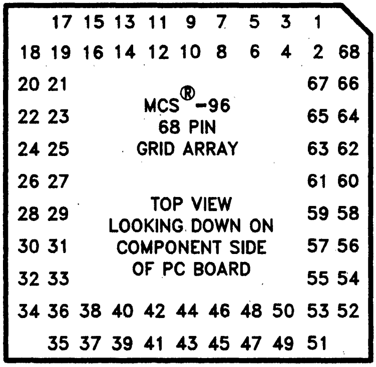
Figure 3. PGA Function Pinouts



270278-2

Figure 4. 48-Pin Package

Pins Facing Down



270278-4

Figure 5. 68-Pin Package (Pin Grid Array—Top View)

PIN DESCRIPTIONS

Symbol	Name and Function
V_{CC}	Main supply voltage (5V).
V_{SS}	Digital circuit ground (0V). There are two V_{SS} pins, both of which must be connected.
V_{PD}	RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e. V_{CC} drops to zero), if $\overline{\text{RESET}}$ is activated before V_{CC} drops below spec and V_{PD} continues to be held within spec., the top 16 bytes in the Register File will retain their contents. $\overline{\text{RESET}}$ must be held low during the Power Down and should not be brought high until V_{CC} is within spec and the oscillator has stabilized.
V_{REF}	Reference voltage for the A/D converter (5V). V_{REF} is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0. Must be connected for A/D and Port 0 to function.
ANGND	Reference ground for the A/D converter. Must be held at nominally the same potential as V_{SS}.
V_{PP}	This pin is V_{BB} on 8X9X-90 parts. V_{PP} must be unconnected on 8X9XBH-10 parts.
XTAL1	Input of the oscillator inverter and of the internal clock generator.
XTAL2	Output of the oscillator inverter.
CLKOUT	Output of the internal clock generator. The frequency of CLKOUT is ⅓ the oscillator frequency. It has a 33% duty cycle.
$\overline{\text{RESET}}$	Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to location 2080H is executed. Input high for normal operation. $\overline{\text{RESET}}$ has an internal pullup.
BUSWIDTH	Input for bus width selection. If CCR Bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is high, a 16-bit bus cycle occurs. If BUSWIDTH is low, an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. This pin is the $\overline{\text{TEST}}$ pin on 8X9X-90 parts. Systems with $\overline{\text{TEST}}$ tied to V_{CC} do not need to change. If this pin is left unconnected, it will rise to V_{CC}.
NMI	A positive transition causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.
INST	Output high during an external memory read indicates the read is an instruction fetch. INST is valid throughout the bus cycle.
$\overline{\text{EA}}$	Input for memory select (External Access). $\overline{\text{EA}}$ equal to a TTL-high causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM. $\overline{\text{EA}}$ equal to a TTL-low causes accesses to these locations to be directed to off-chip memory. $\overline{\text{EA}}$ = +12.5V causes execution to begin in the Programming Mode. $\overline{\text{EA}}$ has an internal pulldown, so it goes low unless driven otherwise. $\overline{\text{EA}}$ is latched at reset.
ALE/$\overline{\text{ADV}}$	Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is $\overline{\text{ADV}}$, it goes inactive high at the end of the bus cycle. $\overline{\text{ADV}}$ can be used as a chip select for external memory. ALE/$\overline{\text{ADV}}$ is activated only during external memory accesses.
$\overline{\text{RD}}$	Read signal output to external memory. $\overline{\text{RD}}$ is activated only during external memory reads.
$\overline{\text{WR}}$/$\overline{\text{WRL}}$	Write and Write Low output to external memory, as selected by the CCR. $\overline{\text{WR}}$ will go low for every external write, while $\overline{\text{WRL}}$ will go low only for external writes where an even byte is being written. $\overline{\text{WR}}$/$\overline{\text{WRL}}$ is activated only during external memory writes.
$\overline{\text{BHE}}$/$\overline{\text{WRH}}$	Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{\text{BHE}}$ low selects the bank of memory that is connected to the high byte of the data bus. A0 low selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 low, $\overline{\text{BHE}}$ high), to the high byte only (A0 high, BHE low), or both bytes (A0 low, $\overline{\text{BHE}}$ low). If the $\overline{\text{WRH}}$ function is selected, the pin will go low if the bus cycle is writing to an odd memory location. $\overline{\text{BHE}}$/$\overline{\text{WRH}}$ is activated only during external memory writes.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
READY	Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the memory controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. The bus cycle can be lengthened by up to 1 μs. When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle held not ready is available through configuration of CCR. READY has a weak internal pullup, so it goes high unless externally pulled low.
HSI	Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as inputs by EPROM parts in Programming Mode.
HSO	Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.
Port 0	8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins are also a mode input to parts in the Programming Mode.
Port 1	8-bit quasi-bidirectional I/O port.
Port 2	8-bit multi-functional port. Six of its pins are shared with other functions, the remaining 2 are quasi-bidirectional. These pins are also used to input and output control signals on parts in Programming Mode.
Ports 3 and 4	8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups. Ports 3 and 4 are also used as a command, address and data path by parts operating in the Programming Mode.

ELECTRICAL CHARACTERISTICS ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	0°C to +70°C
Storage Temperature	−40°C to +150°C
Voltage from $\overline{EA}$ to V_{SS} or ANGND	13.0V
Voltage from Any Other Pin to V_{SS} or ANGND	−0.3V to +7.0V
Average Output Current from Any Pin	10 mA
Power Dissipation	1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	0	+70	C
V_{CC}	Digital Supply Voltage	4.75	5.25	V
V_{REF}	Analog Supply Voltage	4.75	5.25	V
f_{OSC}	Oscillator Frequency	6.0	10.0	MHz
V_{PD}	Power-Down Supply Voltage	4.75	5.25	V

NOTE: ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS (Test Conditions: V_{CC}, V_{REF}, V_{PD}, V_{PP}, V_{EA} = 5.0V ±0.25V; F_{OSC} = 6.0 to 10.0 MHz; T_A = 0°C to 70°C; V_{SS}, ANGND = 0V)

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{CC}	V_{CC} Supply Current (0°C ≤ T_A ≤ 70°C		200	mA	All Outputs Disconnected.
I_{CC1}	V_{CC} Supply Current (T_A = 70°C)		160	mA	
I_{PD}	V_{PD} Supply Current		1	mA	Normal operation and Power-Down.
I_{REF}	V_{REF} Supply Current		5	mA	(Note 4)
V_{IL}	Input Low Voltage (Except $\overline{RESET}$)	−0.3	+0.8	V	
V_{IL1}	Input Low Voltage, $\overline{RESET}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage (Except $\overline{RESET}$, NMI, XTAL1)	2.0	V_{CC} +0.5	V	
V_{IH1}	Input High Voltage, $\overline{RESET}$ Rising	2.4	V_{CC} +0.5	V	
V_{IH2}	Input High Voltage, $\overline{RESET}$ Falling	2.1	V_{CC} +0.5	V	
V_{IH3}	Input High Voltage, NMI, XTAL1	2.2	V_{CC} +0.5	V	
I_{LI}	Input Leakage Current to each pin of HSI, P3, P4, and to P2.1.		±10	μA	V_{in} = 0 to V_{CC}
I_{LI1}	D.C. Input Leakage Current to each pin of P0		+10	μA	V_{in} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{EA}$		100	μA	V_{IH} = 2.4V
I_{IL}	Input Low Current to each pin of P1, and to P2.6, P2.7.		−100	μA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{RESET}$	−0.3	−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4		−50	μA	V_{IL} = 0.45V
I_{IL3}	Input Low Current to READY		−160	μA	V_{IL} = 0.45V
I_{IL4}	Input Low Current to BUSWIDTH		−50	μA	V_{IL} = 0.45V (Note 4)
V_{OL}	Output Low Voltage on Quasi-Bidirectional port pins		0.45	V	I_{OL} = 0.36 mA (Notes 1, 5)
V_{OL2}	Output Low Voltage on Standard Output pins, $\overline{RESET}$ and Bus/Control Pins		0.45	V	I_{OL} = 2.0 mA (Note 1)

D.C. CHARACTERISTICS (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{OH}	Output High Voltage on Quasi-Bidirectional pins	2.4		V	$I_{OH} = -20\ \mu A$ (Note 1)
V_{OH1}	Output High Voltage on Standard Output pins and Bus/Control pins	2.4		V	$I_{OH} = -200\ \mu A$ (Note 1)
I_{OH3}	Output High Current on $\overline{RESET}$	−50		μA	$V_{OH} = 2.4V$ (Note 4)
C_S	Pin Capacitance (Any Pin to V_{SS})		10	pF	fTEST = 1.0 MHz

NOTES:
1. Quasi-bidirectional pins include those on P1, for P2.6 and P2.7. Standard Output Pins include RXD (Mode 0 only), TXD, PWM, and HSO pins. Note 4 applies to RXD in Mode 0. Bus/Control pins include CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, INST and AD0–15.
2. Maximum current per pin must be externally limited to the following values if V_{OL} is held above 0.45V.
   I_{OL} on quasi-bidirectional pins: 4.0 mA
   I_{OL} on standard output pins and $\overline{RESET}$: 8.0 mA
   I_{OL} on Bus/Control pins: 2.0 mA
3. During normal (non-transient) operation the following limits apply:
   Total I_{OL} on Port 1 must not exceed 4.0 mA.
   Total I_{OL} on P2.0, P2.6, $\overline{RESET}$ and all HSO pins must not exceed 17.0 mA.
   Total I_{OL} on P2.5 and P2.7 must not exceed 4.0 mA.
4. These values are not tested in production, and are based on theoretical estimates and/or laboratory tests.
5. I_{OL} is typically greater than 0.4 mA, but is tested to 0.36 mA.

A.C. CHARACTERISTICS $V_{CC}, V_{PD} = 5.0V \pm 0.25V$; $T_A = 0°C$ to $70°C$; $f_{OSC} = 6.0$ MHz to 10.0 MHz

Test Conditions: Load Capacitance on Output Pins = 80 pF
Oscillator Frequency = 10 MHz

TIMING REQUIREMENTS (Other system components must meet these specs.)

Symbol	Parameter	Min	Max	Units
T_{CLYX}(4)	READY Hold after CLKOUT Edge	0(1)		ns
T_{LLYV}	End of ALE/$\overline{ADV}$ to READY Valid		2Tosc−90	ns
T_{LLYH}	End of ALE/$\overline{ADV}$ to READY High	2Tosc+40	4Tosc−50	ns
T_{YLYH}	Non-Ready Time		1000	ns
T_{AVDV}(6)	Address Valid to Input Data Valid		5Tosc−130	ns
T_{RLDV}	$\overline{RD}$ Active to Input Data Valid		3Tosc−100	ns
T_{RHDX}	Data Hold after $\overline{RD}$ Inactive	0		ns
T_{RHDZ}	$\overline{RD}$ Inactive to Input Data Float	0	Tosc−20	ns
T_{AVGV}(4)(6)	Address Valid to BUSWIDTH Valid		2 Tosc −135	ns
T_{LLGX}(4)	BUSWIDTH Hold after ALE/$\overline{ADV}$ Low	Tosc +10		ns
T_{LLGV}(4)	ALE/$\overline{ADV}$ Low to BUSWIDTH Valid		Tosc −70	ns

NOTES:
1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at 2Tosc+60 (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
4. Pins not bonded out on 48-pin parts.
6. The term "Address Valid" applies to AD0–15, $\overline{BHE}$ and INST.

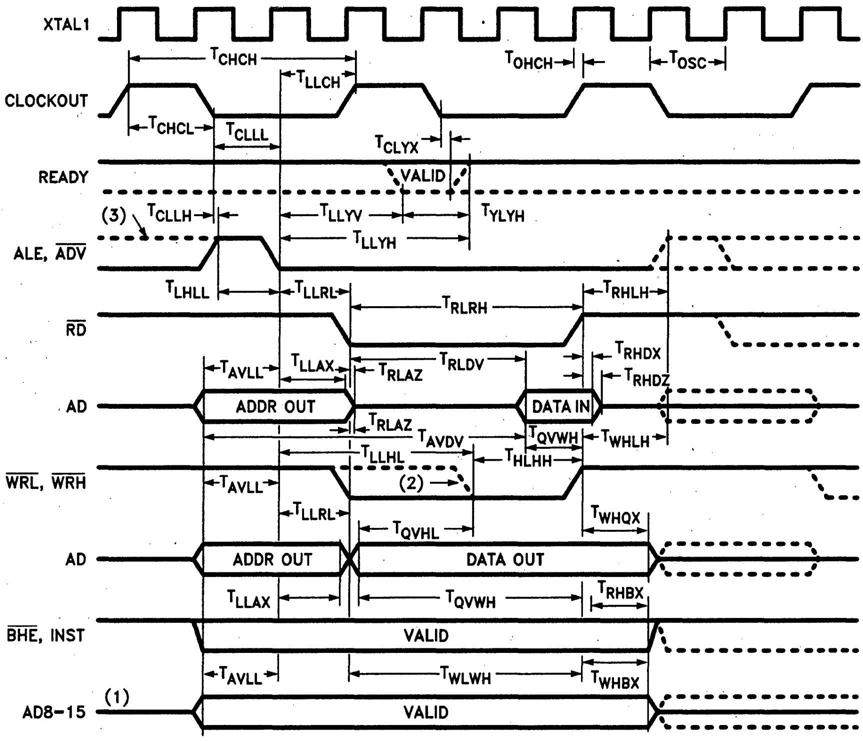
A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES (MCS-96 parts meet these specs.)

Symbol	Parameter	Min	Max	Units
F_{XTAL}	Oscillator Frequency	6.0	10.0	MHz
T_{OSC}	Oscillator Period	100	166	ns
T_{OHCH}	XTAL1 Rising Edge to Clockout Rising Edge	0(4)	120(4)	ns
T_{CHCH}(4)	CLKOUT Period(3)	3Tosc(3)	3Tosc(3)	ns
T_{CHCL}(4)	CLKOUT High Time	Tosc − 20	Tosc + 25	ns
T_{CLLH}(4)	CLKOUT Low to ALE High	−10	20	ns
T_{LLCH}(4)	ALE/$\overline{ADV}$ Low to CLKOUT High	Tosc − 20	Tosc + 40	ns
T_{LHLL}	ALE/$\overline{ADV}$ High Time	Tosc − 35	Tosc + 15(5)	ns
T_{AVLL}(6)	Address Setup to End of ALE/$\overline{ADV}$	Tosc − 65		ns
T_{RLAZ}(7)	$\overline{RD}$ or $\overline{WR}$ Low to Address Float		25(8)	ns
T_{LLRL}	End of ALE/$\overline{ADV}$ to $\overline{RD}$ or $\overline{WR}$ Active	Tosc − 20		ns
T_{LLAX}(7)	Address Hold after End of ALE/$\overline{ADV}$	Tosc − 20		ns
T_{WLWH}	$\overline{WR}$ Pulse Width	3Tosc − 35		ns
T_{QVWH}	Output Data Valid to End of $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$	3Tosc − 65		ns
T_{WHQX}	Output Data Hold after $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$	Tosc − 30		ns
T_{WHLH}	End of $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$ to ALE/$\overline{ADV}$ High	Tosc − 55		ns
T_{RLRH}	$\overline{RD}$ Pulse Width	3Tosc − 30		ns
T_{RHLH}	End of $\overline{RD}$ to ALE/$\overline{ADV}$ High	Tosc − 15		ns
T_{CLLL}(4)	CLOCKOUT Low to ALE/$\overline{ADV}$ Low	Tosc − 40(8)	Tosc + 20(8)	ns
T_{RHBX}(4)	$\overline{RD}$ High to INST, $\overline{BHE}$, AD8-15 Inactive	Tosc	Tosc + 30	ns
T_{WHBX}(4)	$\overline{WR}$ High to INST, $\overline{BHE}$, AD8-15 Inactive	Tosc − 45	Tosc + 30	ns
T_{HLHH}	$\overline{WRL}$, $\overline{WRH}$ Low to $\overline{WRL}$, $\overline{WRH}$ High	2Tosc − 35	2Tosc + 20	ns
T_{LLHL}	ALE/$\overline{ADV}$ Low to $\overline{WRL}$, $\overline{WRH}$ Low	2Tosc − 20	2Tosc + 55	ns
T_{QVHL}	Output Data Valid to $\overline{WRL}$, $\overline{WRH}$ Low	Tosc − 60		ns

NOTES:
2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ± 10 ns if Tosc is constant and the rise and fall times on XTAL1 are less than 10 ns.
4. Pins not bonded out on 48-pin parts.
5. Max spec applies only to ALE. Min spec applies to both ALE and $\overline{ADV}$.
6. The term "Address Valid" applies to AD0–15, $\overline{BHE}$ and INST.
7. The term " Address" in this definition applies to AD0–7 for 8-bit cycles, and AD0–15 for 16-bit cycles.
8. Typical value.

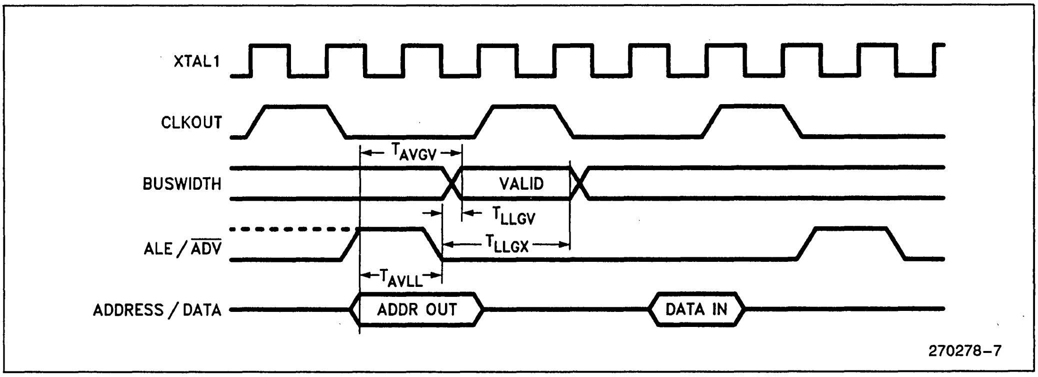
WAVEFORM



270278–6

NOTES:
(1) 8-bit bus only.
(2) 8-bit bus; or when write strobe mode selected.
(3) When $\overline{ADV}$ selected.

WAVEFORM—BUSWIDTH PIN



A.C. CHARACTERISTICS—SERIAL PORT—SHIFT REGISTER MODE

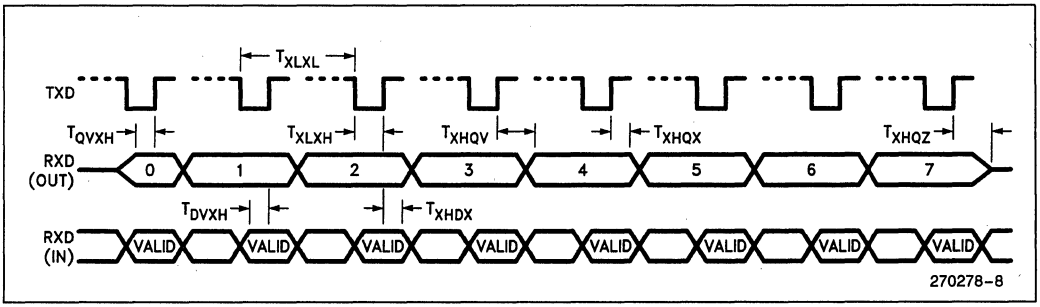
SERIAL PORT TIMING—SHIFT REGISTER MODE

Test Conditions: T_A = 0°C to +70°C; V_{CC} = 5V ±5%; V_{SS} = 0V; Load Capacitance = 80 pF

Symbol	Parameter	Min	Max	Units
T_{XLXL}	Serial Port Clock Period	$8T_{OSC}$		ns
T_{XLXH}	Serial Port Clock Falling Edge to Rising Edge	$4T_{OSC} - 50$	$4T_{OSC} + 50$	ns
T_{QVXH}	Output Data Setup to Clock Rising Edge	$3T_{OSC}$		ns
T_{XHQX}	Output Data Hold After Clock Rising Edge	$2T_{OSC} - 50$		ns
T_{XHQV}	Next Output Data Valid After Clock Rising Edge		$2T_{OSC} + 50$	ns
T_{DVXH}	Input Data Setup to Clock Rising Edge	$2T_{OSC} + 210$		ns
T_{XHDX}	Input Data Hold After Clock Rising Edge	0		ns
T_{XHQZ}	Last Clock Rising to Output Float		$4T_{OSC} + 100$	ns

WAVEFORM—SERIAL PORT—SHIFT REGISTER MODE

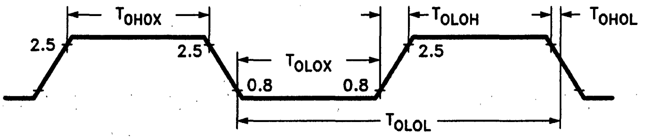
SERIAL PORT WAVEFORM—SHIFT REGISTER MODE



EXTERNAL CLOCK DRIVE

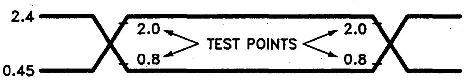
Symbol	Parameter	Min	Max	Units
$1/T_{OLOL}$	Oscillator Frequency	6	10	MHz
T_{OHOX}	High Time	40		ns
T_{OLOX}	Low Time	40		ns
T_{OLOH}	Rise Time		10	ns
T_{OHOL}	Fall Time		10	ns

EXTERNAL CLOCK DRIVE WAVEFORMS



270278–9

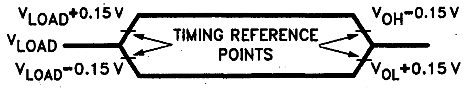
A.C. TESTING INPUT, OUTPUT WAVEFORM



270278–10

A.C. Testing inputs are driven at 2.4V for a Logic "1" and 0.045V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

FLOAT WAVEFORM



270278–11

For Timing Purposes a Port Pin is no Longer Floating when a 100 mV change from Load Voltage Occurs, and Begins to Float when a 100 mV change from the Loaded V_{OH}/V_{OL} Level occurs $I_{OL}/I_{OH} \geq \pm 15$ mA.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097BH-10; 8095BH-10.

The absolute conversion accuracy is dependent on the accuracy of V_{REF}. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at V_{REF} = 5.120V.

OPERATING CONDITIONS

V_{CC}, V_{PD}, V_{REF}	5.0V ±0.25V
V_{SS}, ANGND	0.0V
T_A	0°C to 70°C
F_{OSC} 8X9XBH-10	6.0 to 10.0 MHz
Test Conditions:	
V_{REF}	5.120V
V_{CC}	5.0V

Parameter	Typical*(1)	Minimum	Maximum	Units**	Notes
Resolution		256 8	256 8	Levels Bits	
Absolute Error		0	±1	LSBs	
Full Scale Error	−0.5 ±0.5			LSBs	
Zero Offset Error	±0.5			LSBs	
Non-Linearity		0	±1	LSBs	
Differential Non-Linearity		0	±0.5	LSBs	
Channel-to-Channel Matching		0	±0.1	LSBs	
Repeatability	±0.25			LSBs	1
Temperature Coefficients: Offset Full Scale Differential Non-Linearity	 0.003 0.003 0.003			 LSB/°C LSB/°C LSB/°C	 1 1 1
Off Isolation		−60		dB	1, 2, 4
Feedthrough	−60			dB	1, 2
V_{CC} Power Supply Rejection	−60			dB	1, 2
Input Resistance		1K	5K	Ω	1
D.C. Input Leakage		0	3	μA	
Sample Delay		$3T_{OSC} - 50$	$3T_{OSC} + 50$	ns	1, 3
Sample Time		$12T_{OSC} - 50$	$12T_{OSC} + 50$	ns	1
Sample Capacitance			2	pF	1

NOTES:

* These values are expected for most parts at 25°C.

** An "LSB", as used here, is defined in the glossary which follows and has a value of approximately 20 mV.

1. These values are not tested in production and are based on theoretical estimates and/or laboratory tests.
2. DC to 100 KHz.
3. For starting the A/D with an HSO Command.
4. Multiplexer Break-Before-Make Guaranteed.

A/D GLOSSARY OF TERMS

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An actual characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversions under the same conditions.

BREAK-BEFORE-MAKE—The property of a multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g. the converter will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE CENTER—The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION—The point at which the converter changes from an output code of Q, to a code of Q + 1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK—See "Off-Isolation".

D.C. INPUT LEAKAGE—Leakage current to ground from an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D Converter after the sample window closes.

FULL SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at $V_{IN} = 0.5$ LSB, its last code transition at $V_{IN} = (V_{REF} - 1.5$ LSB) and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB—Least Significant Bit: The voltage corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For an 8-bit converter with a reference voltage of 5.12V, one LSB is 20 mV. Note that this is different than digital LSBs, since an uncertainty of two LSB, when referring to an A/D converter, equals 40 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 80 mV.)

MONOTONIC—The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES—For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY—The maximum deviation of code transitions of the terminal based characteristic from the corresponding code transitions of the ideal characteristic.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY—The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY—The variation in the sample delay.

SAMPLE TIME—The time that the sample window is open.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION—An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An actual characteristic which has been rotated and translated to remove zero offset and full scale error.

V_{CC} REJECTION—Attenuation of noise on the V_{CC} line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

FUNCTIONAL DEVIATIONS

Functional deviations from the 809XBH on the 809XBH-10.

CPU Section

1. Indexed, 3 Operand Multiply—The displacement portion of an indexed, three operand multiply may not be in the range of 200H thru 17FFH inclusive, on 8X9XBH-10 parts. If you must use these displacements, do an indexed, two operand multiply and a move if necessary.
2. JBS, JBC—The JBS and JBC instructions should not be used directly on Port 2.1 or Port 0. If it is necessary to test Port 2.1 or Port 0, the entire port should be loaded into a temporary register, and the bit tested there.
3. STicky Flag—The STicky flag is not affected when a shift by 0 is executed on 8X9XBH-10 parts.
4. Auto Increment Indirect, 3 Word Multiply—The use of these instructions may result in the loss of Special Function Register contents. Use an LD instruction and a 2 Word Multiply with Auto Indirect Addressing Mode.
5. High Current on Power Up. I_{CC} may be up to 500 mA before the oscillator starts.

PRELIMINARY

MCS®-96
809X-90, 839X-90

- 839X: an 809X with 8 Kbytes of On-Chip ROM
- High Speed Pulse I/O
- 10-Bit A/D Converter
- 6.25 µs 16 x 16 Multiply
- 6.25 µs 32/16 Divide
- 8 Interrupt Sources
- Pulse-Width Modulated Output
- 232 Byte Register File
- Memory-to-Memory Architecture
- Full Duplex Serial Port
- Five 8-Bit I/O Ports
- Watchdog Timer
- Four 16-Bit Software Timers

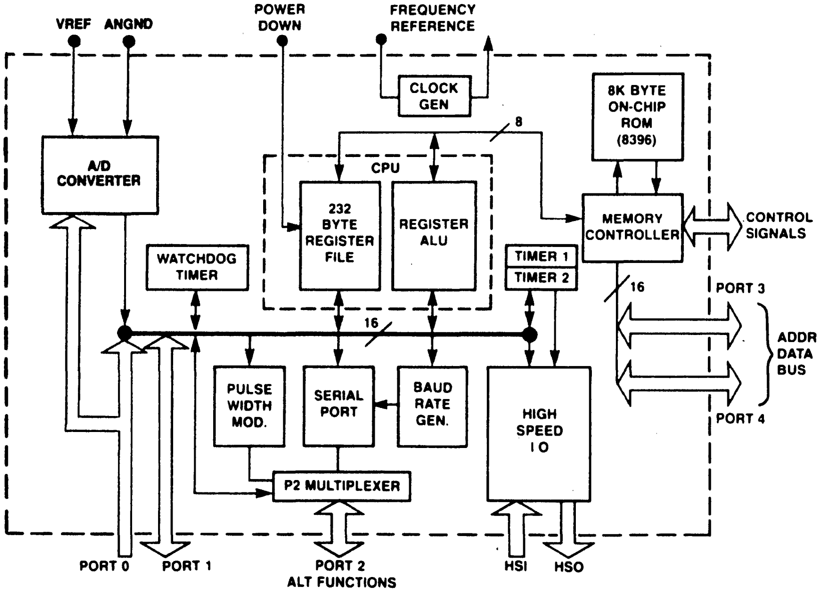
The MCS®-96 family of 16-bit microcontrollers consists of many members, all of which are designed for high-speed control functions. Members with the "−90" suffix are described in this data sheet.

The CPU supports bit, byte, and word operations. 32-bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096 can do a 16-bit addition in 1.0 µs and a 16 x 16-bit multiply or 32/16-bit divide in 6.25 µs. Instruction execution times average 1 to 2 µs in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at present times. The high-speed output unit can simultaneously perform timer functions. Up to four such 16-bit Software Timers can be in operation at once.

An on-chip A/D Converter converts up to 4 (in the 48-pin version) or 8 (in the 68-pin version) analog input channels to 10-bit digital values. This feature is only available on the 8095-90/8395-90 and 8097-90/8397-90.

Also provided on-chip are a serial port, a watchdog timer, and a pulse-width modulated output signal.



270014–1

Figure 1. Block Diagram

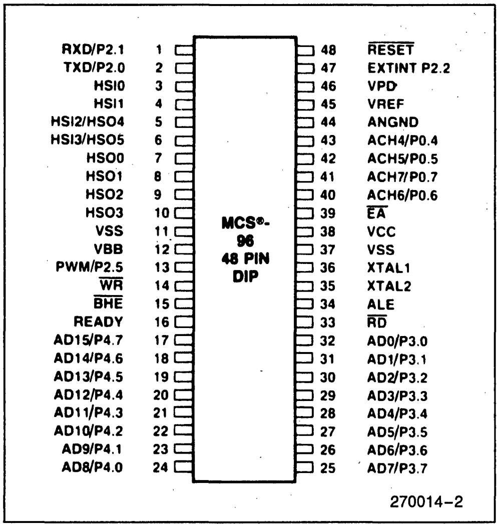


Figure 2. 48-Pin Package

Figure 1 shows a block diagram of the MCS-96 parts, generally referred to as the 8096. The 8096 is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM. The MCS-96 numbering system is shown below:

Options		68-Pin	48-Pin
Digital I/O	ROMLESS	8096-90	
	ROM	8396-90	
Analog and Digital I/O	ROMLESS	8097-90	8095-90
	ROM	8397-90	8395-90

Figures 2, 3 and 4 show the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin version comes in a Plastic Leaded Chip Carrier and a Pin Grid Array.

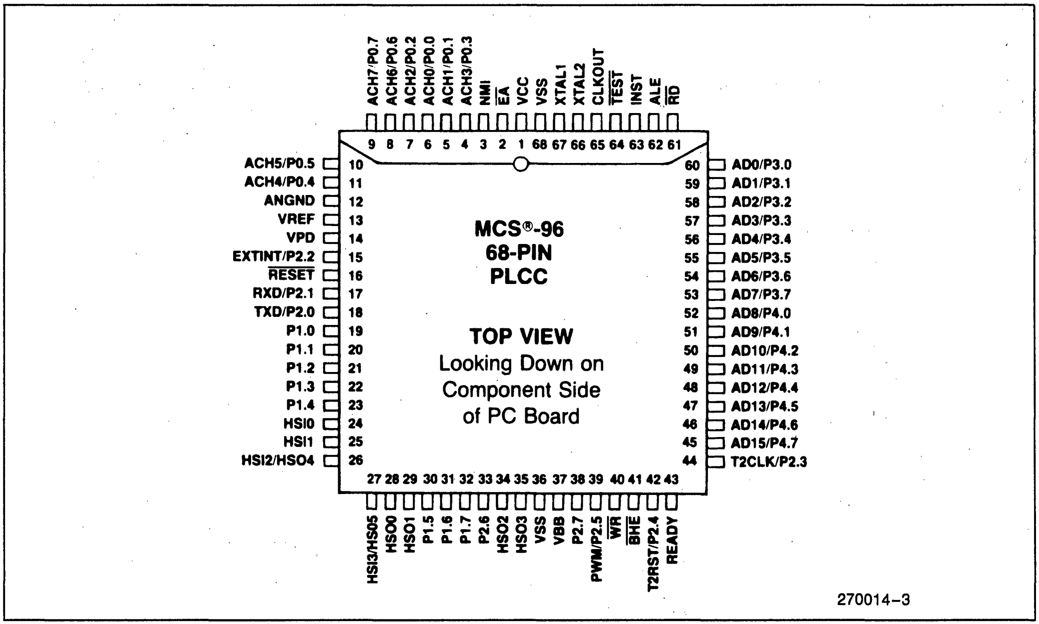
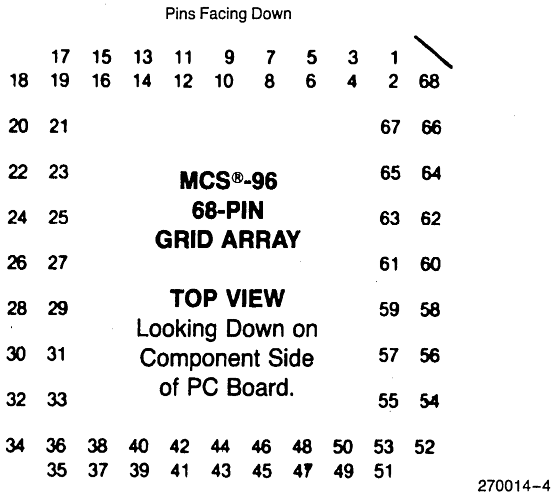


Figure 3. 68-Pin PLCC Package



270014-4

Figure 4. Pin Grid Array

PGA	PLCC	Description	PGA	PLCC	Description	PGA	PLCC	Description
1	9	ACH7/P0.7	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{\text{EA}}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	VCC	32	46	AD14/P4.6	55	23	P1.4
10	68	VSS	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	$\overline{\text{TEST}}$	37	41	$\overline{\text{BHE}}$	60	18	TXD/P2.0
15	63	INST	38	40	$\overline{\text{WR}}$	61	17	RXD/P2.1
16	62	ALE	39	39	PWM/P2.5	62	16	$\overline{\text{RESET}}$
17	61	$\overline{\text{RD}}$	40	38	P2.7	63	15	EXTINT/P2.2
18	60	AD0/P3.0	41	37	VBB	64	14	VPD
19	59	AD1/P3.1	42	36	VSS	65	13	VREF
20	58	AD2/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4
22	56	AD4/P3.4	45	33	P2.6	68	10	ACH5/P0.5
23	55	AD5/P3.5	46	32	P1.7			

FUNCTIONAL OVERVIEW

The following section is an overview of the 8096, the generic part number used to refer to the entire MCS-96 product family. Additional information is available in the Microcontroller Handbook, order number 210918.

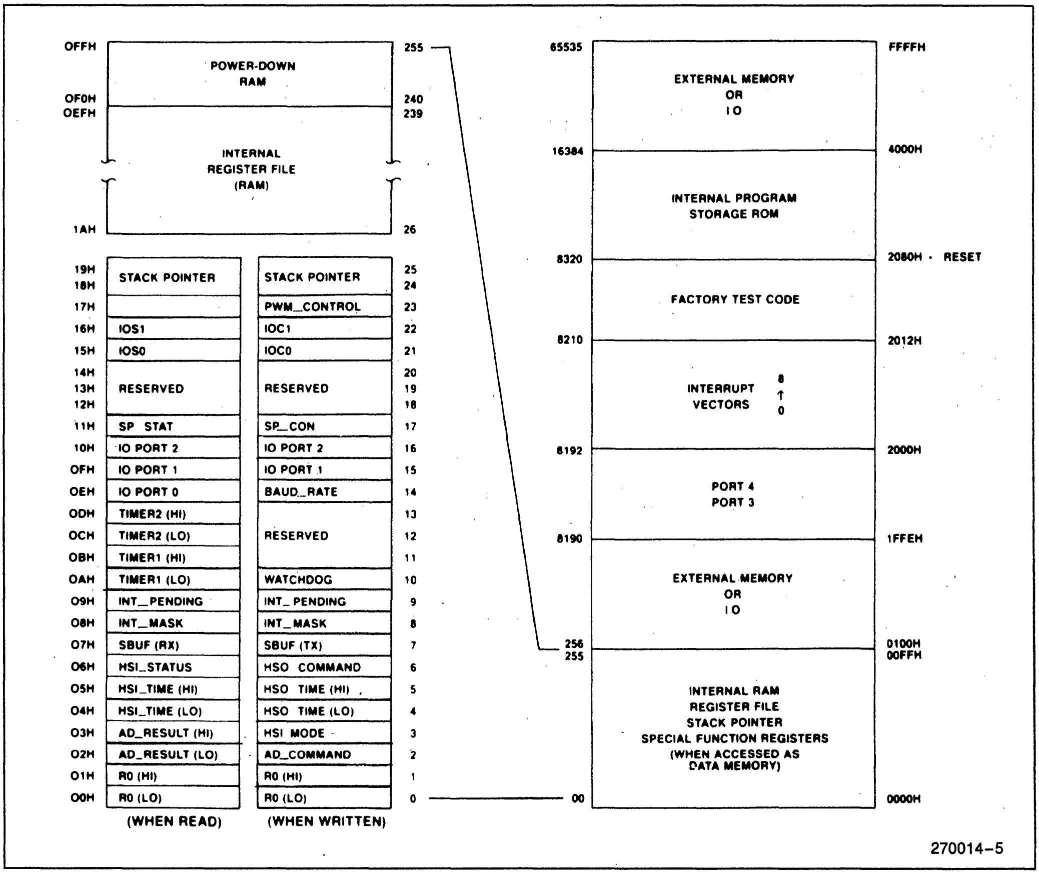
CPU Architecture

The 8096 has 64 Kbyte addressability and uses the same address space for both program and data memory, except in the address range from 00H through 0FFH. Data fetches in this range are always to the Register File, while instruction fetches from these locations are directed to external memory. (Locations 00H through 0FFH in external memory are reserved for Intel development systems.)

Within the Register File, locations 00H through 17H are register mapped I/O control registers, also referred to as Special Function Registers (SFRs). The rest of the Register File (018H through 0FFH) contains 232 bytes of RAM, which can be referenced as bytes, words, or double-words. This register space allows the user to keep the most frequently-used variables in on-chip RAM, which can be accessed faster than external memory. Locations 0F0H through 0FFH can be preserved during power down if power is applied to the VPD pin.

Outside of the register file, program memory, data memory, and peripherals can be intermixed. The addresses with special significance are:

0000H—0017H	Register-mapped I/O (SFRs)
0018H—0019H	Stack Pointer
1FFEH—1FFFH	Ports 3 and 4
2000H—2011H	Interrupt Vectors
2012H—207FH	Factory Test Code
2080H	Reset Location



	(WHEN READ)	(WHEN WRITTEN)	
19H 18H	STACK POINTER	STACK POINTER	25 24
17H		PWM_CONTROL	23
16H	IOS1	IOC1	22
15H	IOS0	IOC0	21
14H 13H 12H	RESERVED	RESERVED	20 19 18
11H	SP STAT	SP_CON	17
10H	IO PORT 2	IO PORT 2	16
0FH	IO PORT 1	IO PORT 1	15
0EH	IO PORT 0	BAUD_RATE	14
0DH	TIMER2 (HI)	RESERVED	13
0CH	TIMER2 (LO)		12
0BH	TIMER1 (HI)		11
0AH	TIMER1 (LO)	WATCHDOG	10
09H	INT_PENDING	INT_PENDING	9
08H	INT_MASK	INT_MASK	8
07H	SBUF (RX)	SBUF (TX)	7
06H	HSI_STATUS	HSO COMMAND	6
05H	HSI_TIME (HI)	HSO TIME (HI)	5
04H	HSI_TIME (LO)	HSO TIME (LO)	4
03H	AD_RESULT (HI)	HSI MODE	3
02H	AD_RESULT (LO)	AD_COMMAND	2
01H	R0 (HI)	R0 (HI)	1
00H	R0 (LO)	R0 (LO)	0

Figure 5. Memory Map

The 839x carries 8 Kbytes of on-chip ROM, occupying addresses 2000H through 3FFFH. Instruction or data fetches from these addresses access the on-chip ROM if the $\overline{EA}$ pin is externally held at a logical 1. If the $\overline{EA}$ pin is at a logical 0 these addresses access off-chip memory.

A memory map for the MCS-96 product family is shown in Figure 5.

The RALU (Register/ALU) section consists of a 17-bit ALU, the Program Status Word, the Program Counter, and several temporary registers. A key feature of the 8096 is that it does not use an accumulator. Rather, it operates directly on any register in the Register File. Being able to operate directly on data in the Register File without having to move it into and out of an accumulator results in a significant improvement in execution speed.

In addition to the normal arithmetic and logical functions, the MCS-96 instruction set provides the following special features:

6.25 μs Multiply and Divide
Multiple Shift Instructions

3 Operand Instructions
Normalize Instruction
Software Reset Instruction

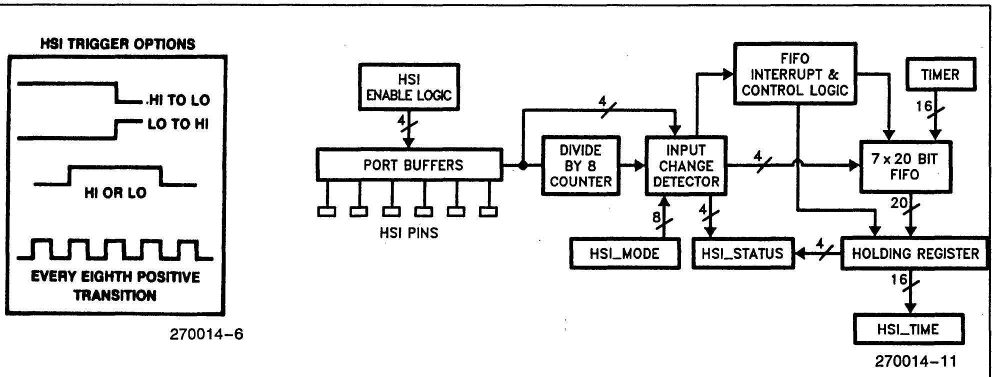
All operations on the 8096 take place in a set number of "State Times." The 8096 uses a three-phase internal clock, so each state time is 3 oscillator periods. With a 12 MHz clock, each state time requires 0.25 microseconds.

High Speed I/O Unit (HSIO)

The HSIO unit consists of the High Speed Input Unit (HSI), the High Speed Output Unit (HSO), one counter and one timer. "High Speed" denotes that the units can perform functions related to the timers without CPU intervention. The HSI records times when events occur and the HSO triggers events at preprogrammed times.

All actions within the HSIO unit are synchronized to the timers. The two 16-bit timer/counter registers in the HSIO unit are cleared on chip reset and can be programmed to generate an interrupt on overflow. The Timer 1 register is automatically incremented every 8 state times (every 2.0 microseconds, with a 12 MHz clock). The Timer 2 register can be programmed to count transitions on either the T2CLK pin or HSI.1 pin. It is incremented on both positive and negative edges of the selected input line. In addition to being cleared by reset, Timer 2 can also be cleared in software or by signals from input pins T2RST or HS1.0. Neither of these timers is required for the watchdog timer or the serial port.

The High Speed Input (HSI) unit can detect transitions on any of its 4 input lines. When one occurs it records the time (from Timer 1) and which input lines



- Pulse measurement with 2.0 μsec resolution
- Input transitions trigger the recording of the reference Timer (16-Bit) and triggered input(s) (4-Bit)

Figure 6. High Speed Input Unit

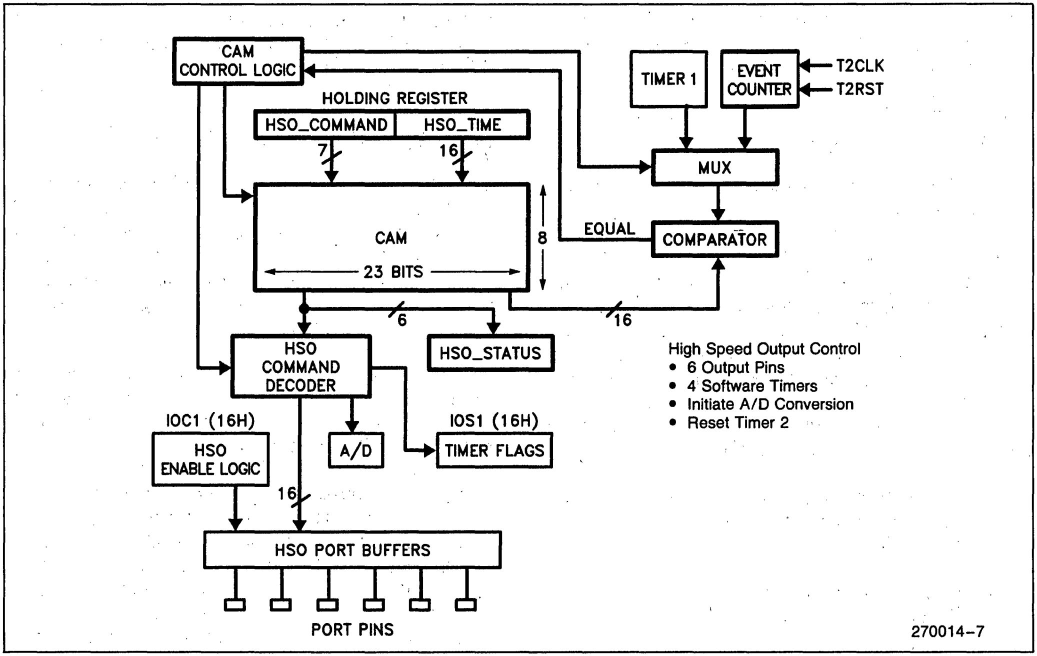


Figure 7. High Speed Output Unit

made the transition. This information is recorded with 2 microsecond resolution and stored in an 8-level FIFO. The unit can be programmed to look for four types of events, as shown in Figure 6. It can activate the HSI Data Available interrupt either when the Holding Registers is loaded or the 6th FIFO entry has been made. Each input line can be individually enabled or disabled to the HSI unit by software.

The High Speed Output (HSO) unit is shown in Figure 7. It can be programmed to set or clear any of its 6 output lines, reset Timer 2, trigger an A/D conversion, or set one of 4 Software Timers flags at a programmed time. An interrupt can be enabled for any of these events. Either Timer 1 or Timer 2 can be referenced for the programmed time value and up to 8 commands for preset actions can be stored in the CAM (Content Addressable Memory) file at any one time. As each action is carried out at its preset time that command is removed from the CAM making space for another command. HSO.4 and HSO.5 are shared with the HSI unit as HSI.2 and HSI.3, and can be individually enabled or disabled as outputs.

Standard I/O Ports

There are 5 8-bit I/O ports on the 8096 in addition to the High Speed I/O lines.

Port 0 is an input-only port which shares its pins with the analog inputs to the A/D Converter. The port can be read digitally and/or, by writing to the A/D Command Register, one of the lines can be selected as the input to the A/D Converter.

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). This configuration allows the pin to be used as either an input or an output without using a data direction register. In parallel with the weak internal pullup, is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time.

Port 2 is multi-functional port. Two of the pins are quasi-bidirectional while the remaining six are shared with other functions in the 8096, as shown below:

Port	Function	Alternate Function
P2.0	output	TXD (serial port transmit)
P2.1	input	RXD (serial port receive)
P2.2	input	EXTINT (external interrupt)
P2.3	input	T2CLK (Timer 2 clock)
P2.4	input	T2RST (Timer 2 reset)
P2.5	output	PWM (pulse-width modulation)

Ports 3 and 4 are bi-directional I/O ports with open drain outputs. These pins are also used as the multiplexed address/data bus when accessing external memory, in which case they have strong internal pullups. The internal pullups are only used during external memory read or write cycles when the pins are outputting address or data bits. At any other time, the internal pullups are disabled.

Serial Port

The serial port is compatible with the MCS®-51 family (8051, 8031 etc.) serial port. It is full duplex, and receive-buffered. There are 3 asynchronous modes and 1 synchronous mode of operation for the serial port. The asynchronous modes allow for 8 or 9 bits of data with even parity optionally inserted for one of the data bits. Selective interrupts based on the 9th data bit are available to support interprocessor communication.

Baud rates in all modes are determined by an independent 16-bit on-chip baud rate generator. Either the XTAL 1 pin or the T2CLK pin can be used as the input to the baud rate generator. The maximum baud rate in the asynchronous mode is 187.5 KBaud.

Pulse Width Modulator (PWM)

The PWM output shares a pin with port bit P2.5. When the PWM output is selected, this pin outputs a pulse train having a fixed period of 256 state times, and a programmable width of 0 to 255 state times. The width is programmed by loading the desired value, in state times, to the PWM Control Register.

A/D Converter

The analog-to-digital converter is a 10-bit, successive approximation converter. It has a fixed conversion time of 168 state times, (42 microseconds with a 12 MHz clock). The analog input must be in the range of 0 to VREF (normally, VREF = 5V). This input can be selected from 8 analog input lines, which connect to the same pins as Port 0. A conversion can be initiated either by setting a control bit in the A/D Command register, or by programming the HSO unit to trigger the conversion at some specified time.

Interrupts

The 8096 has 20 interrupt sources which vector through 8 locations. A 0-to-1 transition from any of the sources sets a corresponding bit in the Interrupt Pending register. The content of the Interrupt Mask register determines if a pending interrupt will be serviced or not. if it is to be serviced, the CPU pushes the current program counter onto the stack and reloads it with the vector corresponding to the desired interrupt. The interrupt vectors are located in addresses 2000H through 2011H, as shown in Figure 8.

Source	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 8. Interrupt Vectors

At the end of the terminal routine the RET instruction pops the program counter from the stack and execution continues where it left off. It is not necessary to store and replace registers during interrupt routines as each routine can be set up to use a different section of the register file. This feature of the architecture provides for very fast context switching.

While the 8096 has a single priority level in the sense that any interrupt may be itself be interrupted, a priority structure exists for resolving simultaneously pending interrupts, as indicated in Figure 8. Since the interrupt pending and interrupt mask registers can be manipulated in software, it is possible to dynamically alter the interrupt priorities to suit the users' software.

Watchdog Timer

The watchdog timer is a 16-bit counter which, once started, is incremented every state time. After 16 milliseconds, if not cleared, it will overflow, pulling down the $\overline{\text{RESET}}$ pin for two state times, causing the system to be reinitialized. This feature is provided as a means of graceful recovery from a software upset. The counter must be cleared by the software before it overflows, or else the system assumes an upset has occurred and activates $\overline{\text{RESET}}$.

PIN DESCRIPTION

VCC

Main supply voltage (5V).

VSS

Digital circuit ground (0V).

VPD

RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e., VCC drops to zero), if $\overline{\text{RESET}}$ is activated before VCC drops below spec and VPD continues to be held within spec, the top 16 bytes in the Register File will retain their contents. $\overline{\text{RESET}}$ must be held low during the Power Down and should not be brought high until VCC is within spec and the oscillator has stablized.

VREF

Reference voltage for the A/D converter (5V). VREF is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0 as digital input.

ANGND

Reference ground for the A/D converter. Should be held at nominally the same potential as VSS.

VBB

Substrate voltage from the on-chip back-bias generator. This pin should be connected to ANGND through a 0.01 μf capacitor (and not connected to anything else).

XTAL1

Input of the oscillator inverter and of the internal clock generator.

XTAL2

Output of the oscillator inverter.

CLKOUT

Output of the internal clock generator. The frequency of CLKOUT is ⅓ the oscillator frequency. It has a 33% duty cycle.

$\overline{\text{RESET}}$

Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared and a jump to address 2080H is executed. Input high for normal operation. $\overline{\text{RESET}}$ has an internal pullup.

TEST

Input low enables a factory test mode. The user should tie this pin to VCC for normal operation.

NMI

A positive transition clears the watchdog timer, and causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.

INST

Output high during an external memory read indicates the read is an instruction fetch. INST needs to be latched on the falling edge of ALE.

$\overline{\text{EA}}$

Input for memory select (External Access). $\overline{\text{EA}}$ = 1 causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM. $\overline{\text{EA}}$ = 0 causes accesses to these locations to be directed to off-chip memory. $\overline{\text{EA}}$ has an internal pulldown, so it goes to 0 unless driven to 1. $\overline{\text{EA}}$ is not latched internally during RESET.

ALE

Address Latch Enable output. ALE is activated only during external memory accesses. It is used to latch the address from the multiplexed address/data bus, and is placed in a low condition during reset.

$\overline{\text{RD}}$

Read signal output to external memory. $\overline{\text{RD}}$ is activated only during external memory reads.

$\overline{\text{WR}}$

Write signal output to external memory. $\overline{\text{WR}}$ is activated only during external memory writes.

$\overline{\text{BHE}}$

Bus High Enable signal output to external memory. $\overline{\text{BHE}}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bust. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{\text{BHE}}$ = 1), to the high byte only (A0 = 1, $\overline{\text{BHE}}$ = 0), or to both bytes (A0 = 0, $\overline{\text{BHE}}$ = 0). $\overline{\text{BHE}}$ is activated only when required during accesses to external memory. $\overline{\text{BHE}}$ can be ignored during read operations. This pin must be latched on the falling edge of ALE.

READY

The READY input is used to lengthen external memory bus cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high CPU operation continues in a normal manner. If the pin is low prior to the first rising edge of CLKOUT after ALE, the Memory Controller goes into a wait mode until the next negative transition in CLKOUT after ALE occurs with READY high. The bus cycle can be lengthened by up to 1 μs. When the external memory bus is not being used, READY has no effect. READY has a weak internal pullup, so it goes to 1 unless externally pulled low.

HSI

Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit.

HSO

Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.

Port 0

8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter.

Port 1

8-bit quasi-bidirectional I/O port.

Port 2

8-bit multi-functional port. Six of its pins are shared with other functions in the 8096, the remaining 2 are quasi-bidirectional.

Ports 3 and 4

8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups.

INSTRUCTION SET

The 8096 instruction set makes use of six addressing modes as described below:

DIRECT—The operand is specified by an 8-bit address field in the instruction. The operand must be in the Register File or SFR space (locations 0000H through 00FFH).

IMMEDIATE—The operand itself follows the opcode in the instruction stream as immediate data. The immediate data can be either 8-bits or 16-bits as required by the opcode.

INDIRECT—An 8-bit address field in the instruction gives the address of a word register in the Register File which contains the 16-bit address of the operand. The operand can be anywhere in memory.

INDIRECT WITH AUTO-INCREMENT—Same as Indirect, except that, after the operand is referenced, the word register that contains the operand's address is incremented by 1 if the operand is a byte, or by 2 if the operand is a word.

INDEXED—The instruction contains an 8-bit address field and either an 8-bit or a 16-bit displacement field. The 8-bit address field gives the address of a word register in the Register File which contains a 16-bit base address. The 8- or 16-bit displacement field contains a signed displacement that will be added to the base address to produce the address of the operand. The operand can be anywhere in memory.

The 8096 contains a Zero Register at word address 0000H (and which contains 0000H). This register is available for performing comparisons and for use as a base register in indexed addressing. This effectively provides direct addressing to all 64K of memory.

In the 8096, the Stack Pointer is at word address 0018H in the Register File. If the 8-bit address field in an indexed instruction contains 18H, the Stack Pointer becomes the base register. This allows direct accessing of variables in the stack.

The following tables list the MCS-96 instructions, their opcodes, and execution times.

Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	✓	✓	✓	✓	↑	—	
ADD/ADDB	3	D ← B + A	✓	✓	✓	✓	↑	—	
ADDC/ADDCB	2	D ← D + A +C	↓	✓	✓	✓	↑	—	
SUB/SUBB	2	D ← D − A	✓	✓	✓	✓	↑	—	
SUB/SUBB	3	D ← B − A	✓	✓	✓	✓	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	✓	✓	✓	↑	—	
CMP/CMPB	2	D − A	✓	✓	✓	✓	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	✓	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	✓	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	2
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	3
AND/ANDB	2	D ← D and A	✓	✓	0	0	—	—	
AND/ANDB	3	D ← B and A	✓	✓	0	0	—	—	
OR/ORB	2	D ← D or A	✓	✓	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	✓	✓	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ✓	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR (indirect)	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Instruction Summary (Continued)

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 − D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	✓	✓	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — — lsb ← 0	✓	?	✓	✓	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — — lsb → C	✓	?	✓	0	—	✓	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — — lsb → C	✓	✓	✓	0	—	✓	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left Shift Till msb = 1; D ← shift count	✓	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⊛					INDEXED⊛				
								NORMAL			AUTO-INC.		SHORT			LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29

270014–9

NOTES:

* Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect+ or Long Indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

1. Number of state times shown for internal/external operands.
2. The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.
3. State times shown for 16-bit bus.

		DIRECT			IMMEDIATE			INDIRECT⑥					INDEXED⑥				
								NORMAL			AUTO-INC.		SHORT			LONG	
MNEMONIC	OPERANDS	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	——	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	——	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	——	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	——	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
BR[ ]	E3	2	8	RET	F0	1	12/16⑤
				TRAP③	F7	1	21/24

270014-10

NOTES:
1. Number of state times shown for internal/external operands.
3. The assembler does not accept this mnemonic.
4. The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
5. State times for stack located internal/external.

Conditional Jumps

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.

Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic	Opcode
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

Jump on Bit Clear or Bit Set

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.

	Bit Number							
Mnemonic	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

DJNZ OPCODE EO; 3 BYTES; 5/9 STATE TIMES (NOT TAKEN/TAKEN)

Single Register Instructions

Mnemonic	Opcode	Bytes	States	Mnemonic	Opcode	Bytes	States
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

Shift Instructions

Instr Mnemonic	Word		Instr Mnemonic	Byte		Instr Mnemonic	DBL WD		State Times
	OP	B		OP	B		OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT(7)
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT(7)
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT(7)

Special Control Instructions

Mnemonic	Opcode	Bytes	States	Mnemonic	Opcode	Bytes	States
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST (6)	FF	1	166	SKIP	00	2	4

Normalize

Mnemonic	Opcode	Bytes	State Times
NORML	0F	3	11 + 1 PER SHIFT

NOTES:

6. This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.
7. Execution will take at least 8 states, even for 0 shift.

FUNCTIONAL DEVIATIONS

Functional deviations from the 809x and 839x on the 809x–90 and 839x–90.

CPU Section

1. Indexed, 3 Operand Multiply—The displacement portion of an indexed, three word multiply may not be in the range of 200H thru 17FFH inclusive. This also applies to byte multiples that use 3 operands.
2. Add or Subtract with carry—The zero flag is both set and cleared by these instructions. Zero checking must be done after each operation.
3. EXT—This instruction never sets the N flag, and always sets the Z flag. The EXTB works correctly. Check the flags before executing an EXT instruction. Additionally, having more than two wait states during an EXT (extend word only) instruction may cause the instruction to give an incorrect result.
4. Read-Modify-Write on Interrupt Pending—A read-modify-write instruction on the interrupt pending register may cause interrupts that occur during execution of the instruction to be missed.
5. READY line—The READY line should not be brought low during the execution of an instruction that accesses HSI_TIME, SP_STAT or IOS1. It should also not be brought low for a data write during the instruction immediately preceding one of the above operations. Do not use wait states for program memory that holds these instructions. Also place a NOP between writes to slow memory and accesses to HSO_TIME, SP_STAT or IOS1.

   The READY line also should not be brought low for more than two state times when using the EXT (extend word) instruction.
6. Signed Divide—The V and VT flags may indicate an overflow after a signed divide when no overflow has occurred.
7. The sticky flag is not affected when a shift by zero is executed on an 8X9X-90.
8. The JBS and JBC instructions should not be used directly on Port 2.1 or any pins of Port 0 if used as digital input. If it is necessary to test these pins, first LD the port data into a temporary register, and then test the bit there.

HSI/HSO Section

1. HSI Timing—An event occurring within 16 state times of a prior event on the same HSI line may not be recorded. Additionally, an event occurring within 16 state times of a prior event on another HSI line may be recorded with a time tag one count earlier than expected. Events are defined as the condition the line is set to trigger on. The effective resolution is increased to 4 μs for such closely spaced events.
2. HSI Divide by 8 Mode—If an event on a pin set to look for every eighth transition occurs less than 16 state times after an event on any other pin, then the divide by 8 event will be recorded twice in the HSI FIFO. The time tag of the duplicate FIFO entry will be equal to that of the initial entry plus one. The programmer's software should detect and discard the second entry.
3. HSO Interrupts—Software timer interrupts cannot be generated by the HSO commands that reset Timer 2 or start an A to D conversion.
4. The first few instructions of an interrupt service routine should check IOS1.7 and exit if the Holding Register is not loaded. This will successfully clear unwanted events.

Serial Port Section

1. Serial Port Flags—Reading SP_STAT may not clear the TI or RI flag if that flag was set within two state times prior to the read. In addition, the parity error bit (RPE/RB8) may not be correct if it is read within two state times after RI is set.

Use the following code to replace ORB sp_image, SP_STAT.

```
SP_READ:
  LDB TEMP, SP_STAT
  ORB SP_IMAGE, SP_STAT
  JBS TEMP,5,SP_READ  ; if TI bit is set
                      ; then read again
  JBS TEMP,6,SP_READ  ; if RI bit is set
                      ; then read again
  ANDB SP_IMAGE,#7FH  ; clear false
                      ; RB8/RPE
  ORB SP_IMAGE,TEMP   ; load correct
                      ; RB8/RPE
```

2. Serial Port Mode 0—The serial port is not tested in mode 0. The receive function in this mode does not work correctly. The receive function will not work unless the first bit shifted in is a one.

3. Serial Port Baud Value—Loading the baud rate register with 8000H (maximum baud rate, internal clock) may cause an 11 millisecond delay (at Fosc = 12 MHz) before the port is properly initialized. After initialization the port works properly. Include a 44000 state time delay after writing 8000H to the Baud Rate Register.

Standard I/O Section

1. Ports 3 and 4 (Internal Execution Mode Only)—To be used as outputs, Ports 3 and 4 each must be addressed as words but written to as bytes. To write to Port 3 use "ST temp, 1ffeh", where the low byte of "temp" contains the data for the port.

To write to Port 4, use the DCB operator to generate the opcode sequence "0C3H, 001H, 0FFH, 01FH, (temp)", where the high byte of "temp" contains the data for the port. Ports 3 and 4 will not work as input ports.

Also, when writing to Ports 3 and 4, the address of the port, (1FFEH, 1FFFH) will appear on the bus pins for 2 oscillator periods before the new data is presented to the pins. Since normal bus control signals (ALE, RD, etc.) are suppressed during writes to these addresses, there is no way to latch the data and prevent this address "glitch" to the outside world. If this presents a problem in an application, port reconstruction must be done at another address as described in the MCS-96 Hardware Design Information Chapter.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . 0°C to +70°C

Storage Temperature . . . . −40°C to +150°C

Voltage from Any Pin to
V_{SS} or ANGND . . . . −0.3V to +7.0V

Average Output Current from Any Pin . . . . 10 mA

Power Dissipation . . . . 1.5 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	0	+70	C
V_{CC}	Digital Supply Voltage	4.50	5.50	V
V_{REF}	Analog Supply Voltage	4.5	5.5	V
f_{OSC}	Oscillator Frequency	6.0	12	MHz
V_{PD}	Power-Down Supply Voltage	4.50	5.50	V

NOTE:
V_{BB} should be connected to ANGND through a 0.01 μF capacitor. ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage (Except $\overline{\text{RESET}}$)	−0.3	+0.8	V	
V_{IL1}	Input Low Voltage, $\overline{\text{RESET}}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage (Except $\overline{\text{RESET}}$, NMI, XTAL1)	2.0	V_{CC} +0.5	V	
V_{IH1}	Input High Voltage, $\overline{\text{RESET}}$ Rising	2.4	V_{CC} +0.5	V	
V_{IH2}	Input High Voltage, $\overline{\text{RESET}}$ Falling	2.1	V_{CC} +0.5	V	
V_{IH3}	Input High Voltage, NMI, XTAL1	2.4	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	V_{CC} Supply Current		200	mA	All Outputs Disconnected
I_{PD}	VPD Supply Current		1	mA	Normal operation and Power-Down
I_{REF}	V_{REF} Supply Current		8	mA	
I_{LI}	Input Leakage Current to all pins of HSI, P3, P4, and to P2.1		±10	μA	V_{in} = 0 to V_{CC}
I_{LI1}	Input Leakage to Port 0		±3	μA	V_{IN} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{\text{EA}}$		100	μA	V_{IH} = 2.4V
I_{IL}	Input Low Current to all pins of P1, and to P2.6, P2.7		−100	μA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{\text{RESET}}$	0.3	−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4, READY		−50	μA	V_{IL} = 0.45V
C_S	Pin Capacitance (Any Pin to V_{SS})		10	pF	fTEST = 1.0 MHz

NOTES:

1. I_{OL} = 0.4 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports. I_{OL} = 2.0 mA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{\text{BHE}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, and $\overline{\text{RESET}}$ and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15).
2. I_{OH} = −20 μA for all pins of P1, for P2.6 and P2.7. I_{OH} = −200 μA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{\text{BHE}}$, $\overline{\text{WR}}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15). P3 and P4, when used as ports, have open-drain outputs.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of VREF. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at VREF = 5.120V.

Resolution	±0.001 VREF
Accuracy	±0.004 VREF
Differential nonlinearity	±0.002 VREF max
Integral nonlinearity	±0.004 VREF max
Channel-to-channel matching	±1 LSB
Crosstalk (DC to 100 KHz)	−60 dB max

A.C. CHARACTERISTICS

(VCC, VPD = 4.5 to 5.5 Volts; T_A = 0°C to 70°C; fosc = 6.0 to 12.0 MHz)

Test Conditions: Load Capacitance on Output Pins = 80 pF
Oscillator Frequency = 12.00 MHz

TIMING REQUIREMENTS (Other system components must meet these specs.)

Symbol	Parameter	Min	Max	Units
TCLYX	READY Hold after CLKOUT Edge	0		ns
TLLYV	End of ALE to READY Setup	−Tosc	2Tosc−60	ns
TLLYH	End of ALE to READY High	2 Tosc+40	4Tosc−60(1)	ns
TYLYH	Non-ready Time		1000	ns
TAVDV	Address Valid to Input Data Valid		5Tosc−90	ns
TRLDV	RD/Active to Input Data Valid		3Tosc−60	ns
TRXDX	Data Hold after RD/inactive(2)	0		ns
TRXDZ	RD/Inactive to Input Data Float(2)		Tosc−20	ns

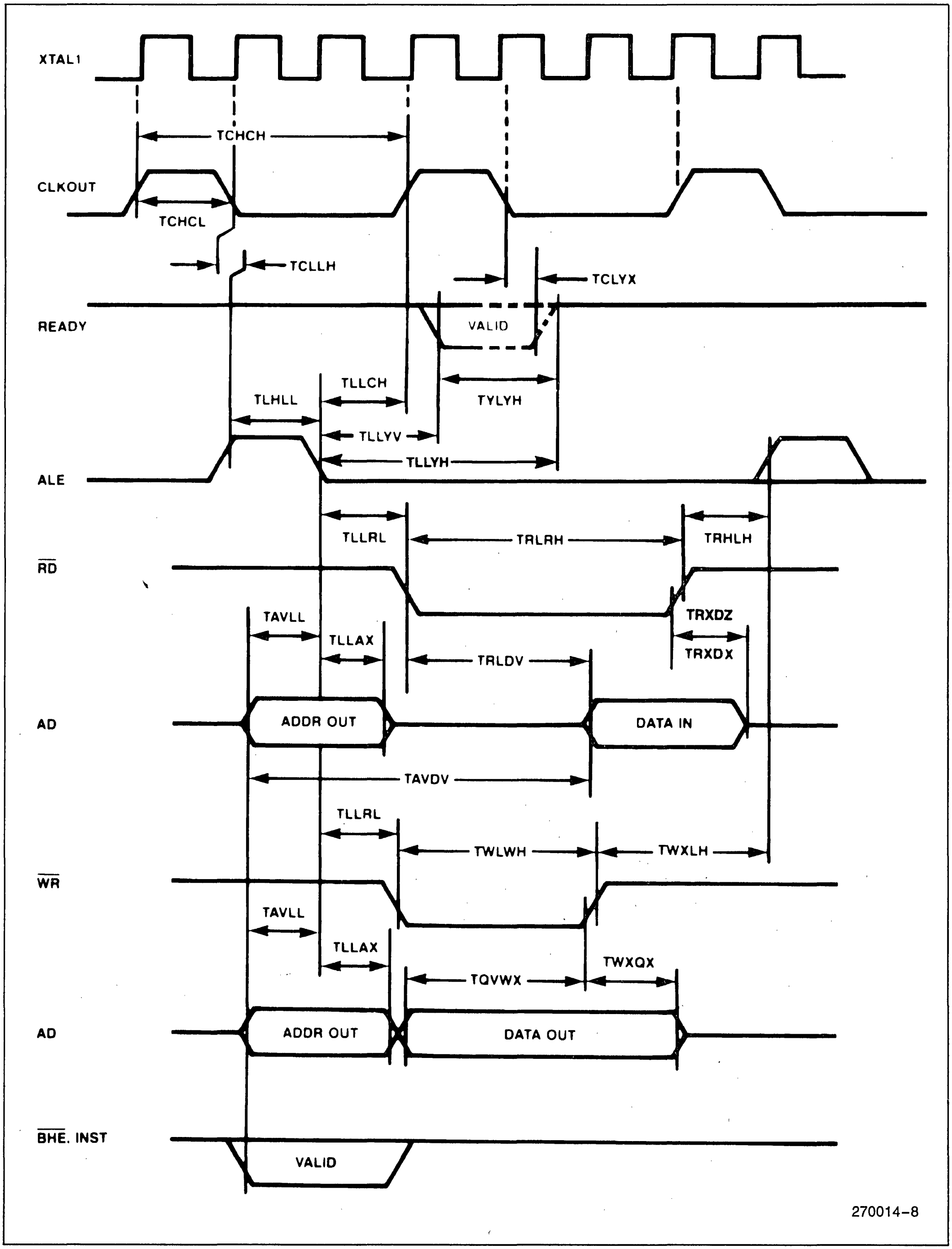
TIMING RESPONSES (MCS-96 parts meet these specs.)

Symbol	Parameter	Min	Max	Units
FXTAL	Oscillator Frequency	6.00	12.00	MHz
Tosc	Oscillator Period	83	166	ns
TOHCH	Oscillator High to CLKOUT High(3)	0	120	ns
TCHCH	CLKOUT Period(2)	3Tosc(3)	3Tosc(3)	ns
TCHCL	CLKOUT High Time	Tosc−20	Tosc+20	ns
TCLLH	CLKOUT Low to ALE High	−25	20	ns
TLLCH	ALE Low to CLKOUT High	Tosc−20	Tosc+40	ns
TLHLL	ALE Pulse Width	Tosc−25	Tosc+15	ns
TAVLL	Address Setup to End of ALE	Tosc−50		ns
TLLRL	End of ALE to RD/ or WR/ Active	Tosc−20		ns
TLLAX	Address Hold After End of ALE	Tosc−20		ns
TWLWH	WR/ Pulse Width	2Tosc−35		ns
TQVWX	Output Data Setup to End of WR/	2Tosc−60		ns
TWXQX	Output Data Hold After End of WR/	Tosc−25		ns
TWXLH	End of WR/ to Next ALE	2Tosc−30		ns
TRLRH	RD/ Pulse Width	3Tosc−30		ns
TRHLH	End of RD/ to Next ALE	Tosc−25		ns

NOTES:

1. If more than one wait state is desired, add 3Tosc for each additional wait state.
2. This specification is not tested, but is verified by design analysis and/or derived from other tested parameters.
3. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ±10 ns if Tosc is constant and the rise and fall times on XTAL 1 are less than 10 ns. CLKOUT is not bonded out on 48-pin parts.

WAVEFORM



Bus Signal Timings

MCS®-96 809XBH/839XBH/879XBH *Express*

- **Extended Temperature Range (−40°C to +85°C)**
- **Burn-In**

The Intel EXPRESS system offers enhancements to the operational specifications of the MCS®-96 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

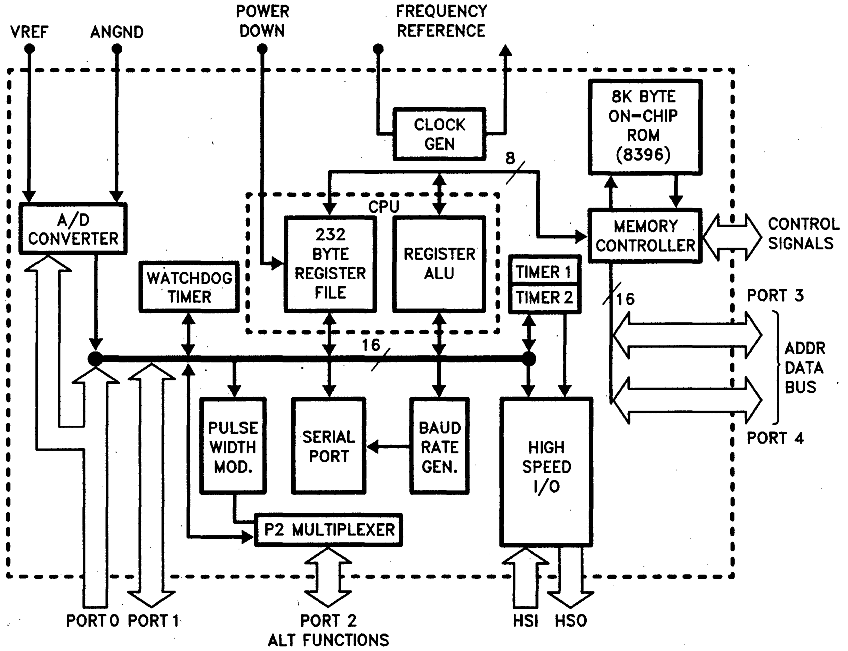
The EXPRESS program includes the commercial standard temperature range with burn-in, and an extended temperature range with or without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to +70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with V_{CC} = 5.5V ±0.5V, following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

This data sheet specifies the parameters for the extended temperature range option. The commercial temperature range data sheets are applicable otherwise.



270433-1

MCS®-96 Block Diagram

ELECTRICAL CHARACTERISTICS ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . −40°C to +85°C

Storage Temperature .... −40°C to +150°C

Voltage from V_{PP} or $\overline{\text{EA}}$
to V_{SS} or ANGND .... −0.3V to +13.0V

Voltage from Any Other Pin to
V_{SS} or ANGND .... −0.3V to +7.0V*

Average Output Current from Any Pin .... 10 mA

Power Dissipation 1.5W

*This includes V_{PP} on ROM and CPU devices.

**Notice:* *Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	−40	+85	C
V_{CC}	Digital Supply Voltage	4.50	5.50	V
V_{REF}	Analog Supply Voltage	4.50	5.50	V
f_{OSC}	Oscillator Frequency	6.0	12	MHz
V_{PD}	Power-Down Supply Voltage	4.50	5.50	V

NOTE:
ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS (Under listed operating conditions)

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{CC}	V_{CC} Supply Current ($-40°C \leq T_A \leq +85°C$)		270	mA	All Outputs Disconnected.
I_{CC1}	V_{CC} Supply Current ($T_A = +85°C$)		185	mA	
I_{PD}	V_{PD} Supply Current		1	mA	Normal operation and Power-Down.
I_{REF}	V_{REF} Supply Current		10	mA	
V_{IL}	Input Low Voltage (Except $\overline{\text{RESET}}$)	−0.3	+0.8	V	
V_{IL1}	Input Low Voltage, $\overline{\text{RESET}}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage (Except $\overline{\text{RESET}}$, NMI, XTAL1)	2.0	V_{CC} +0.5	V	
V_{IH1}	Input High Voltage, $\overline{\text{RESET}}$ Rising	2.4	V_{CC} +0.5	V	
V_{IH2}	Input High Voltage, $\overline{\text{RESET}}$ Falling Hysteresis	2.1	V_{CC} +0.5	V	
V_{IH3}	Input High Voltage, NMI, XTAL1	2.3	V_{CC} +0.5	V	
I_{LI}	Input Leakage Current to each pin of HSI, P3, P4, and to P2.1.		±10	μA	V_{in} = 0 to V_{CC}
I_{LI1}	D.C. Input Leakage Current to each pin of P0		+3	μA	V_{in} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{\text{EA}}$		100	μA	V_{IH} = 2.4V
I_{IL}	Input Low Current to each pin of P1, and to P2.6, P2.7.		−150	μA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{\text{RESET}}$	−0.25	−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4, READY, BUSWIDTH		−50	μA	V_{IL} = 0.45V
V_{OL}	Output Low Voltage on Quasi-Bidirectional port pins and P3, P4 when used as ports		0.45	V	I_{OL} = 0.8 mA (Note 1)
V_{OL1}	Output Low Voltage on Quasi-Bidirectional port pins and P3, P4 when used as ports		0.75	V	I_{OL} = 2.0 mA (Notes 1, 2, 3)
V_{OL2}	Output Low Voltage on Standard Output pins, $\overline{\text{RESET}}$ and Bus/Control Pins		0.45	V	I_{OL} = 2.0 mA (Notes 1, 2, 3, 4)

D.C. CHARACTERISTICS (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{OH}	Output High Voltage on Quasi-Bidirectional pins	2.4		V	I_{OH} = −20 μA (Note 1)
V_{OH1}	Output High Voltage on Standard Output pins and Bus/Control pins	2.4		V	I_{OH} = −200 μA (Note 1)
I_{OH3}	Output High Current on $\overline{RESET}$	−50		μA	V_{OH} = 2.4V
C_S	Pin Capacitance (Any Pin to V_{SS})		10	pF	fTEST = 1.0 MHz

NOTES:
1. Quasi-bidirectional pins include those on P1, for P2.6 and P2.7. Standard Output Pins include TXD, RXD (Mode 0 only), PWM, and HSO pins. Bus/Control pins include CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, INST and AD0–15.
2. Maximum current per pin must be externally limited to the following values if V_{OL} is held above 0.45V.
   I_{OL} on quasi-bidirectional pins and Ports 3 and 4 when used as ports: 4.0 mA
   I_{OL} on standard output pins and $\overline{RESET}$: 8.0 mA
   I_{OL} on Bus/Control pins: 2.0 mA
3.During normal (non-transient) operation the following limits apply:
   Total I_{OL} on Port 1 must not exceed 8.0 mA.
   Total I_{OL} on P2.0, P2.6, $\overline{RESET}$ and all HSO pins must not exceed 15 mA.
   Total I_{OL} on Port 3 must not exceed 10 mA.
   Total I_{OL} on P2.5, P2.7, and Port 4 must not exceed 20 mA.
4. I_{OL} on HSO.X (X = 0, 4, 5) = 1.6 mA @ 0.5V.

A.C. CHARACTERISTICS (Under listed operating conditions)

Test Conditions: Load Capacitance on Output Pins = 80 pF
Oscillator Frequency = 10 MHz

TIMING REQUIREMENTS (Other system components must meet these specs.)

Symbol	Parameter	Min	Max	Units
T_{CLYX}(4)	READY Hold after CLKOUT Edge	0(1)		ns
T_{LLYV}	End of ALE/$\overline{ADV}$ to READY Valid		2Tosc−70	ns
T_{LLYH}	End of ALE/$\overline{ADV}$ to READY High	2Tosc+40	4Tosc−80	ns
T_{YLYH}	Non-Ready Time		1000	ns
T_{AVDV}(6)	Address Valid to Input Data Valid		5Tosc−120	ns
T_{RLDV}	$\overline{RD}$ Active to Input Data Valid		3Tosc−100	ns
T_{RHDX}	Data Hold after $\overline{RD}$ Inactive	0		ns
T_{RHDZ}	$\overline{RD}$ Inactive to Input Data Float	0	Tosc−25	ns
T_{AVGV}(4)(6)	Address Valid to BUSWIDTH Valid		2 Tosc −125	ns
T_{LLGX}(4)	BUSWIDTH Hold after ALE/$\overline{ADV}$ Low	Tosc +40		ns
T_{LLGV}(4)	ALE/$\overline{ADV}$ Low to BUSWIDTH Valid		Tosc −75	ns

NOTES:
1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at 2Tosc+55 (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
4. Pins not bonded out on 48-pin parts.
6. The term "Address Valid" applies to AD0–15, $\overline{BHE}$ and INST.

A.C. CHARACTERISTICS (Continued)

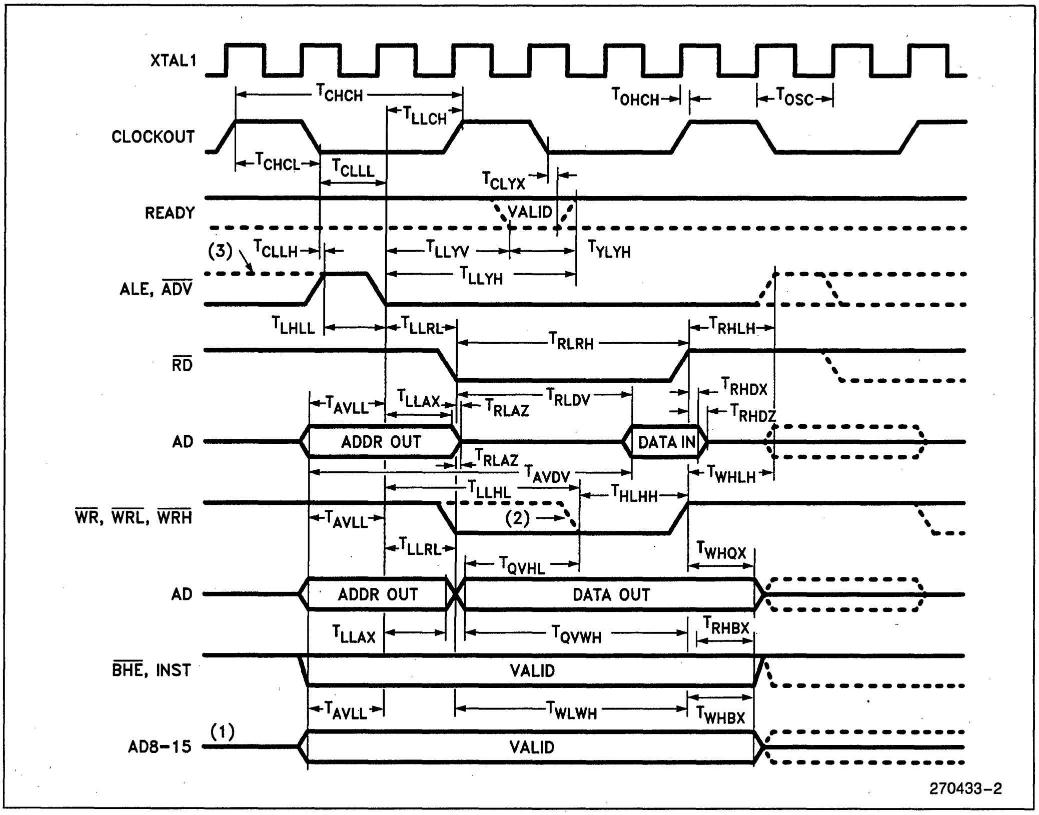
TIMING RESPONSES (MCS-96 parts meet these specs.)

Symbol	Parameter	Min	Max	Units
F_{XTAL}	Oscillator Frequency	6.0	12.0	MHz
T_{OSC}	Oscillator Period	83	166	ns
T_{OHCH}	XTAL1 Rising Edge to Clockout Rising Edge	0(4)	120(4)	ns
T_{CHCH}(4)	CLKOUT Period(3)	3Tosc(3)	3Tosc(3)	ns
T_{CHCL}(4)	CLKOUT High Time	Tosc − 35	Tosc + 10	ns
T_{CLLH}(4)	CLKOUT Low to ALE High	−20	+25	ns
T_{LLCH}(4)	ALE/$\overline{ADV}$ Low to CLKOUT High	Tosc − 25	Tosc + 45	ns
T_{LHLL}	ALE/$\overline{ADV}$ High Time	Tosc − 30	Tosc + 35(5)	ns
T_{AVLL}(6)	Address Setup to End of ALE/$\overline{ADV}$	Tosc − 50		ns
T_{RLAZ}(7)	$\overline{RD}$ or $\overline{WR}$ Low to Address Float		25	ns
T_{LLRL}	End of ALE/$\overline{ADV}$ to $\overline{RD}$ or $\overline{WR}$ Active	Tosc − 40		ns
T_{LLAX}(7)	Address Hold after End of ALE/$\overline{ADV}$	Tosc − 40		ns
T_{WLWH}	$\overline{WR}$ Pulse Width	3Tosc − 35		ns
T_{QVWH}	Output Data Valid to End of $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$	3Tosc − 60		ns
T_{WHQX}	Output Data Hold after $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$	Tosc − 50		ns
T_{WHLH}	End of $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$ to ALE/$\overline{ADV}$ High	Tosc − 75		ns
T_{RLRH}	$\overline{RD}$ Pulse Width	3Tosc − 30		ns
T_{RHLH}	End of $\overline{RD}$ to ALE/$\overline{ADV}$ High	Tosc − 45		ns
T_{CLLL}(4)	CLOCKOUT Low to ALE/$\overline{ADV}$ Low	Tosc − 40	Tosc + 35	ns
T_{RHBX}(4)	$\overline{RD}$ High to INST, $\overline{BHE}$, AD8-15 Inactive	Tosc − 25	Tosc + 30	ns
T_{WHBX}(4)	$\overline{WR}$ High to INST, $\overline{BHE}$, AD8-15 Inactive	Tosc − 50	Tosc + 100	ns
T_{HLHH}	$\overline{WRL}$, $\overline{WRH}$ Low to $\overline{WRL}$, $\overline{WRH}$ High	2Tosc − 35	2Tosc + 40	ns
T_{LLHL}	ALE/$\overline{ADV}$ Low to $\overline{WRL}$, $\overline{WRH}$ Low	2Tosc − 30	2Tosc + 55	ns
T_{QVHL}	Output Data Valid to $\overline{WRL}$, $\overline{WRH}$ Low	Tosc − 60		ns

NOTES:

2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ± 10 ns if Tosc is constant and the rise and fall times on XTAL1 are less than 10 ns.
4. Pins not bonded out on 48-pin parts.
5. Max spec applies only to ALE. Min spec applies to both ALE and $\overline{ADV}$.
6. The term "Address Valid" applies to AD0–15, $\overline{BHE}$ and INST.
7. The term " Address" in this definition applies to AD0–7 for 8-bit cycles, and AD0–15 for 16-bit cycles.

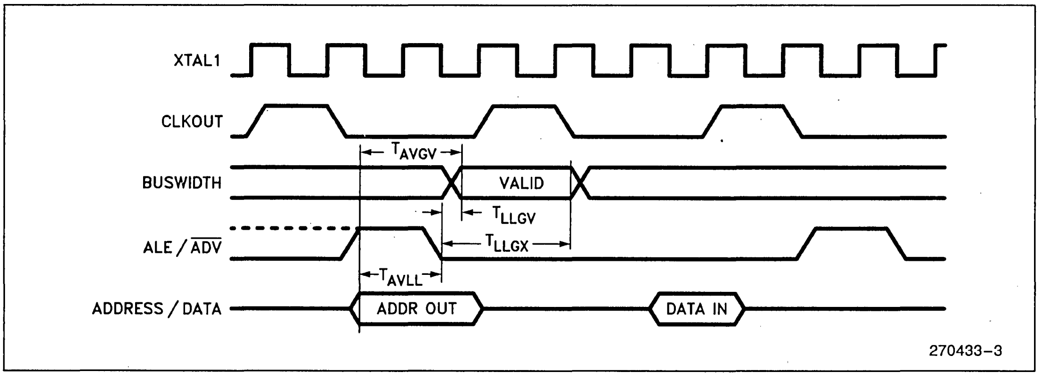
WAVEFORM



270433–2

NOTES:
(1) 8-bit bus only.
(2) 8-bit bus; or when write strobe mode selected.
(3) When ADV selected.

WAVEFORM—BUSWIDTH PIN



270433-3

A.C. CHARACTERISTICS—SERIAL PORT—SHIFT REGISTER MODE

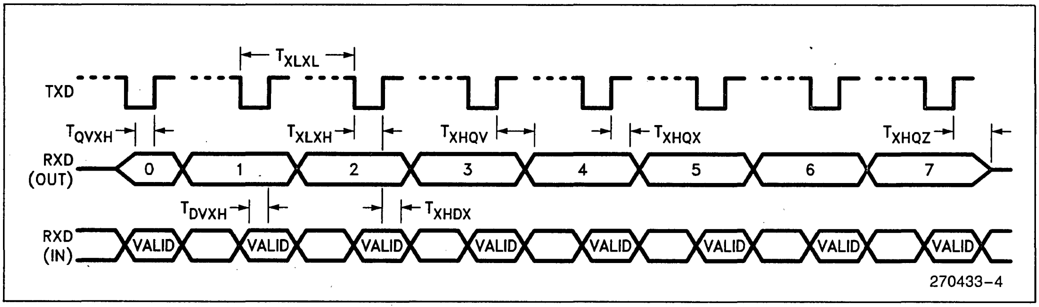
SERIAL PORT TIMING—SHIFT REGISTER MODE

Test Conditions: T_A = 0°C to +70°C; V_{CC} = 5V ±10%; V_{SS} = 0V; Load Capacitance = 80 pF

Symbol	Parameter	Min	Max	Units
T_{XLXL}	Serial Port Clock Period	$8T_{OSC}$		ns
T_{XLXH}	Serial Port Clock Falling Edge to Rising Edge	$4T_{OSC} - 50$	$4T_{OSC} + 50$	ns
T_{QVXH}	Output Data Setup to Clock Rising Edge	$3T_{OSC}$		ns
T_{XHQX}	Output Data Hold After Clock Rising Edge	$2T_{OSC} - 50$		ns
T_{XHQV}	Next Output Data Valid After Clock Rising Edge		$2T_{OSC} + 50$	ns
T_{DVXH}	Input Data Setup to Clock Rising Edge	$2T_{OSC} + 200$		ns
T_{XHDX}	Input Data Hold After Clock Rising Edge	0		ns
T_{XHQZ}	Last Clock Rising to Output Float		$5T_{OSC}$	ns

WAVEFORM—SERIAL PORT—SHIFT REGISTER MODE

SERIAL PORT WAVEFORM—SHIFT REGISTER MODE

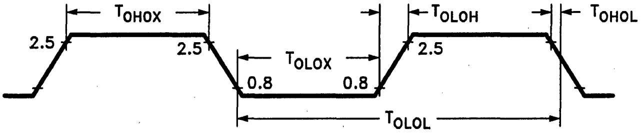


270433-4

EXTERNAL CLOCK DRIVE

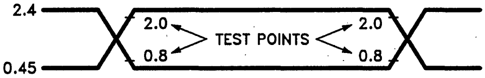
Symbol	Parameter	Min	Max	Units
$1/T_{OLOL}$	Oscillator Frequency	6	12	MHz
T_{OHOX}	High Time	25		ns
T_{OLOX}	Low Time	25		ns
T_{OLOH}	Rise Time		15	ns
T_{OHOL}	Fall Time		15	ns

EXTERNAL CLOCK DRIVE WAVEFORMS



270433-5

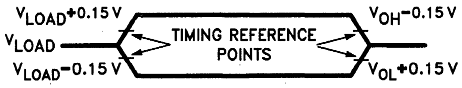
A.C. TESTING INPUT, OUTPUT WAVEFORM



270433-6

A.C. Testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

FLOAT WAVEFORM



270433-7

For Timing Purposes a Port Pin is no Longer Floating when a 100 mV change from Load Voltage Occurs, and Begins to Float when a 100 mV change from the Loaded V_{OH}/V_{OL} Level occurs $I_{OL}/I_{OH} \geq \pm 15$ mA.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097BH, 8397BH, 8095BH, 8395BH, 8797BH, 8795BH.

The absolute conversion accuracy is dependent on the accuracy of V_{REF}. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at $V_{REF} = 5.120V$.

Parameter	Typical*(1)	Minimum	Maximum	Units**	Notes
Resolution		1024 10	1024 10	Levels Bits	
Absolute Error		0	±4	LSBs	
Full Scale Error	−0.5 ±0.5			LSBs	
Zero Offset Error	±0.5			LSBs	
Non-Linearity		0	±4	LSBs	
Differential Non-Linearity		0	±2	LSBs	
Channel-to-Channel Matching		0	±1	LSBs	
Repeatability	±0.25			LSBs	1
Temperature Coefficients: Offset Full Scale Differential Non-Linearity	 0.009 0.009 0.009			 LSB/°C LSB/°C LSB/°C	 1 1 1
Off Isolation		−60		dB	1, 2, 4
Feedthrough	−60			dB	1, 2
V_{CC} Power Supply Rejection	−60			dB	1, 2
Input Resistance		1K	5K	Ω	1
D.C. Input Leakage		0	3.0	μA	
Sample Delay		$3T_{OSC} - 50$	$3T_{OSC} + 50$	ns	1, 3
Sample Time		$12T_{OSC} - 50$	$12T_{OSC} + 50$	ns	1
Sampling Capacitor			2	pF	

NOTES:
* These values are expected for most parts at 25°C.
** An "LSB", as used here, is defined in the glossary which follows and has a value of approximately 5 mV.
1. These values are not tested in production and are based on theoretical estimates and laboratory tests.
2. DC to 100 KHz.
3. For starting the A/D with an HSO Command.
4. Multiplexer Break-Before-Make Guaranteed.

8096 BH Products

Code Memory	A/D	Analog Inputs	I/O Pins	Leads	Product	Package*
ROMless	No	0	48	68	8096BH	N
	Yes	4	32	48	8095BH	P LP
		8	48	68	8097BH	A LA N LN
ROM	No	0	48	68	8396BH	A LA TA N LN TN
	Yes	4	32	48	8395BH	P LP TP
		8	48	68	8397BH	A LA TA N LN TN
EPROM	Yes	4	32	48	8795BH	C LC
		8	48	68	8797BH	A LA R LR

Table 1. MCS®-96 Prefix Identification

*A =	Commercial/No Burn-In 68L Ceramic F6A	TX =	Extended Temp/No Burn-In
N =	Commercial/No Burn-In 68L PLCC	QX =	Commercial/With Burn-In
C =	Commercial/No Burn-In 48L DIP (Ceramic)	LX =	Extended Temp/With Burn-In
P =	Commercial/No Burn-In 48L DIP (Plastic)		

PRELIMINARY

MCS®-96
809X-90, 839X-90
Express

- **Extended Temperature Range (−40°C to +85°C)**
- **Burn-In**

The Intel EXPRESS system offers enhancements to the operational specifications of the MCS®-96 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

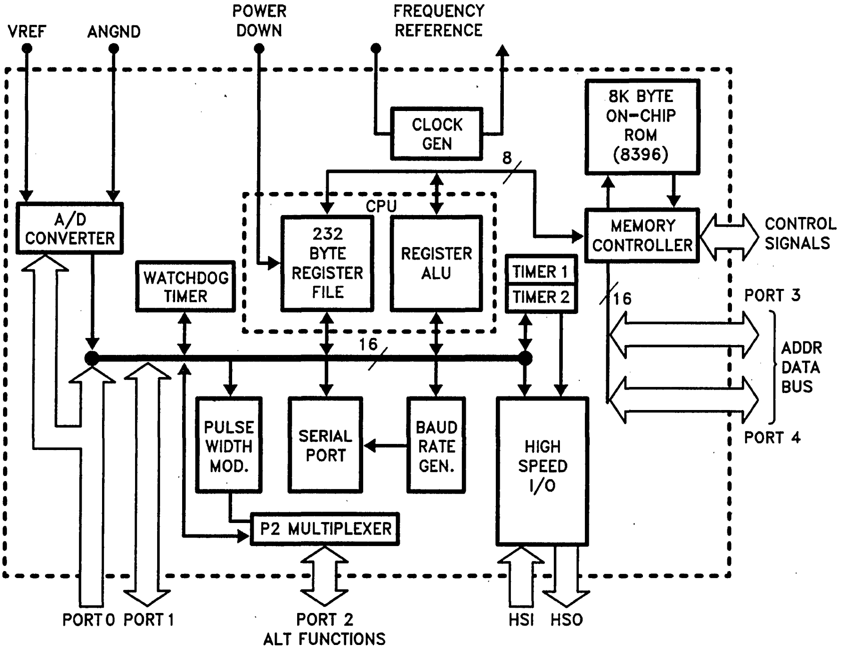
The EXPRESS program includes the commercial standard temperature range with burn-in, and an extended temperature range with or without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to 70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with V_{CC} = 5.5V ±0.5V, following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

This data sheet specifies the parameters for the extended temperature range option. The commercial temperature range data sheets are applicable otherwise.



270104–1

MCS-96 Block Diagram

ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . −40°C to +85°C

Storage Temperature .... −40°C to +150°C

Voltage from Any Pin to
VSS or ANGND .... −0.3V to +7.0V

Average Output Current from Any Pin .... 10 mA

Power Dissipation.......................... 1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	−40	+85	°C
V_{CC}	Digital Supply Voltage	4.5	5.5	V
V_{REF}	Analog Supply Voltage	4.5	5.5	V
f_{OSC}	Oscillator Frequency	6.0	12	MHz
V_{PD}	Power-Down Supply Voltage	4.5	5.5	V

NOTE:
V_{BB} should be connected to ANGND through a 0.01 μF capacitor. ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS T_A = −40°C to +85°C

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage (Except $\overline{\text{RESET}}$)	−0.3	+0.8	V	
V_{IL1}	Input Low Voltage, $\overline{\text{RESET}}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage (Except $\overline{\text{RESET}}$, NMI, XTAL1)	2.0	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage, NMI, XTAL1, $\overline{\text{RESET}}$	2.4	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.5	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	V_{CC} Supply Current		200	mA	All Outputs Disconnected
I_{PD}	V_{PD} Supply Current		1	mA	Normal Operation and Power-Down
I_{REF}	V_{REF} Supply Current		10	mA	
I_{LI}	Input Leakage Current to All Pins of HSI, P0, P3, P4, and to P2.1		±10	μA	V_{in} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{\text{EA}}$		100	μA	V_{IH} = 2.4V
I_{IL}	Input Low Current to All Pins of P1, and to P2.6, P2.7		−100	μA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{\text{RESET}}$		−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4, READY		−50	μA	V_{IL} = 0.45V
C_s	Pin Capacitance (Any Pin to V_{SS})		10	pF	f_{TEST} = 1 MHz

NOTES:

1. I_{OL} = 0.4 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports. I_{OL} = 2.0 mA for TXD, RSD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{\text{BHE}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15).
2. I_{OH} = −20 μA for all pins of P1, for P2.6 and P2.7. I_{OH} = −200 μA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{\text{BHE}}$, $\overline{\text{WR}}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15). P3 and P4, when used as ports, have open-drain outputs.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of V_{REF}. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at V_{REF} = 5.120V.

Resolution	±0.001 V_{REF}
Accuracy	±0.004 V_{REF}
Differential nonlinearity	±0.002 V_{REF} max
Integral nonlinearity	±0.004 V_{REF} max

A.C. CHARACTERISTICS V_{CC}, V_{PD} = 4.5V to 5.5V, T_A = −40°C to +85°C; f_{OSC} = 6.0 MHz to 12.0 MHz

Test Conditions: Load capacitance on output pins = 80 pF
Oscillator Frequency = 12.00 MHz

TIMING REQUIREMENTS Other system components must meet these specs

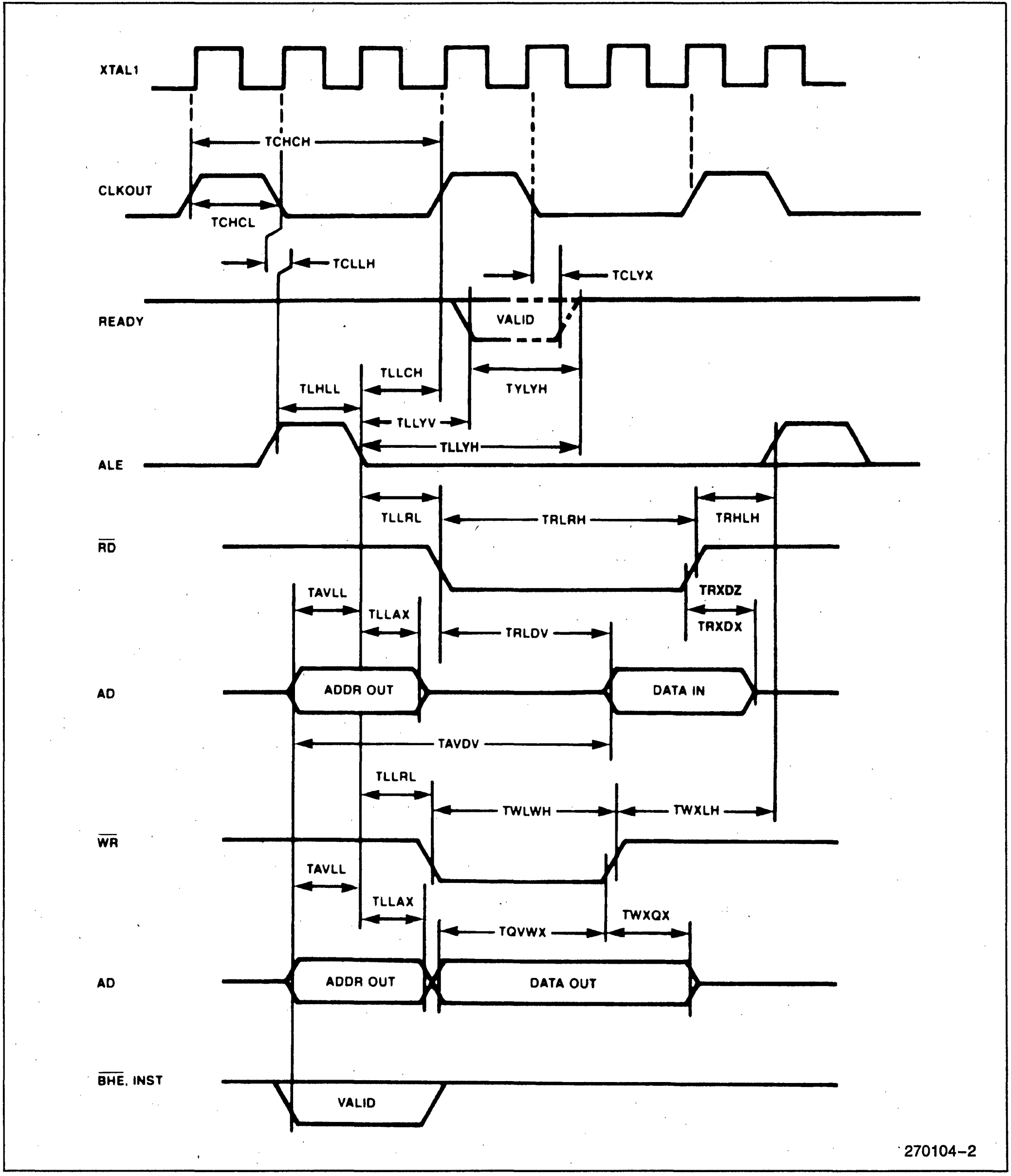
Symbol	Parameter	Min	Max	Units
TCLYX	READY Hold after CLKOUT Falling Edge	0 (Note 1)		ns
TLLYV	End of ALE to READY Setup	−Tosc	2Tosc − 60	ns
TLLYH	End of ALE to READY High	2Tosc + 60	4Tosc − 60 (Note 2)	ns
TYLYH	Non-Ready Time		1000	ns
TAVDV	Address Valid to Input Data Valid		5Tosc − 90	ns
TRLDV	$\overline{RD}$ Active to Input Data Valid		3Tosc − 60	ns
TRXDX	Data Hold after $\overline{RD}$ Inactive (Note 3)	0		ns
TRXDZ	$\overline{RD}$ Inactive to Input Data Float (Note 3)		Tosc − 20	ns

TIMING RESPONSES MCS-96 parts meet these specs

Symbol	Parameter	Min	Max	Units
FXTAL	Oscillator Frequency	6.00	12.00	MHz
Tosc	Oscillator Period	83	166	ns
TCHCH	CLKOUT Period (Note 3)	3Tosc (Note 4)	3Tosc (Note 4)	ns
TCHCL	CLKOUT High Time	Tosc − 20	Tosc + 20	ns
TCLLH	CLKOUT Low to ALE High	−10	30	ns
TLLCH	ALE Low to CLKOUT High	Tosc − 20	Tosc + 40	ns
TLHLL	ALE Pulse Width	Tosc − 25	Tosc + 20	ns
TAVLL	Address Setup to End of ALE	Tosc − 50		ns
TLLRL	End of ALE to $\overline{RD}$ or $\overline{WR}$ Active	Tosc − 20		ns
TLLAX	Address Hold after End of ALE	Tosc − 20		ns
TWLWH	$\overline{WR}$ Pulse Width	2Tosc − 35		ns
TQVWX	Output Data Setup to End of $\overline{WR}$	2Tosc − 60		ns
TWXQX	Output Data Hold after End of $\overline{WR}$	Tosc − 25		ns
TWXLH	End of $\overline{WR}$ to Next ALE	2Tosc − 30		ns
TRLRH	$\overline{RD}$ Pulse Width	3Tosc − 30		ns
TRHLH	End of $\overline{RD}$ to Next ALE	Tosc − 30		ns

NOTES:
1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at 2Tosc + 60 (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. This specification is not tested, but is verified by design analysis and/or derived from other tested parameters.
4. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ±10 ns if Tosc is constant and the rise and fall times on XTAL 1 are less than 10 ns.

WAVEFORM



Bus Signal Timings

Table 1. MCS®-96 Prefix Identification

Prefix	Package Type	Temperature Range	Burn-In
A	Ceramic PGA-68L	Commercial	No
N	PLCC-68L	Commercial	No
C	Ceramic DIP-48L	Commercial	No
TA	Ceramic PGA-68L	Extended	No
TN	PLCC-68L	Extended	No
TC	Ceramic DIP-48L	Extended	No
QA	Ceramic PGA-68L	Commercial	Yes
QN	PLCC-68L	Commercial	Yes
QC	Ceramic DIP-48L	Commercial	Yes
LA	Ceramic PGA-68L	Extended	Yes
LN	PLCC-68L	Extended	Yes
LC	Ceramic DIP-48L	Extended	Yes

EXAMPLES:
A8097-90 indicates an 8097-90 in a ceramic pin grid array package specified for commercial temperature without burn-in.
LC8095-90 indicates an 8095-90 in a ceramic DIP package specified for extended temperature range with burn-in.

80C196KA
16-BIT HIGH PERFORMANCE CHMOS MICROCONTROLLER

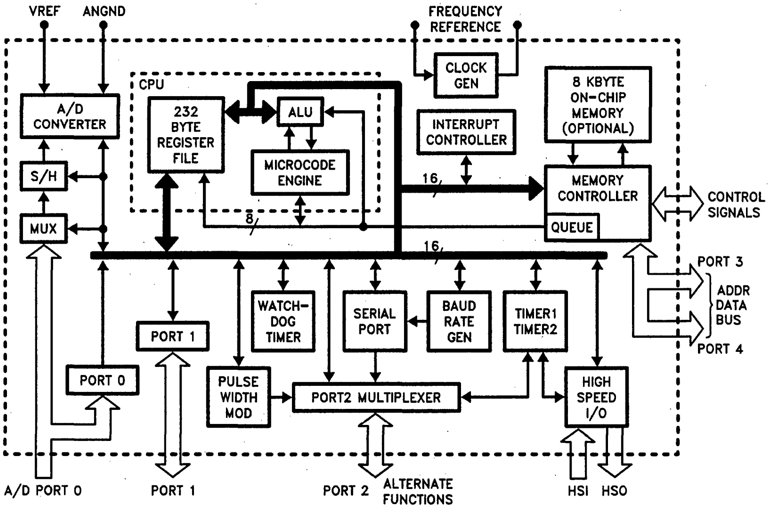
- **232 Byte Register File**
- **Register-to-Register Architecture**
- **28 Interrupt Sources/16 Vectors**
- **2.3 μs 16 x 16 Multiply**
- **4.0 μs 32/16 Divide**
- **Powerdown and Idle Modes**
- **Five 8-Bit I/O Ports**
- **16-Bit Watchdog Timer**
- **Full Duplex Serial Port**
- **Dedicated Baud Rate Generator**
- **High Speed I/O Subsystem**
- **16-Bit Timer**
- **16-Bit Up/Down Counter with Capture**
- **Pulse-Width-Modulated Output**
- **Four 16-Bit Software Timers**
- **10-Bit A/D Converter with S/H**
- **Dynamically Configurable 8-Bit or 16-Bit Buswidth**

The 80C196KA is the CHMOS upgrade for the 8096. It is pin-for-pin compatible and uses a true superset of the 8096 instructions. At the same oscillator frequency the 80C196KA state time generator operates 1.5 times as fast as the 8096. In addition, many instruction execution times have been reduced providing up to twice the performance of a 12 MHz 8096 with a 12 MHz 80C196KA. Intel's CHMOS process provides a high performance processor along with low power consumption. To further reduce power requirements, the processor can be placed into Idle or Powerdown Mode.

Bit, byte, word and some 32-bit operations are available on the 80C196KA. With a 12 MHz oscillator a 16-bit addition takes 0.66 μs, and the instruction times average 0.5 μs to 1.5 μs in typical applications.

Four high-speed capture inputs are provided to record times which events occur. Six high-speed outputs are available for pulse or waveform generation. The high-speed output can also generate four software timers or start an A/D conversion. Events can be based on the timer or up/down counter.

Also provided on-chip are an A/D converter, serial port, watchdog timer, and a pulse-width-modulated output signal.



270428-1

Figure 1. 80C196KA Block Diagram

ARCHITECTURE

The 80C196KA is a member of the MCS®-96 family, and as such has the same architecture and uses the same instruction set as the 8096. Many new features have been added on the 80C196KA including:

CPU FEATURES

Divide by 2 instead of divide by 3 clock for 1.5X performance

Faster instructions, especially indexed/indirect data operations

2.33 μs 16 × 16 multiply with 12 MHz clock (was 6.25 μs)

Faster interrupt response (almost twice as fast)

Powerdown and Idle Modes

Clock Failure Detect

6 new instructions including Compare Long and Block Move

8 new interrupt vectors/6 new interrupt sources

PERIPHERAL FEATURES

SFR Window switching allows read-only registers to be written and vice-versa

Timer2 can count up and down by external selection

Timer2 has an independent capture register

HSO line events are stored in a register

HSO has CAM Lock and CAM Clear commands

New Baud Rate values are needed for serial port, higher speeds possible in all modes

Double buffered serial port transmit register

Serial Port Receive Overrun and Framing Error Detection

PWM has a Divide-by-2 Prescaler

NEW INSTRUCTIONS

PUSHA — PUSHes the PSW, IMASK, IMASK1, and WSR
(Used instead of PUSHF when new interrupts and registers are used.)

assembly language format: PUSHA
object code format: <11110100>
bytes: 1
states: on-chip stack: 12
off-chip stack: 18

POPA — POPs the PSW, IMASK, IMASK1, and WSR
(Used instead of POPF when new interrupts and registers are used.)

assembly language format: POPA

object code format: <11110101>

bytes: 1

states: on-chip stack: 12
off-chip stack:18

IDLPD — Sets the part into Idle or Powerdown Mode

assembly language format: IDLPD #key (key=1 for Idle, key=2 for Powerdown.)

object code format: <11110110> <key>

bytes: 2

states: legal key: 8
illegal key: 25

DJNZW — Decrement Jump Not Zero using a Word counter

assembly language format: DJNZW wreg, cadd

object code format: <11100001> <wreg> <disp>

bytes: 3

states: jump not taken: 5
jump taken: 9

CMPL — Compare 2 long direct values
assembly language format:

```
        DST   SRC
CMPL    Lreg, Lreg
```

object code format: <11000101> <src Lreg> <dst Lreg>

bytes: 3

states: 7

BMOV — Block move using 2 auto-incrementing pointers and a counter

assembly language format:

```
        PTRS  CNTREG
BMOV    Lreg, wreg
```

object code format: <11000001> <wreg> <Lreg>

bytes: 3

states:

internal/internal:	8 per transfer + 6
external/internal:	11 per transfer + 6
external/external:	14 per transfer + 6

SFR OPERATION

All of the registers that were present on the 8096 work the same way as they did, except that the baud rate value is different. The new registers shown in the memory map control new functions. The most important new register is the Window Select Register (WSR) which allows reading of the formerly write-only registers and vice-versa. Using the WSR is described later in this data sheet.

PACKAGING

The 80C196KA is available in 68-pin PLCC and LCC packages. Contact your local sales office to determine the exact ordering code for the part desired.

LCC	PLCC	Description	LCC	PLCC	Description	LCC	PLCC	Description
1	9	ACH7/P0.7	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{\text{EA}}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	V_{CC}	32	46	AD14/P4.6	55	23	P1.4
10	68	V_{SS}	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	BUSWIDTH	37	41	$\overline{\text{BHE}}/\overline{\text{WRH}}$	60	18	TXD/P2.0
15	63	INST	38	40	$\overline{\text{WR}}/\overline{\text{WRL}}$	61	17	RXD/P2.1
16	62	ALE/$\overline{\text{ADV}}$	39	39	PWM/P2.5	62	16	$\overline{\text{RESET}}$
17	61	$\overline{\text{RD}}$	40	38	P2.7/T2CAPTURE	63	15	EXTINT/P2.2
18	60	AD0/P3.0	41	37	V_{PP}	64	14	CDE
19	59	AD1/P3.1	42	36	V_{SS}	65	13	V_{REF}
20	58	AD2/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4
22	56	AD4/P3.4	45	33	P2.6/T2UP/DN	68	10	ACH5/P0.5
23	55	AD5/P3.5	46	32	P1.7			

Figure 2. Pin Definitions

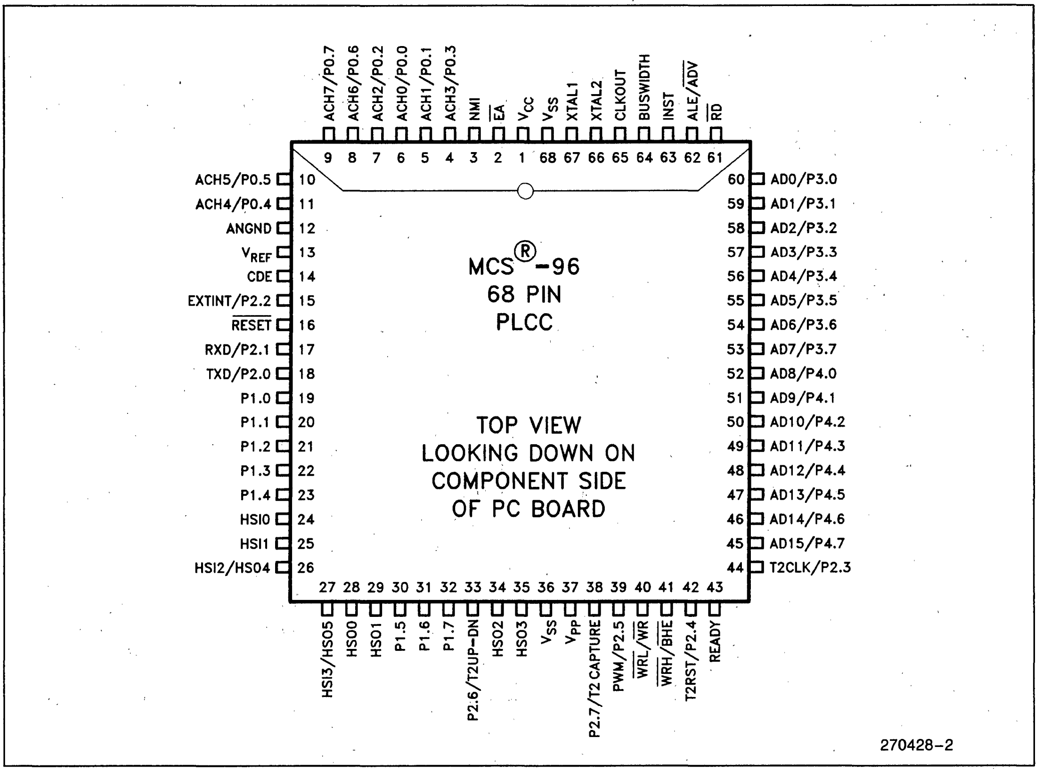


Figure 3. 68-Pin Package (PLCC—Top View)

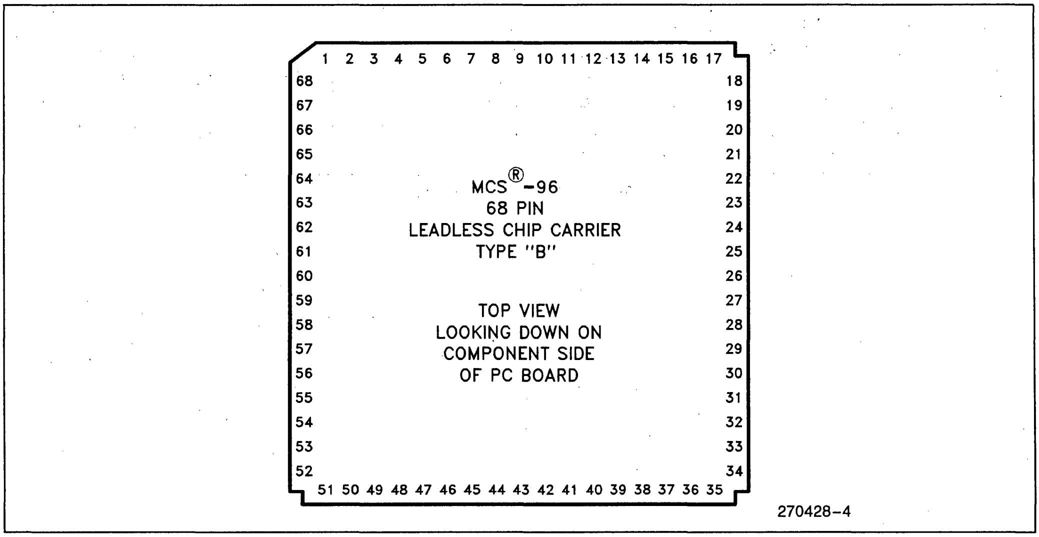


Figure 4. 68-Pin Package (LCC—Top View)

PIN DESCRIPTIONS

Symbol	Name and Function
V_{CC}	Main supply voltage (5V).
V_{SS}	Digital circuit ground (0V). There are two V_{SS} pins, both of which must be connected.
CDE	Clock Detect Enable - When pulled high enables the clock failure detection circuit. If the XTAL1 frequency falls below a specified limit the $\overline{\text{RESET}}$ pin will be pulled low.
V_{REF}	Reference voltage for the A/D converter (5V). V_{REF} is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0. Must be connected for A/D and Port 0 to function.
ANGND	Reference ground for the A/D converter. Must be held at nominally the same potential as V_{SS}.
V_{PP}	Timing pin for the return from powerdown circuit. Connect this pin with a 1 μF capacitor to V_{SS} and a 1 MΩ resistor to V_{CC}. If this function is not used V_{PP} may be tied to V_{CC}. This pin was V_{BB} on the 8X9X-90 parts and will be the programming voltage on future EPROM parts.
XTAL1	Input of the oscillator inverter and of the internal clock generator.
XTAL2	Output of the oscillator inverter.
CLKOUT	Output of the internal clock generator. The frequency of CLKOUT is ½ the oscillator frequency. It has a 50% duty cycle.
$\overline{\text{RESET}}$	Reset input to the chip. Input low for at least 4 state times to reset the chip. The subsequent low-to-high transition re- synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to location 2080H is executed. Input high for normal operation. $\overline{\text{RESET}}$ has an internal pullup.
BUSWIDTH	Input for buswidth selection. If CCR bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is a 1, a 16-bit bus cycle occurs. If BUSWIDTH is a 0 an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. This pin is the $\overline{\text{TEST}}$ pin on 8X9X-90 parts. Systems with $\overline{\text{TEST}}$ tied to V_{CC} do not need to change.
NMI	A positive transition causes a vector through 203EH.
INST	Output high during an external memory read indicates the read is an instruction fetch. INST is valid throughout the bus cycle. INST is activated only during external memory accesses.
$\overline{\text{EA}}$	Input for memory select (External Access). $\overline{\text{EA}}$ equal to a TTL-high causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM/EPROM. $\overline{\text{EA}}$ equal to a TTL-low causes acccesses to these locations to be directed to off-chip memory.
ALE/$\overline{\text{ADV}}$	Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is $\overline{\text{ADV}}$, it goes inactive high at the end of the bus cycle. $\overline{\text{ADV}}$ can be used as a chip select for external memory. ALE/$\overline{\text{ADV}}$ is activated only during external memory accesses.
$\overline{\text{RD}}$	Read signal output to external memory. $\overline{\text{RD}}$ is activated only during external memory reads.
$\overline{\text{WR}}$/$\overline{\text{WRL}}$	Write and Write Low output to external memory, as selected by the CCR. $\overline{\text{WR}}$ will go low for every external write, while $\overline{\text{WRL}}$ will go low only for external writes where an even byte is being written. $\overline{\text{WR}}$/$\overline{\text{WRL}}$ is activated only during external memory writes.
$\overline{\text{BHE}}$/$\overline{\text{WRH}}$	Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{\text{BHE}}$ =0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{\text{BHE}}$ = 1), to the high byte only (A0 = 1, $\overline{\text{BHE}}$ = 0), or both bytes (A0 = 0, $\overline{\text{BHE}}$ = 0). If the $\overline{\text{WRH}}$ function is selected, the pin will go low if the bus cycle is writing to an odd memory location. $\overline{\text{BHE}}$/$\overline{\text{WRH}}$ is valid only during 16-bit external memory write cycles.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
READY	Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the memory controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle held not ready is available through configuration of CCR.
HSI	Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as inputs by future EPROM parts in Programming Mode.
HSO	Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.
Port 0	8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins are also a mode input to future EPROM parts in the Programming Mode.
Port 1	8-bit quasi-bidirectional I/O port.
Port 2	8-bit multi-functional port. All of its pins are shared with other functions in the 80C196KA.
Ports 3 and 4	8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups. Available only on future ROM and EPROM parts.

Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	✓	✓	✓	✓	↑	–	
ADD/ADDB	3	D ← B + A	✓	✓	✓	✓	↑	–	
ADDC/ADDCB	2	D ← D + A + C	↓	✓	✓	✓	↑	–	
SUB/SUBB	2	D ← D – A	✓	✓	✓	✓	↑	–	
SUB/SUBB	3	D ← B – A	✓	✓	✓	✓	↑	–	
SUBC/SUBCB	2	D ← D – A + C – 1	↓	✓	✓	✓	↑	–	
CMP/CMPB	2	D – A	✓	✓	✓	✓	↑	–	
MUL/MULU	2	D,D + 2 ← D × A	–	–	–	–	–	–	2
MUL/MULU	3	D,D + 2 ← B × A	–	–	–	–	–	–	2
MULB/MULUB	2	D,D + 1 ← D × A	–	–	–	–	–	–	3
MULB/MULUB	3	D,D + 1 ← B × A	–	–	–	–	–	–	3
DIVU	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	✓	↑	–	2
DIVUB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	✓	↑	–	3
DIV	2	D ← (D,D + 2) /A,D + 2 ← remainder	–	–	–	✓	↑	–	
DIVB	2	D ← (D,D + 1) /A,D + 1 ← remainder	–	–	–	✓	↑	–	
AND/ANDB	2	D ← D AND A	✓	✓	0	0	–	–	
AND/ANDB	3	D ← B AND A	✓	✓	0	0	–	–	
OR/ORB	2	D ← D OR A	✓	✓	0	0	–	–	
XOR/XORB	2	D ← D (ecxl. or) A	✓	✓	0	0	–	–	
LD/LDB	2	D ← A	–	–	–	–	–	–	
ST/STB	2	A ← D	–	–	–	–	–	–	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	–	–	–	–	–	–	3,4
LDBZE	2	D ← A; D + 1 ← 0	–	–	–	–	–	–	3,4
PUSH	1	SP ← SP – 2; (SP) ← A	–	–	–	–	–	–	
POP	1	A ← (SP); SP + 2	–	–	–	–	–	–	
PUSHF	0	SP ← SP – 2; (SP) ← PSW; PSW ← 0000H; I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ✓	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LJMP	1	PC ← PC + 16-bit offset	–	–	–	–	–	–	5
BR[indirect]	1	PC ← (A)	–	–	–	–	–	–	
SCALL	1	SP ← SP – 2; (SP) ← PC; PC ← PC + 11-bit offset	–	–	–	–	–	–	5
LCALL	1	SP ← SP – 2; (SP) ← PC; PC ← PC + 16-bit offset	–	–	–	–	–	–	5

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	jump if C = 0	—	—	—	—	—	—	5
JE	1	jump if Z = 1	—	—	—	—	—	—	5
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 0	—	—	—	—	—	—	5
JNV	1	Jump if V = 1	—	—	—	—	—	—	5
JVT	1	Jump if VT= 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5,6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5,6
DJNZ/ DJNZW	1	D ← D − 1; If D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 − D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	✓	✓	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb - - - - - lsb ← 0	✓	✓	✓	✓	↑	—	7
SHR/SHRB/SHRL	2	0 → msb - - - - - lsb → C	✓	✓	✓	0	—	✓	7
SHRA/SHRAB/SHRAL	2	msb → msb - - - - - lsb → C	✓	✓	✓	0	—	✓	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
CLRVT	0	VT ← 0	–	–	–	–	0	–	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interupts (I ← 0)	–	–	–	–	–	–	
EI	0	Enable All Interupts (I ← 1)	–	–	–	–	–	–	
NOP	0	PC ← PC + 1	–	–	–	–	–	–	
SKIP	0	PC ← PC + 2	–	–	–	–	–	–	
NORML	2	Left shift till msb = 1;D ← shift count	✓	✓	0	–	–	–	7
TRAP	0	SP ← SP − 2; (SP) ← PC; PC ← (2010H)	–	–	–	–	–	–	9

PUSHA	1	SP ← SP-2; (SP) ← PSW;
		PSW ← 0000H; SP ← SP-2;
		(SP) ← IMASK1/WSR; IMASK1 ← 00H
			0	0	0	0	0	0	

POPA	1	IMASK1/WSR ← (SP); SP ← SP+2 PSW ← (SP); SP ← SP+2	✓	✓	✓	✓	✓	✓	

IDLPD	1	IDLE MODE IF KEY=1;
		POWERDOWN MODE IF KEY =2;
		CHIP RESET OTHERWISE
			–	–	–	–	–	–	

CMPL	2	D-A	✓	✓	✓	✓	↑	–	
BMOV	2	[PTR_HI]+ ← [PTR_LOW]+ ; UNTIL COUNT=0	–	–	–	–	–	–	

NOTES:

1. If the mnemonic ends in "B" a byte operation is performed, otherwise a word operation is done. Operands is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D,D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D,D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to word.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

Instruction Execution State Times

MNEMONIC	DIRECT	IMMED	INDIRECT		INDEXED	
			NORMAL*	A-INC*	SHORT*	LONG*
ADD (3-op)	5	6	7/9	8/10	7/9	8/10
SUB (3-op)	5	6	7/9	8/10	7/9	8/10
ADD (2-op)	4	5	6/8	7/9	6/8	7/9
SUB (2-op)	4	5	6/8	7/9	6/8	7/9
ADDC	4	5	6/8	7/9	6/8	7/9
SUBC	4	5	6/8	7/9	6/8	7/9
CMP	4	5	6/8	7/9	6/8	7/9
ADDB (3-op)	5	5	7/9	8/10	7/9	8/10
SUBB (3-op)	5	5	7/9	8/10	7/9	8/10
ADDB (2-op)	4	4	6/8	7/9	6/8	7/9
SUBB (2-op)	4	4	6/8	7/9	6/8	7/9
ADDCB	4	4	6/8	7/9	6/8	7/9
SUBCB	4	4	6/8	7/9	6/8	7/9
CMPB	4	4	6/8	7/9	6/8	7/9
MUL (3-op)	16	17	18/21	19/22	19/22	20/23
MULU (3-op)	14	15	16/19	17/20	17/20	18/21
MUL (2-op)	16	17	18/21	19/22	19/22	20/23
MULU (2-op)	14	15	16/19	17/20	17/20	18/21
DIV	26	27	28/31	29/32	29/32	30/33
DIVU	24	25	26/29	27/30	27/30	28/31
MULB (3-op)	12	12	14/17	15/18	15/18	16/19
MULUB (3-op)	10	10	12/15	12/16	12/16	14/17
MULB (2-op)	12	12	14/17	15/18	15/18	16/19
MULUB (2-op)	10	10	12/15	12/16	12/16	14/17
DIVB	18	18	20/23	21/24	21/24	22/25
DIVUB	16	16	18/21	19/22	19/22	20/23
AND (3-op)	5	6	7/9	8/10	7/9	8/10
AND (2-op)	4	5	6/8	7/9	6/8	7/9
OR (2-op)	4	5	6/8	7/9	6/8	7/9
XOR	4	5	6/8	7/9	6/8	7/9
ANDB (3-op)	5	5	7/9	8/10	7/9	8/10
ANDB (2-op)	4	4	6/8	7/9	6/8	7/9
ORB (2-op)	4	4	6/8	7/9	6/8	7/9
XORB	4	4	6/8	7/9	6/8	7/9
LD/LDB	4	5	5/7	6/8	6/8	7/9
ST/STB	4	5	5/7	6/8	6/8	7/9
LDBSE	4	4	5/7	6/8	6/8	7/9
LDBZE	4	4	5/7	6/8	6/8	7/9
BMOV	6+8 per word			6 + 11/14 per word		
PUSH (int stack)	6	7	9/12	10/13	10/13	11/14
POP (int stack)	8	—	10/12	11/13	11/13	12/14
PUSH (ext stack)	8	9	11/14	12/15	12/15	13/16
POP (ext stack)	11	—	13/15	14/16	14/16	15/17

*Times for (Internal/External) Operands

Instruction Execution State Times (Continued)

MNEMONIC		MNEMONIC	
PUSHF (int stack)	6	PUSHF (ext stack)	8
POPF (int stack)	7	POPF (ext stack)	10
PUSHA (int stack)	12	PUSHA (ext stack)	18
POPA (int stack)	12	POPA (ext stack)	18
TRAP (int stack)	16	TRAP (ext stack)	18
LCALL (int stack)	11	LCALL (ext stack)	13
SCALL (int stack)	11	SCALL (ext stack)	13
RET (int stack)	11	RET (ext stack)	14
CMPL	7	DEC/DECB	3
CLR/CLRB	3	EXT/EXTB	4
NOT/NOTB	3	INC/INCB	3
NEG/NEGB	3		
LJMP	7		
SJMP	7		
BR [indirect]	7		
JNST, JST	4/8 jump not taken/jump taken		
JNH, JH	4/8 jump not taken/jump taken		
JGT, JLE	4/8 jump not taken/jump taken		
JNC, JC	4/8 jump not taken/jump taken		
JNVT, JVT	4/8 jump not taken/jump taken		
JNV, JV	4/8 jump not taken/jump taken		
JGE, JLT	4/8 jump not taken/jump taken		
JNE, JE	4/8 jump not taken/jump taken		
JBC, JBS	5/9 jump not taken/jump taken		
DJNZ	5/9 jump not taken/jump taken		
DJNZW	5/9 jump not taken/jump taken		
NORML	8 + 1 per shift (9 for 0 shift)		
SHRL	7+ 1 per shift (8 for 0 shift)		
SHLL	7 + 1 per shift (8 for 0 shift)		
SHRAL	7 + 1 per shift (8 for 0 shift)		
SHR/SHRB	6 + 1 per shift (7 for 0 shift)		
SHL/SHLB	6 + 1 per shift (7 for 0 shift)		
SHRA/SHRAB	6 + 1 per shift (7 for 0 shift)		
CLRC	2		
SETC	2		
DI	2		
EI	2		
CLRVT	2		
NOP	2		
RST	15 (includes fetch of configuration byte)		
SKIP	3		
IDLPD	8/25 (proper key/improper key)		

MEMORY MAP

	0FFFFH
EXTERNAL MEMORY OR I/O	
	4000H
INTERNAL ROM/EPROM OR EXTERNAL MEMORY*	
	2080H
RESERVED	
	2040H
UPPER 8 INTERRUPT VECTORS (NEW ON 80C196KA)	
	2030H
ROM/EPROM SECURITY KEY*	
	2020H
RESERVED	
	2019H
CHIP CONFIGURATION BYTE	
	2018H
RESERVED	
	2014H
LOWER 8 INTERRUPT VECTORS PLUS 2 SPECIAL INTERRUPTS	
	2000H
PORT 3 AND PORT 4	
	1FFEH
EXTERNAL MEMORY OR I/O	
	0100H
INTERNAL DATA MEMORY - REGISTER FILE (STACK POINTER, RAM AND SFRS) EXTERNAL PROGRAM CODE MEMORY	
	0000H

*ROM/EPROM will be available on future versions of 80C196.

80C196KA INTERRUPTS

Number	Source	Vector Location	Priority
INT15	NMI	203EH	15
INT14	HSI FIFO Full	203CH	14
INT13	EXTINT Pin	203AH	13
INT12	TIMER2 Overflow	2038H	12
INT11	TIMER2 Capture	2036H	11
INT10	4th Entry into HSI FIFO	2034H	10
INT09	RI	2032H	9
INT08	TI	2030H	8
SPECIAL	Unimplemented Opcode	2012H	N/A
SPECIAL	Trap	2010H	N/A
INT07	EXTINT	200EH	7
INT06	Serial Port	200CH	6
INT05	Software Timer	200AH	5
INT04	HSI.0 Pin	2008H	4
INT03	High Speed Outputs	2006H	3
INT02	HSI Data Available	2004H	2
INT01	A/D Conversion Complete	2002H	1
INT00	Timer Overflow	2000H	0

	WHEN READ		WHEN WRITTEN
19H 18H	STACK POINTER	19H 18H	STACK POINTER
17H	*IOS2	17H	PWM_CONTROL
16H	IOS1	16H	IOC1
15H	IOS0	15H	IOC0
14H	*WSR	14H	*WSR
13H	*INT_MASK 1	13H	*INT_MASK 1
12H	*INT_PEND 1	12H	*INT_PEND 1
11H	*SP_STAT	11H	*SP_CON
10H	PORT2	10H	PORT2
0FH	PORT1	0FH	PORT1
0EH	PORT0	0EH	BAUD RATE
0DH	TIMER2 (HI)	0DH	TIMER2 (HI)
0CH	TIMER2 (LO)	0CH	TIMER2 (LO)
0BH	TIMER1 (HI)	0BH	*IOC2
0AH	TIMER1 (LO)	0AH	WATCHDOG
09H	INT_PENDING	09H	INT_PENDING
08H	INT_MASK	08H	INT_MASK
07H	SBUF(RX)	07H	SBUF(TX)
06H	HSI_STATUS	06H	HSO_COMMAND
05H	HSI_TIME (HI)	05H	HSO_TIME (HI)
04H	HSI_TIME (LO)	04H	HSO_TIME (LO)
03H	AD_RESULT (HI)	03H	HSI_MODE
02H	AD_RESULT (LO)	02H	AD_COMMAND
01H	ZERO REG (HI)	01H	ZERO REG (HI)
00H	ZERO REG (LO)	00H	ZERO REG (LO)

WSR = 0

0DH	*T2 CAPTURE (HI)
0CH	*T2 CAPTURE (LO)

WSR = 15

OTHER SFRS IN WSR
15 BECOME READABLE
IF THEY WERE WRITABLE
IN WSR = 0 AND WRITABLE
IF THEY WERE READABLE
IN WSR = 0

*NEW OR CHANGED
REGISTER FUNCTION

USING THE ALTERNATE REGISTER WINDOW (WSR = 15)

I/O register expansion on the new CHMOS members of the MCS-96 family has been provided by making two register windows available. Switching between these windows is done using the Window Select Register (WSR). The PUSHA and POPA instructions can be used to push and pop the WSR and second interrupt mask when entering or leaving interrupts, so it is easy to change between windows.

On the 80C196KA only Window 0 and Window 15 are active. Window 0 is a true superset of the standard 8096 SFR space, while Window 15 allows the read-only registers to be written and write-only registers to be read. The only major exception to this is the Timer2 register which is the Timer2 capture register in Window 15. The writeable register for Timer2 is in Window 0. There are also some minor changes and cautions. The descriptions of the registers which have different functions in Window 15 than in Window 0 are listed below:

AD_COMMAND (02H) — Read the last written command
AD_RESULT (02H, 03H) — Write a value into the result register
HSI_MODE (03H) — Read the value in HSI_MODE
HSI_TIME (04H,05H) — Write to FIFO Holding register
HSO_TIME (04H,05H) — Read the last value placed in the holding register
HSI_STATUS (06H) — Write to status bits but not to HSI pin bits. (Pin bits are 1,3,5,7).
HSO_COMMAND (06H) — Read the last value placed in the holding register
SBUF(RX) (07H) — Write a value into the receive buffer
SBUF(TX) (07H) — Read the last value written to the transmit buffer
WATCHDOG(0AH) — Read the value in the upper byte of the WDT
TIMER1 (0AH,0BH) — Write a value to Timer1
TIMER2 (0CH,0DH) — Read/Write the Timer2 capture register.
Note that Timer2 read/write is done with WSR = 0.
IOC2 (0BH) — Last written value is readable, except bit 7 (note 1)
BAUD_RATE (0EH) — No function, cannot be read
PORT0 (0EH) — No function, no output drivers on the pins
SP_STAT (11H) — Set the status bits, TI and RI can be set, but it will not cause an interrupt
SP_CON (11H) — Read the current control byte
IOS0 (15H) — Writing to this register controls the HSO pins. Bits 6 and 7 are inactive for writes.
IOC0 (15H) — Last written value is readable, except bit 1 (note 1)
IOS1 (16H) — Writing to this register will set the status bits, but not cause interrupts. Bits 6 and 7 are not functional
IOC1 (16H) — Last written value is readable
IOS2 (17H) — Writing to this register will set the status bits, but not cause interrupts.
PWM_CONTROL (17H) — Read the duty cycle value written to PWM_CONTROL

NOTE:
1. IOC2.7 (CAM CLEAR) and IOC0.1 (T2RST) are not latched and will read as a 1 (precharged bus) .

Being able to write to the read-only registers and vice-versa provides a lot of flexibility. One of the most useful advantages is the ability to set the timers and HSO lines for initial conditions other than zero.

SFR BIT SUMMARY

A summary of the SFRs which control I/O functions has been included in this section. The summary is separated into a list of those SFRs which have changed on the 80C196 and a list of those which have remained the same.

The following 80C196 SFRs are different than those on the 8096BH:

(The Read and Write comments indicate the register's function in Window 0 unless otherwise specified.)

SBUF(TX):
07h
write

Now double buffered

BAUD RATE:
0Eh
write

Uses new Baud Rate Values

SP_STAT:
11h
read

7	6	5	4	3	2	1	0
RB8/ RPE	RI	TI	FE	TXE	OE	X	X

RPE :	Receive Parity Error
RI :	Receive Indicator
TI :	Transmit Indicator
FE :	Framing Error
TXE :	Transmitter Empty
OE :	Receive Overrun Error

IPEND1:
IMASK1:
12h,13h
read/write

7	6	5	4	3	2	1	0
NMI	FIFO FULL	EXT INT	T2 OVF	T2 CAP	HSI4	RI	TI

NMI :	Non-Maskable Interrupt
FIFO FULL :	HSIO FIFO full
EXTINT :	External Interrupt Pin
T2OVF :	Timer2 Overflow
T2CAP :	Timer2 Capture
HSI4 :	HSI has 4 or more entries in FIFO
RI :	Receive Interrupt
TI :	Transmit Interrupt

WSR:

7	6	5	4	3	2	1	0
X	X	X	X	W	W	W	W

14h
read/write

WWWW= 0 : SFRs function like a superset of 8096 SFRs

WWWW=15
: Exchange read/write registers

WWWW=OTHER : Undefined, do not use

XXXX=0000B : These bits must always be written as zeros to provide compatibility with future products.

IOS2:

7	6	5	4	3	2	1	0
START A2D	T2 RESET	HSO.5	HSO.4	HSO.3	HSO.2	HSO.1	HSO.0

17h
read

Indicates which HSO event occured

START A2D : HSO_CMD 15, start A to D

T2RESET : HSO_CMD 14, Timer 2 reset

HSO.0-5 : Output pins HSO.0 through HSO.5

IOC2:

7	6	5	4	3	2	1	0
CLEAR CAM	ENA LOCK	T2ALT INT	A2D CPD	NOSH	SLOW PWM	T2UD ENA	FAST T2EN

0Bh
write

CLEAR_CAM : Clear Entire CAM

ENA_LOCK : Enable lockable CAM entry feature

T2ALT INT : Enable T2 Alternate Interrupt at 8000H

A2D_CPD : Clock Prescale Disable for low XTAL frequency (A to D conversion in fewer state times)

NOSH : Disable A/D Sample and Hold

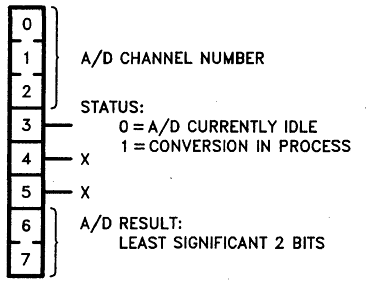
SLOW_PWM : Turn on divide by 2 Prescaler on PWM

T2UD ENA : Enable Timer 2 as up/down counter

FAST_T2EN : Enable Fast increment of T2; once per state time.

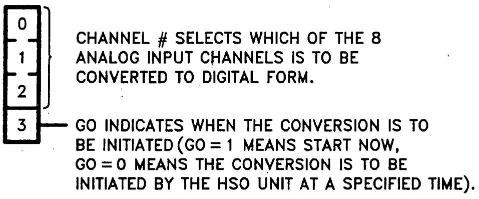
The following registers are the same on the 80C196 as they were on the 8096BH:

A/D Result LO (02H)



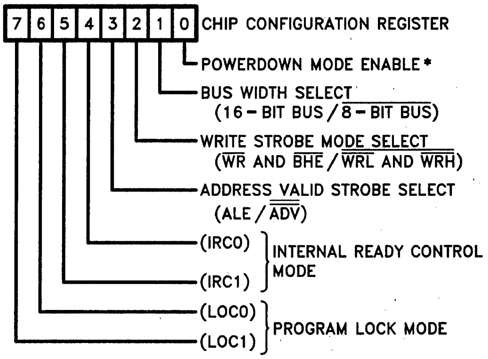
270428-5

A/D Command (02H)


0
1
2
3
CHANNEL # SELECTS WHICH OF THE 8 ANALOG INPUT CHANNELS IS TO BE CONVERTED TO DIGITAL FORM.
GO INDICATES WHEN THE CONVERSION IS TO BE INITIATED (GO = 1 MEANS START NOW, GO = 0 MEANS THE CONVERSION IS TO BE INITIATED BY THE HSO UNIT AT A SPECIFIED TIME).

270428-6

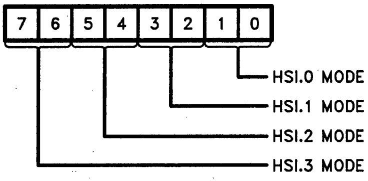
Chip Configuration (2108H)


7 6 5 4 3 2 1 0
CHIP CONFIGURATION REGISTER
POWERDOWN MODE ENABLE*
BUS WIDTH SELECT (16-BIT BUS/$\overline{8\text{-BIT BUS}}$)
WRITE STROBE MODE SELECT ($\overline{WR}$ AND $\overline{BHE}$/$\overline{WRL}$ AND $\overline{WRH}$)
ADDRESS VALID STROBE SELECT (ALE/$\overline{ADV}$)
(IRC0)
(IRC1)
INTERNAL READY CONTROL MODE
(LOC0)
(LOC1)
PROGRAM LOCK MODE

270428-7

*Minor Change

HSI_Mode (03H)

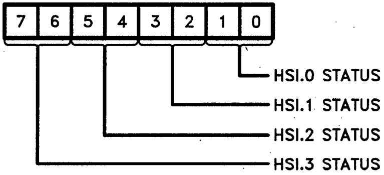


WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00	8 POSITIVE TRANSITIONS
01	EACH POSITIVE TRANSITION
10	EACH NEGATIVE TRANSITION
11	EVERY TRANSITION (POSITIVE AND NEGATIVE)

270428-8

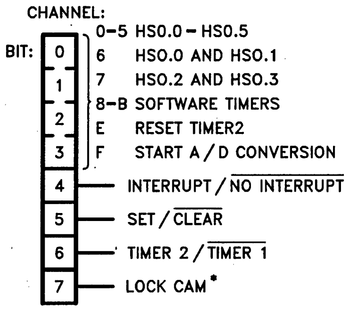
HSI_Status (06H)



WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

270428-9

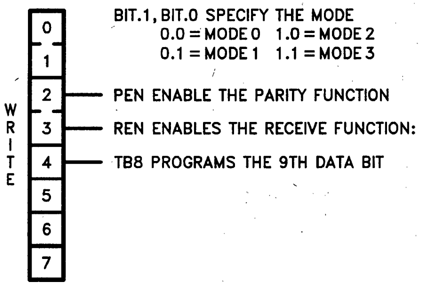
HSO Command (06H)



270428-10

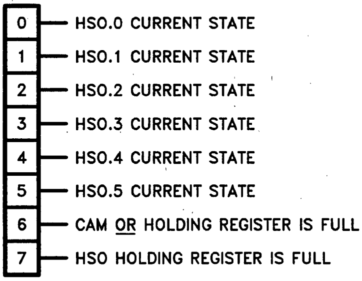
*Minor Change

SPCON (11H)


0
1
2
3
4
5
6
7
WRITE
BIT.1, BIT.0 SPECIFY THE MODE
0.0 = MODE 0 1.0 = MODE 2
0.1 = MODE 1 1.1 = MODE 3
PEN ENABLE THE PARITY FUNCTION
REN ENABLES THE RECEIVE FUNCTION:
TB8 PROGRAMS THE 9TH DATA BIT

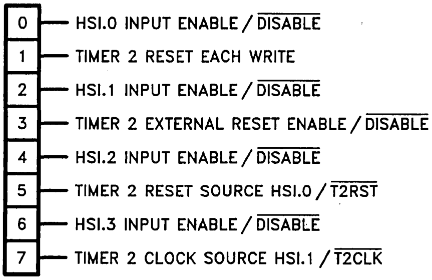
270428-11

IOS0 (15H)



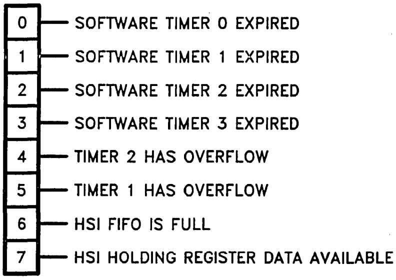
270428-12

IOC0 (15H)


0 — HSI.0 INPUT ENABLE / $\overline{\text{DISABLE}}$
1 — TIMER 2 RESET EACH WRITE
2 — HSI.1 INPUT ENABLE / $\overline{\text{DISABLE}}$
3 — TIMER 2 EXTERNAL RESET ENABLE / $\overline{\text{DISABLE}}$
4 — HSI.2 INPUT ENABLE / $\overline{\text{DISABLE}}$
5 — TIMER 2 RESET SOURCE HSI.0 / $\overline{\text{T2RST}}$
6 — HSI.3 INPUT ENABLE / $\overline{\text{DISABLE}}$
7 — TIMER 2 CLOCK SOURCE HSI.1 / $\overline{\text{T2CLK}}$

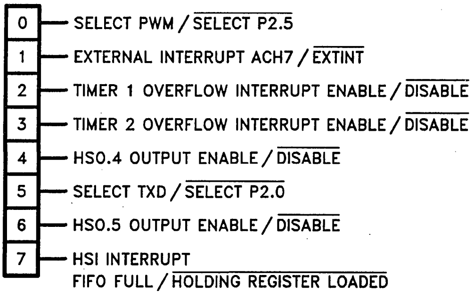
270428–13

IOS1 (16H)


0 — SOFTWARE TIMER 0 EXPIRED
1 — SOFTWARE TIMER 1 EXPIRED
2 — SOFTWARE TIMER 2 EXPIRED
3 — SOFTWARE TIMER 3 EXPIRED
4 — TIMER 2 HAS OVERFLOW
5 — TIMER 1 HAS OVERFLOW
6 — HSI FIFO IS FULL
7 — HSI HOLDING REGISTER DATA AVAILABLE

270428–14

IOC1 (16H)


0 — SELECT PWM / $\overline{\text{SELECT P2.5}}$
1 — EXTERNAL INTERRUPT ACH7 / $\overline{\text{EXTINT}}$
2 — TIMER 1 OVERFLOW INTERRUPT ENABLE / $\overline{\text{DISABLE}}$
3 — TIMER 2 OVERFLOW INTERRUPT ENABLE / $\overline{\text{DISABLE}}$
4 — HSO.4 OUTPUT ENABLE / $\overline{\text{DISABLE}}$
5 — SELECT TXD / $\overline{\text{SELECT P2.0}}$
6 — HSO.5 OUTPUT ENABLE / $\overline{\text{DISABLE}}$
7 — HSI INTERRUPT
FIFO FULL / $\overline{\text{HOLDING REGISTER LOADED}}$

270428–15

Port 2 Multiple Functions

Pin	Func.	Alternative Function	Control Reg.
2.0	Output	TXD (Serial Port Transmit)	IOC1.5
2.1	Input	RXD (Serial Port Receive)	SPCON.3
2.3	Input	T2CLK (Timer2 Clock & Baud)	IOC0.7
2.4	Input	T2RST (Timer2 Reset)	IOC0.5
2.5	Output	PWM Output	IOC1.0
2.6	QBD*	Timer2 up/ down select	IOC2.1
2.7	QBD*	Timer2 Capture	N/A

*QBD = Quasi-bidirectional

Baud Rate Calculations

Asynchronous Modes 1, 2 and 3:

$$\text{Baud\_Reg} = \frac{\text{XTAL1}}{\text{Baud Rate} \times 16} - 1 \text{ OR } \frac{\text{T2CLK}}{\text{Baud Rate} \times 8}$$

Synchronous Mode 0:

$$\text{Baud\_Reg} = \frac{\text{XTAL1}}{\text{Baud Rate} \times 2} - 1 \text{ OR } \frac{\text{T2CLK}}{\text{Baud Rate}}$$

Baud Rates and Baud Register Values

Baud Rate	XTAL Frequency		
	8.0 MHz	10.0 MHz	12.0 MHz
300	1666/−0.02	2082/0.02	2499/0.00
1200	416/−0.08	520/−0.03	624/0.00
2400	207/0.16	259/0.16	312/−0.16
4800	103/−0.16	129/0.16	155/0.16
9600	51/−0.16	64/0.16	77/0.16
19.2K	25/0.16	32/1.40	38/0.16

Baud Register Value/% Error

A maximum baud rate of 750 Kbaud is available in the asynchronous modes with 12 MHz on XTAL1. The synchronous mode has a maximum rate of 3.0 Mbaud with a 12 MHz clock. Location 0EH is the Baud Register. It is loaded sequentially in two bytes, with the low byte being loaded first. This register may not be loaded with zero in serial port Mode 0.

ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings*

Ambient Temperature Under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
Voltage On Any Pin to V_{SS}	−0.5V to +7.0V
Power Dissipation	1.5W

**Notice:* Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

Operating Conditions

Symbol	Description	Min	Max	Units
T_A	Ambient Temperature Under Bias	0	+70	°C
V_{CC}	Digital Supply Voltage	4.5	5.50	V
T_{REF}	Analog Supply Voltage	4.5	5.50	V
f_{OSC}	Oscillator Frequency	3.5	12	MHz

NOTE:
ANGND and V_{SS} should be nominally at the same potential.

D.C. Characteristics (Over specified operating conditions)

Symbol	Description	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage (except XTAL 1)	$0.2\ V_{CC} + 0.9$	$V_{CC} + 0.5$	V	
V_{IH1}	Input High Voltage on XTAL 1	$0.7\ V_{CC}$	$V_{CC} + 0.5$	V	
V_{OL}	Output Low Voltage		0.3 0.45 1.0	V V V	$I_{OL} = 200\ \mu A$ $I_{OL} = 3.2\ mA$ $I_{OL} = 7\ mA$
V_{OH}	Output High Voltage (Standard Outputs)	$V_{CC} - 0.3$ $V_{CC} - 0.7$ $V_{CC} - 1.5$		V V V	$I_{OH} = -200\ \mu A$ $I_{OH} = -3.2\ mA$ $I_{OH} = -7\ mA$
V_{OH1}	Output High Voltage (Quasi-bidirectional Outputs)	$V_{CC} - 0.3$ $V_{CC} - 0.7$ $V_{CC} - 1.5$		V V V	$I_{OH} = -10\ \mu A$ $I_{OH} = -30\ \mu A$ $I_{OH} = -60\ \mu A$
I_{LI}	Input Leakage Current (Std. Inputs)		±10	μA	$0 < V_{IN} < V_{CC} - 0.3V$
I_{LI1}	Input Leakage Current (Port 0)		±3	μA	$0 < V_{IN} < V_{REF}$
I_{TL}	1 to 0 Transition Current (QBD Pins)		−650	μA	$V_{IN} = 2.0V$
I_{IL}	Logical 0 Input Current (QBD Pins)		−50	μA	$V_{IN} = 0.45V$
I_{IL1}	Logical 0 Input Current in Reset (ALE, RD, WR, BHE, INST, P2.0)		−500	μA	$V_{IN} = 0.45\ V$

D.C. Characteristics (Over specified operating conditions) (Continued)

Symbol	Description	Min	Max	Units	Test Conditions
I_{CC}	Active Mode Current in Reset		60	mA	XTAL1 = 12 MHz $V_{CC} = V_{PP} = V_{REF} = 5.5V$
I_{REF}	A/D Converter Reference Current		5	mA	
I_{idle}	Idle Mode Current		22	mA	
I_{CC1}	Active Mode Current (Typical)		15	mA	XTAL1 = 3.5 MHz
I_{PD}	Powerdown Mode Current		TBD	μA	$V_{CC} = V_{PP} = V_{REF} = 5.5V$
R_{RST}	Reset Pullup Resistor	6K	50K	Ω	
C_S	Pin Capacitance (Any Pin to V_{SS})		10	pF	$f_{TEST} = 1.0$ MHz

NOTES:

1. QBD (Quasi-bidirectional) pins include Port 1, P2.6 and P2.7.
2. Standard Outputs include all bus pins (data and control), HSO pins, PWM/P2.5, CLKOUT, RESET, Ports 3 and 4, TXD/P2.0, and RXD (in serial mode 0). The V_{OH} specification is not valid for RESET. Ports 3 and 4 are open-drain outputs, which will be available on future ROM and EPROM parts.
3. Standard Inputs include HSI pins, CDE, EA, READY, BUSWIDTH, NMI, RXD/P2.1, EXTINT/P2.2, T2CLK/P2.3, and T2RST/P2.4.
4. Maximum current per pin must be externally limited to the following values if V_{OL} is held above 0.45V or V_{OH} is held below V_{CC} − 0.7V:

   I_{OL} on Output pins: 10 mA I_{OH} on quasi-bidirectional pins: self limiting
   I_{OH} on Standard Output pins: 10 mA
5. Maximum current per bus pin (data and control) during normal operation is ±3.2 mA.
6. During normal (non-transient) conditions the following total current limits apply to each group of pins:

Port 1, P2.6	I_{OL}: 29 mA	I_{OH} is self limiting
HSO, P2.0, RXD, $\overline{\text{RESET}}$	I_{OL}: 29 mA	I_{OH}: 26 mA
P2.7, P2.5, $\overline{\text{WR}}$, $\overline{\text{BHE}}$	I_{OL}: 13 mA	I_{OH}: 11 mA
AD0–AD15	I_{OL}: 52 mA	I_{OH}: 52 mA
$\overline{\text{RD}}$, ALE, INST-CLKOUT	I_{OL}: 13 mA	I_{OH}: 13 mA

A.C. Characteristics (Over specified operating conditions)

These are ADVANCED specifications, the parameters may change before Intel releases the product for sale.

Test Conditions: Capacitive load on all pins = 100 pF, Rise and fall times = 10 ns, f_{OSC} = 12 MHz

The system must meet these specifications to work with the 80C196:

Symbol	Description	Min	Max	Units	Notes
T_{AVYV}	Address Valid to READY Setup		$2T_{OSC} - 55$	ns	
T_{LLYV}	ALE Low to READY Setup		$T_{OSC} - 55$	ns	
T_{YLYH}	NonREADY Time	No upper limit		ns	
T_{CLYX}	READY Hold after CLKOUT Low	0	$T_{OSC} - 30$	ns	(Note 2)
T_{LLYX}	READY Hold after ALE Low	$T_{OSC}+5$	$2T_{OSC}-40$	ns	(Note 2)
T_{AVGV}	Address Valid to Buswidth Setup		$2T_{OSC} - 55$	ns	
T_{LLGV}	ALE Low to Buswidth Setup		$T_{OSC} - 55$	ns	
T_{CLGX}	Buswidth Hold after CLKOUT Low	0		ns	
T_{AVDV}	Address Valid to Input Data Valid		$3T_{OSC} - 60$	ns	
T_{RLDV}	RD# Active to Input Data Valid		$T_{OSC} - 25$	ns	
T_{CLDV}	CLKOUT Low to Input Data Valid		$T_{OSC} - 55$	ns	
T_{RHDZ}	End of RD# to Input Data Float		$T_{OSC} - 20$	ns	
T_{RXDX}	Data Hold after RD# Inactive	0		ns	

NOTES:

1. Typical specification, not guaranteed.
2. If max is exceeded, additional wait states will occur.

A.C. Characteristics (Over specified operating conditions) (Continued)

These are ADVANCED specifications, the parameters may change before Intel releases the product for sale.

Test Conditions: Capacitive load on all pins = 100 pF, Rise and fall times = 10 ns, f_{OSC} = 12 MHz

The 80C196KA will meet these specifications:

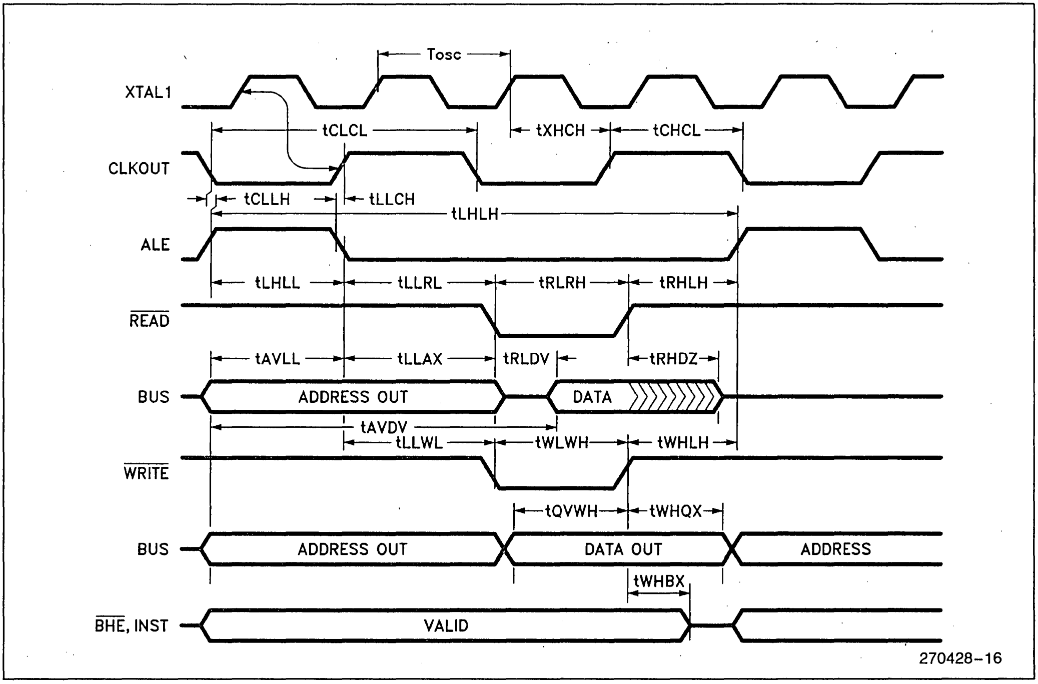
Symbol	Description	Min	Max	Units	Notes
F_{XTAL}	Frequency on XTAL1	3.5	12.0	MHz	
T_{OSC}	$1/F_{XTAL}$	83	286	ns	
T_{XHCH}	XTAL1 High to CLKOUT High or Low	40	110	ns	(Note 1)
T_{CLCL}	CLKOUT Cycle Time	$2T_{OSC}$		ns	
T_{CHCL}	CLKOUT High Period	$T_{OSC} - 10$	$T_{OSC} + 10$	ns	
T_{CLLH}	CLKOUT Falling Edge to ALE Rising	−10	10	ns	
T_{LLCH}	ALE Falling Edge to CLKOUT Rising	−10	10	ns	
T_{LHLH}	ALE Cycle Time	$4T_{OSC}$		ns	
T_{LHLL}	ALE High Period	$T_{OSC} - 10$	$T_{OSC} + 10$	ns	
T_{AVLL}	Address Setup to ALE Falling Edge	$T_{OSC} - 25$		ns	
T_{LLAX}	Address Hold after ALE Falling Edge	$T_{OSC} - 15$		ns	
T_{LLRL}	ALE Falling Edge to $\overline{RD}$ Falling Edge	$T_{OSC} - 25$		ns	
T_{RLCL}	$\overline{RD}$ Falling Edge to CLKOUT Falling Edge	0	20	ns	
T_{RLRH}	$\overline{RD}$ Low Period	$T_{OSC} - 5$		ns	
T_{RHLH}	$\overline{RD}$ Rising Edge to ALE Rising Edge	$T_{OSC} - 15$	$T_{OSC} + 15$	ns	(Note 2)
T_{LLWL}	ALE Falling Edge to $\overline{WR}$ Falling Edge	$T_{OSC} - 10$		ns	
T_{CLWL}	CLKOUT Low to $\overline{WR}$ Falling Edge	−5	15	ns	
T_{QVWH}	Data Stable to $\overline{WR}$ Rising Edge	$T_{OSC} - 20$		ns	
T_{CHWH}	CLKOUT Rising Edge to $\overline{WR}$ Rising Edge	−10	10	ns	
T_{WLWH}	$\overline{WR}$ Low Period	$T_{OSC} - 20$		ns	
T_{WHQX}	Data Hold after $\overline{WR}$ Rising Edge	$T_{OSC} - 20$		ns	
T_{WHLH}	$\overline{WR}$ Rising Edge to ALE Rising Edge	$T_{OSC} - 20$	$T_{OSC} + 20$	ns	(Note 2)
T_{WHBX}	$\overline{BHE}$, INST HOLD after $\overline{WR}$ Rising Edge	$T_{OSC} - 30$		ns	

NOTES:

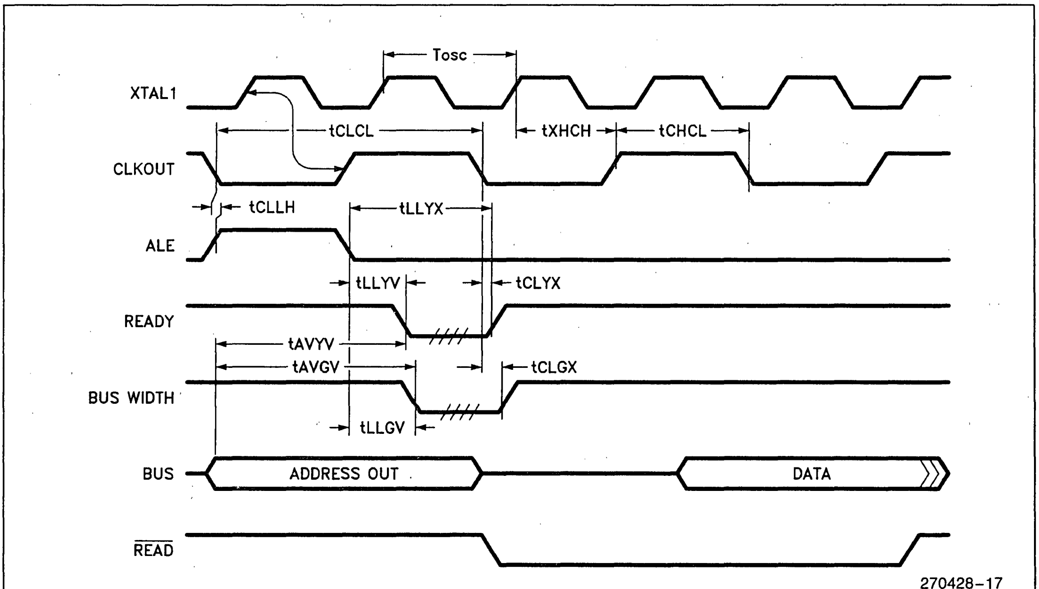
T_{OSC} = 83.3 ns at 12 MHz; T_{OSC} = 125 ns at 8 MHz.

1. Typical specification, not guaranteed.
2. Assuming back-to-back bus cycles.

System Bus Timings



270428–16

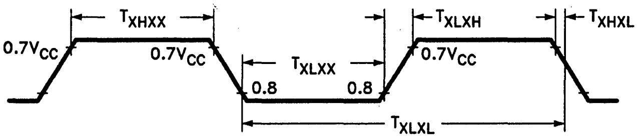


270428–17

EXTERNAL CLOCK DRIVE

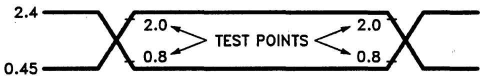
Symbol	Parameter	Min	Max	Units
$1/T_{XLXL}$	Oscillator Frequency	3.5	12	MHz
T_{XLXL}	Oscillator Period (T_{OSC})	83	286	ns
T_{XHXX}	High Time	32		ns
T_{XLXX}	Low Time	32		ns
T_{XLXH}	Rise Time		10	ns
T_{XHXL}	Fall Time		10	ns

EXTERNAL CLOCK DRIVE WAVEFORMS



270428-18

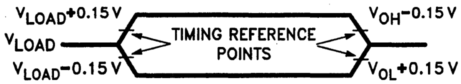
A.C. TESTING INPUT, OUTPUT WAVEFORM



270428-19

A.C. Testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0" Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

FLOAT WAVEFORM



270428-20

For Timing Purposes a Port Pin is no Longer Floating when a 100 mV change from Load Voltage Occurs and Begins to Float when a 100 mV change from the Loaded V_{OH}/V_{OL} Level occurs I_{OL}/I_{OH} = ±15 mA.

EXPLANATION OF AC SYMBOLS

Each symbol is two pairs of letters prefixed by "T" for time. The characters in a pair indicate a signal and its condition, respectively. Symbols represent the time between the two signal/condition points.

Conditions:

H - High
L - Low
V - Valid
X - No Longer Valid
Z - Floating

Signals:

A - Address
B - $\overline{BHE}$
C - CLKOUT
D - DATA
G - Buswidth
L - ALE/$\overline{ADV}$
R - $\overline{RD}$
W - $\overline{WR}/\overline{WRH}/\overline{WRL}$
X - XTAL1
Y - READY

A TO D CHARACTERISTICS

There are four modes of A/D operation.

In Modes 2 and 3 the maximum XTAL1 frequency is 10.0 MHz. Accuracy will degrade at higher frequencies.

A/D CONVERTER SPECIFICATIONS

The absolute conversion accuracy is dependent on the accuracy of V_{REF}. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at V_{REF} = 5.120V, 10.0 MHz, A/D Mode 2.

	Clock Prescaler On **IOC2.4 = 0**	**Clock Prescaler Off** **IOC2.4 = 1**	
IOC2.3 = 0 **with S&H**	Mode 0–158 States 26.33 µs @ 12 MHz	Mode 2–91 States 22.75 µs @ 8 MHz	91 States 18.2 µs @ 10 MHz
IOC2.3 = 1 **without S&H**	Mode 1–293 States 48.83 µs @ 12 MHz	Mode 3–163 States 40.75 µs @ 8 MHz	163 States 32.6 µs @ 10 MHz

Parameter	Typical*(1)	Minimum	Maximum	Units**	Notes
Resolution		256(5)	1024 10	Levels Bits	5
Absolute Error		0	±4	LSBs	
Full Scale Error	−0.5 ±0.5			LSBs	
Zero Offset Error	±0.5			LSBs	
Non-Linearity		0	±4	LSBs	
Differential Non-Linearity		0	±2	LSBs	5
Channel-to-Channel Matching		0	±1	LSBs	
Repeatability	±0.25			LSBs	1
Temperature Coefficients:					
Offset	0.009			LSB/°C	1
Full Scale	0.009			LSB/°C	1
Differential Non-Linearity	0.009			LSB/°C	1
Off Isolation		−60		dB	1,2,4
Feedthrough	−60			dB	1,2
V_{CC} Power Supply Rejection	−60			dB	1,2
Input Resistance		1K	5K	Ω	1
D.C. Input Leakage		0	3.0	µA	

NOTES:
* These values are expected for most parts at 25°C.
**An "LSB", as used here, has a value of approximately 5 mV.
1. These values are not tested in production and are based on theoretical estimates and laboratory tests.
2. DC to 100 KHz.
3. For starting the A/D with an HSO Command.
4. Multiplexer Break-Before-Make Guaranteed.
5. See functional deviations list.

A/D GLOSSARY OF TERMS

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An actual characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversions under the same conditions.

BREAK-BEFORE-MAKE—The property of a multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g. the converter will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE CENTER—The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION—The point at which the converter changes from an output code of Q, to a code of Q + 1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK—See "Off-Isolation".

D.C. INPUT LEAKAGE—Leakage current to ground from an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D Converter after the sample window closes.

FULL SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at V_{IN} = 0.5 LSB, its last code transition at V_{IN} = (V_{REF} − 1.5 LSB) and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB—Least Significant Bit: The voltage corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For an 8-bit converter with a reference voltage of 5.12V, one LSB is 20 mV. Note that this is different than digital LSBs, since an uncertainty of two LSB, when referring to an A/D converter, equals 40 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 80 mV.)

MONOTONIC—The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES—For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY—The maximum deviation of code transitions of the terminal based characteristic from the corresponding code transitions of the ideal characteristic.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY—The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY—The variation in the sample delay.

SAMPLE TIME—The time that the sample window is open.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION—An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An actual characteristic which has been rotated and translated to remove zero offset and full scale error.

V_{CC} REJECTION—Attenuation of noise on the V_{CC} line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

80C196KA FUNCTIONAL DEVIATIONS

The 80C196KA has the following problems. We are working on, or have already defined, silicon fixes for all these problems.

1. Byte shifts on odd addresses do not work properly (SHRB and SHLB). Byte shifts can be done on even addresses, and word and long shifts work correctly.
2. The Unsigned Divide operations (Byte and Word), may result in a quotient that is one count larger than the correct value (DIVU and DIVUB). This can only occur if the most significant bit of the divisor is a one. The problem will not always occur if the MSB is one, and determining if the problem will occur or not is very difficult.
3. The current in the power down mode is on the order of 1 milliamp.
4. The PUSHA instruction works properly with internal stack. When external stack is used, the PUSHA instruction will cause the data to be written into the location pointed to by the lower byte of the stack pointer. Since the PUSHA instruction is simply a fast way of doing a PUSHF, and pushing WSR/IMASK1 and clearing IMASK1, a macro can be written to work around this problem.
5. The A/D converter differential non-linearity error becomes larger as V_{in} approaches V_{ref}. This results in the potential for missed codes at 10-bit resolution.
6. The reset pin must have a rise time less than 4 state times. An External Schmitt trigger reset circuit is recommended. A capacitor only or RC circuit directly connected to the pin will not work reliably. If a bad reset occurs, the chip will lock-up. A good reset will cause the part to work correctly; the chip does not have to be powered on and off.

NOTE:
Instruction bugs 1, 2, and 4 may prevent high level language compilers from generating code which works correctly. If a problem is suspected, generate an assembler code output of the high level language and examine the listing for the above instructions. If any of the instructions are present, the code may have to be rewritten.

DIFFERENCES BETWEEN THE 80C196KA AND THE 8096BH

CONVERTING FROM OTHER MCS®-96 PRODUCTS TO THE 80C196KA

The following list of suggestions for designing an 8X9XBH system will yield a design that is easily converted to the 80C196KA.

1. Do not base critical timing loops on instruction or peripheral execution times.
2. Use equate statements to set all timing parameters, including the baud rate.
3. Do not base hardware timings on CLKOUT or XTAL1. The timings of the 80C196KA are different than those of the 8X9XBH, but they will function with standard ROM/EPROM/Peripheral type memory systems.
4. Make sure all inputs are tied high or low and not left floating.
5. On the 8X9XBH, the $\overline{\text{WRL}}/\overline{\text{WR}}$ and $\overline{\text{WRH}}/\overline{\text{BHE}}$ signals both go low for byte writes to odd addresses in 8-bit write strobe mode. On the 80C196KA, only the $\overline{\text{WRH}}/\overline{\text{BHE}}$ signal goes low for this type of operation.
6. Indexed and indirect operations relative to the stack pointer (SP) work differently on the 80C196KA than on the 8096. On the 8096, the address is calculated based on the un-updated version of the stack pointer. The 80C196KA uses the updated version. The offset for PUSH[SP], POP[SP], PUSH nn[SP] and POP nn[SP] instructions may need to be changed by a count of 2.

NEW FEATURE SUMMARY

CPU FEATURES

Divide by 2 instead of divide by 3 clock for 1.5X performance

Faster instructions, especially indexed/indirect data operations

2.33 μs 16 × 16 multiply with 12 MHz clock (was 6.25 μs)

Faster interrupt response (almost twice as fast)

Different Reset Sequence

Powerdown and Idle Modes

Clock Failure Detect

6 new instructions including Compare Long and Block Move

8 new interrupt vectors

PERIPHERAL FEATURES

SFR Window switching allows read-only registers to be written and vice-versa

Timer2 can count up and down by external selection

Timer2 has an independent capture register

HSO lines which transitioned are saved

HSO lines can be written directly

HSO has CAM Lock and CAM Clear commands

A to D has a selectable sample and hold and speed control

New Baud Rate values are needed for serial port, higher speeds possible in all modes

Double buffered serial port transmit register

Serial Port Receive Overrun and Framing Error Detection

PWM has a Divide-by-2 Prescaler

APPLICATION NOTE

AP-248

September 1987

Using The 8096

IRA HORDEN
MCO APPLICATIONS ENGINEER

Order Number: 270061-002

1.0 INTRODUCTION

High speed digital signals are frequently encountered in modern control applications. In addition, there is often a requirement for high speed 16-bit and 32-bit precision in calculations. The MCS®-96 product line, generically referred to as the 8096, is designed to be used in applications which require high speed calculations and fast I/O operations.

The 8096 is a 16-bit microcontroller with dedicated I/O subsystems and a complete set of 16-bit arithmetic instructions including multiply and divide operations. This Ap-note will briefly describe the 8096 in section 2, and then give short examples of how to use each of its key features in section 3. The concluding sections feature a few examples which make use of several chip features simultaneously and some hardware connection suggestions. Further information on the 8096 and its use is available from the sources listed in the bibliography.

2.0 8096 OVERVIEW

2.1. General Description

Unlike microprocessors, microcontrollers are generally optimized for specific applications. Intel's 8048 was optimized for general control tasks while the 8051 was optimized for 8-bit math and single bit boolean operations. The 8096 has been designed for high speed/high performance control applications. Because it has been designed for these applications the 8096 architecture is different from that of the 8048 or 8051.

There are two major sections of the 8096; the CPU section and the I/O section. Each of these sections can be subdivided into functional blocks as shown in Figure 2-1.

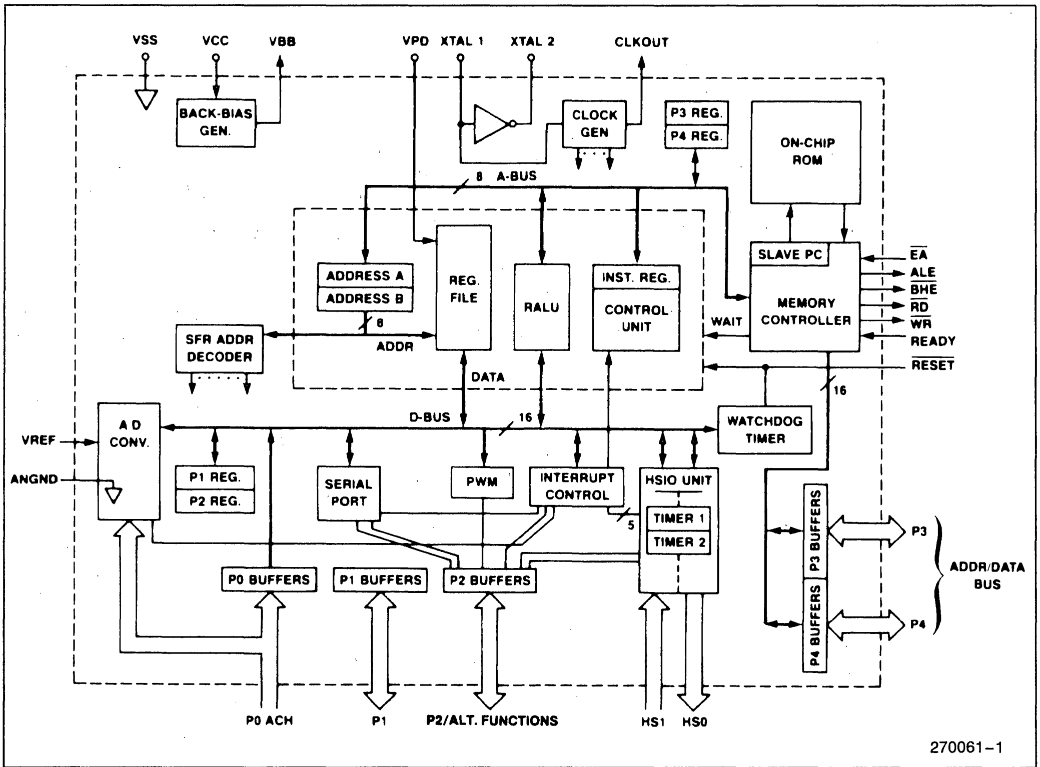
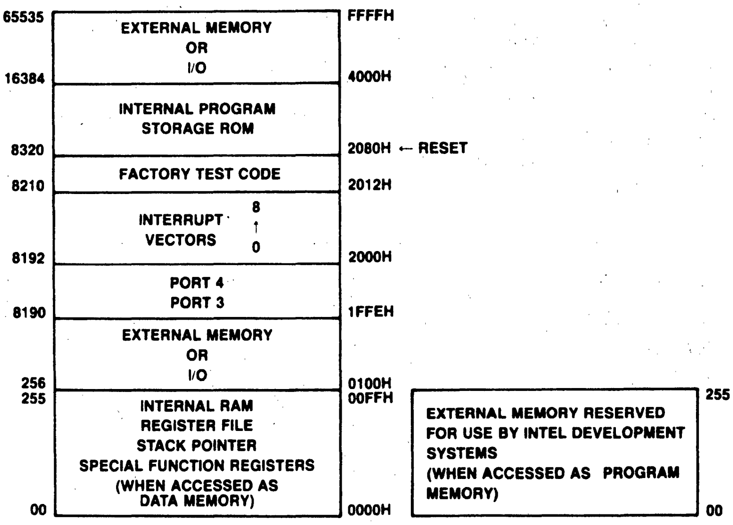


Figure 2-1. 8096 Block Diagram

2.1.1. CPU SECTION

The CPU of the 8096 uses a 16-bit ALU which operates on a 256-byte register file instead of an accumulator. Any of the locations in the register file can be used for sources or destinations for most of the instructions. This is called a register to register architecture. Many of the instructions can also use bytes or words from anywhere in the 64K byte address space as operands. A memory map is shown in Figure 2-2.

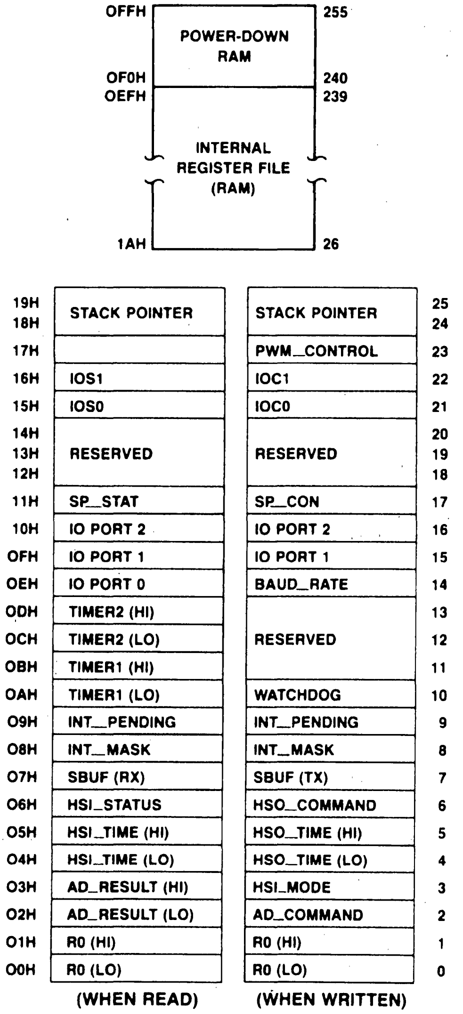
In the lower 24 bytes of the register file are the register-mapped I/O control locations, also called Special Function Registers or SFRs. These registers are used to control the on-chip I/O features. The remaining 232 bytes are general purpose RAM, the upper 16 of which can be kept alive using a low current power-down mode.



270061-2

Figure 2-2. Memory Map

Figure 2-3 shows the layout of the register mapped I/O. Some of these registers serve two functions, one if they are read from and another if they are written to. More information about the use of these registers is included in the description of the features which they control.



	(WHEN READ)	(WHEN WRITTEN)	
19H 18H	STACK POINTER	STACK POINTER	25 24
17H		PWM_CONTROL	23
16H	IOS1	IOC1	22
15H	IOS0	IOC0	21
14H 13H 12H	RESERVED	RESERVED	20 19 18
11H	SP_STAT	SP_CON	17
10H	IO PORT 2	IO PORT 2	16
0FH	IO PORT 1	IO PORT 1	15
0EH	IO PORT 0	BAUD_RATE	14
0DH	TIMER2 (HI)	RESERVED	13
0CH	TIMER2 (LO)		12
0BH	TIMER1 (HI)		11
0AH	TIMER1 (LO)	WATCHDOG	10
09H	INT_PENDING	INT_PENDING	9
08H	INT_MASK	INT_MASK	8
07H	SBUF (RX)	SBUF (TX)	7
06H	HSI_STATUS	HSO_COMMAND	6
05H	HSI_TIME (HI)	HSO_TIME (HI)	5
04H	HSI_TIME (LO)	HSO_TIME (LO)	4
03H	AD_RESULT (HI)	HSI_MODE	3
02H	AD_RESULT (LO)	AD_COMMAND	2
01H	R0 (HI)	R0 (HI)	1
00H	R0 (LO)	R0 (LO)	0

270061-3

Figure 2-3: SFR Layout

2.1.2. I/O FEATURES

Many of the I/O features on the 8096 are designed to operate with little CPU intervention. A list of the major I/O functions is shown in Figure 2-4. The Watchdog Timer is an internal timer which can be used to reset the system if the software fails to operate properly. The Pulse-Width-Modulation (PWM) output can be used as a rough D to A, a motor driver, or for many other purposes. The A to D converter (ADC) has 8 multiplexed inputs and 10-bit resolution. The serial port has several modes and its own baud rate generator. The High Speed I/O section includes a 16-bit timer, a 16-bit counter, a 4-input programmable edge detector, 4 software timers, and a 6-output programmable event generator. All of these features will be described in section 2.3.

2.2. The Processor Section

2.2.1. OPERATIONS AND ADDRESSING MODES

The 8096 has 100 instructions, some of which operate on bits, some on bytes, some on words and some on longs (double words). All of the standard logical and arithmetic functions are available for both byte and word operations. Bit operations and long operations are provided for some instructions. There are also flag manipulation instructions as well as jump and call instructions. A full set of conditional jumps has been included to speed up testing for various conditions.

Bit operations are provided by the Jump Bit and Jump Not Bit instructions, as well as by immediate masking of bytes. These bit operations can be performed on any of the bytes in the register file or on any of the special function registers. The fast bit manipulation of the SFRs can provide rapid I/O operations.

A symmetric set of byte and word operations make up the majority of the 8096 instruction set. The assembly language for the 8096 (ASM-96) uses a "B" suffix on a mnemonic to indicate a byte operation, without this suffix a word operation is indicated. Many of these operations can have one, two or three operands. An example of a one operand instruction would be:

```
NOT  Value1 ;  Value1 : = 1's complement (Value1)
```

A two operand instruction would have the form:

```
ADD  Value2,Value1 ;  Value2 : = Value2 + Value1
```

A three operand instruction might look like:

```
MUL  Value3,Value2,Value1 ;
                         Value3 : = Value2* Value1
```

The three operand instructions combined with the register to register architecture almost eliminate the necessity of using temporary registers. This results in a faster processing time than machines that have equivalent instruction execution times, but use a standard architecture.

Long (32-bit) operations include shifts, normalize, and multiply and divide. The word divide is a 32-bit by 16-bit operation with a 16-bit quotient and 16-bit remainder. The word multiply is a word by word multiply with a long result. Both of these operations can be done in either the signed or unsigned mode. The direct unsigned modes of these instructions take only 6.5 microseconds. A normalize instruction and sticky bit flag have been included in the instruction set to provide hardware support for the software floating point package (FPAL-96).

Major I/O Functions	
High Speed Input Unit	Provides Automatic Recording of Events
High Speed Output Unit	Provides Automatic Triggering of Events and Real-Time Interrupts
Pulse Width Modulation	Output to Drive Motors or Analog Circuits
A to D Converter	Provides Analog Input
Watchdog Timer	Resets 8096 if a Malfunction Occurs
Serial Port	Provides Synchronous or Asynchronous Link
Standard I/O Lines	Provide Interface to the External World when other Special Features are not needed

Figure 2-4. Major I/O Functions

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	✓	✓	✓	✓	↑	—	
ADD/ADDB	3	D ← B + A	✓	✓	✓	✓	↑	—	
ADDC/ADDCB	2	D ← D + A +C	↓	✓	✓	✓	↑	—	
SUB/SUBB	2	D ← D − A	✓	✓	✓	✓	↑	—	
SUB/SUBB	3	D ← B − A	✓	✓	✓	✓	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	✓	✓	✓	↑	—	
CMP/CMPB	2	D − A	✓	✓	✓	✓	↑	—	
MUL/MULU	2	D, D + 2 ← D*A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B*A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D*A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B*A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	✓	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	✓	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	2
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	3
AND/ANDB	2	D ← D and A	✓	✓	0	0	—	—	
AND/ANDB	3	D ← B and A	✓	✓	0	0	—	—	
OR/ORB	2	D ← D or A	✓	✓	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	✓	✓	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ✓	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR (indirect)	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

Figure 2-5. Instruction Summary

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 − D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	✓	✓	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — — lsb ← 0	✓	?	✓	✓	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — — lsb → C	✓	?	✓	0	—	✓	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — — lsb → C	✓	✓	✓	0	—	✓	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left Shift Till msb = 1; D ← shift count	✓	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

Figure 2-5. Instruction Summary (Continued)

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

One operand of most of the instructions can be used with any one of six addressing modes. These modes increase the flexibility and overall execution speed of the 8096. The addressing modes are: register-direct, immediate, indirect, indirect with auto-increment, and long and short indexed.

The fastest instruction execution is gained by using either register direct or immediate addressing. Register-direct addressing is similar to normal direct addressing, except that only addresses in the register file or SFRs can be addressed. The indexed mode is used to directly address the remainder of the 64K address space. Immediate addressing operates as would be expected, using the data following the opcode as the operand.

Both of the indirect addressing modes use the value in a word register as the address of the operand. If the indirect auto-increment mode is used then the word register is incremented by one after a byte access or by two after a word access. This mode is particularly useful for accessing lookup tables.

Access to any of the locations in the 64K address space can be obtained by using the long indexed addressing mode. In this mode a 16-bit 2's complement value is added to the contents of a word register to form the address of the operand. By using the zero register as the index, ASM96 (the assembler) can accept "direct" addressing to any location. The zero register is located at 0000H and always has a value of zero. A short indexed mode is also available to save some time and code. This mode uses an 8-bit 2's complement number as the offset instead of a 16-bit number.

2.2.2. ASSEMBLY LANGUAGE

The multiple addressing modes of the 8096 make it easy to program in assembly language and provide an excellent interface to high level languages. The instructions accepted by the assembler consist of mnemonics followed by either addresses or data. A list of the mnemonics and their functions are shown in Figure 2-5. The addresses or data are given in different formats depending on the addressing mode. These modes and formats are shown in Figure 2-6.

Additional information on 8096 assembly language is available in the MCS-96 Macro Assembler Users Guide, listed in the bibliography.

```
Mnem  Dest or Src1                ; One operand direct
Mnem  Dest, Src1                  ; Two operand direct
Mnem  Dest, Src1, Src2            ; Three operand direct

Mnem  #Src1                       ; One operand immediate
Mnem  Dest, #Src1                 ; Two operand immediate
Mnem  Dest, Src1, #Src2           ; Three operand immediate

Mnem  [addr]                      ; One operand indirect
Mnem  [addr]+                     ; One operand indirect auto-increment
Mnem  Dest, [addr]                ; Two operand indirect
Mnem  Dest, [addr]+               ; Two operand indirect auto-increment
Mnem  Dest, Src1, [addr]          ; Three operand indirect
Mnem  Dest, Src1, [addr]+         ; Three operand indirect auto-increment

Mnem  Dest, offs [addr]           ; Two operand indexed (short or long)
Mnem  Dest, Src1, offs [addr]     ; Three operand indexed (short or long)

Where: "Mnem" is the instruction mnemonic
       "Dest" is the destination register
       "Src1", "Src2" are the source registers
       "addr" is a register containing a value to be used in computing the address of an operand
       "offs" is an offset used in computing the address of an operand
```

270061-B3

Figure 2-6. Instruction Format

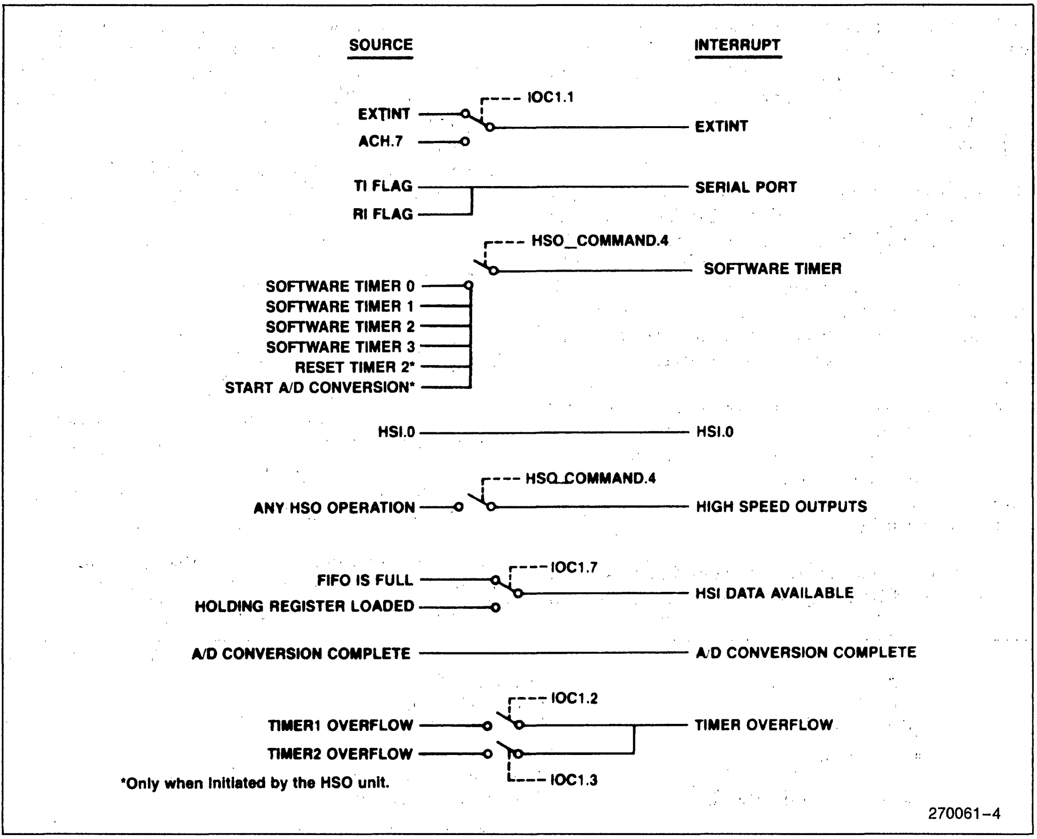


Figure 2-7. Interrupt Sources

2.2.3. INTERRUPTS

The flexibility of the instruction set is carried through into the interrupt system. There are 20 different interrupt sources that can be used on the 8096. The 20 sources vector through 8 locations or interrupt vectors. The vector names and their sources are shown in Figure 2-7, with their locations listed in Figure 2-8. Control of the interrupts is handled through the Interrupt Pending Register (INT_PENDING), the Interrupt Mask Register (INT_MASK), and the I bit in the PSW (PSW.9). Figure 2-9 shows a block diagram of the interrupt structure. The INT_PENDING register contains bits which get set by hardware when an interrupt occurs. If the interrupt mask register bit for that source is a 1 and PSW.9 = 1, a vector will be taken to the address listed in the interrupt vector table for that

Source	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 2-8. Interrupt Vectors and Priorities

source. When the vector is taken the INT_PENDING bit is cleared. If more than one bit is set in the INT_PENDING register with the corresponding bit set in the INT_MASK register, the Interrupt with the highest priority shown in Figure 2-8 will be executed.

The software can make the hardware interrupts work in almost any fashion desired by having each routine run with its own setup in the INT_MASK register. This will be clearly seen in the examples in section 4 which change the priority of the vectors in software. The

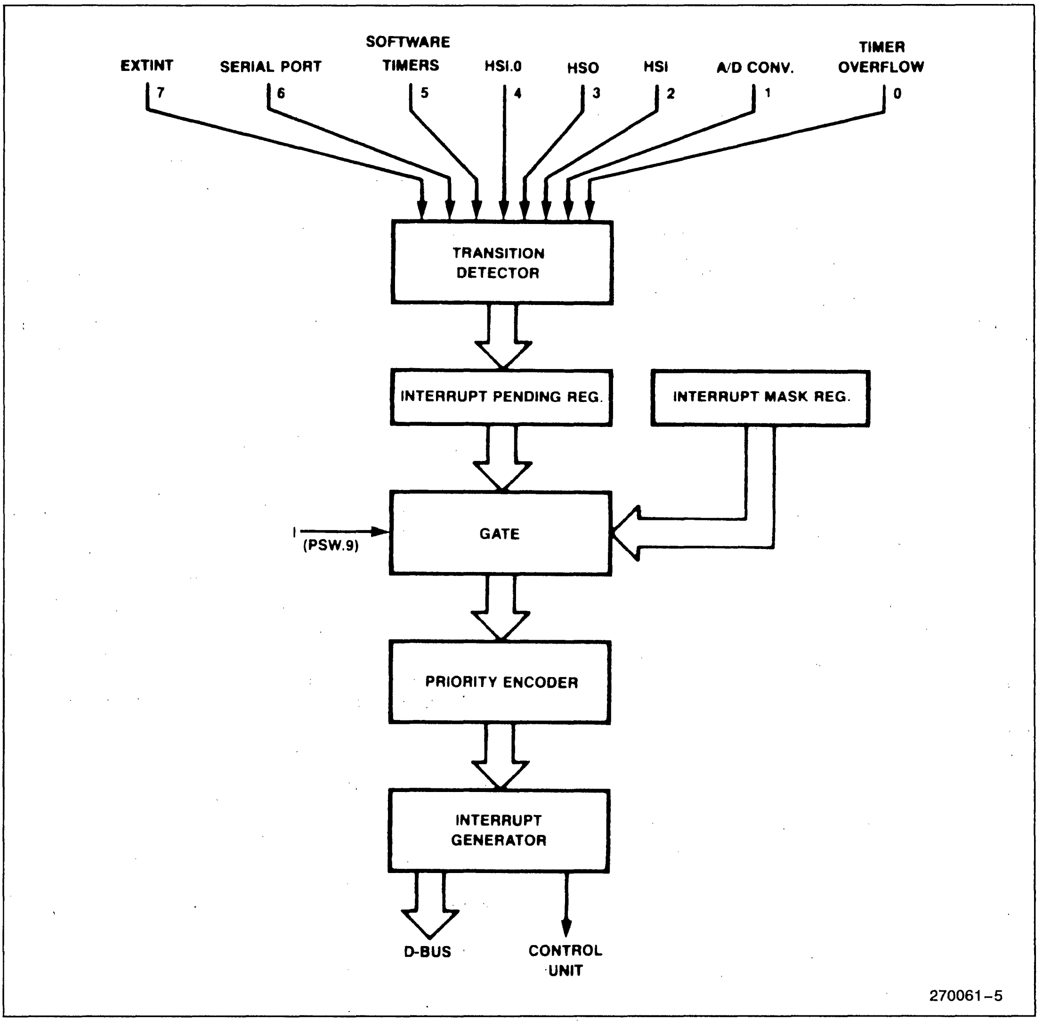


Figure 2-9. Interrupt Structure Block Diagram

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	INT__MASK							

WHERE:

Z is the zero flag. It is set when the result of an operation is zero.

N is the negative flag. It is set to the algebraically correct sign of the result regardless of overflows.

V is the overflow flag. It is set if an overflow occurs.

VT is the overflow trap flag. It is set when the VT flag is set and cleared by JVT, JNVT, or CLRVT.

C is the carry flag. It is set if a carry was generated by the prior operation.

I is the global interrupt enable bit.

ST is the sticky bit. It is set during a right shift if a one was shifted into and then out of the carry flag.

INT__MASK is the interrupt mask register and contains bits which individually enable the 8 interrupt vectors.

Figure 2-10. The PSW Register

PSW (shown in Figure 2-10), stores the INT__MASK register in its lower byte so that the mask register can be pushed and popped along with the machine status when moving in and out of routines. The action of pushing flags clears the PSW which includes PSW.9, the interrupt enable bit. Therefore, after a PUSHF instruction interrupts are disabled. In most cases an interrupt service routine will have the basic structure shown below.

```
INT    VECTOR:

  PUSHF
  LDB  INT_MASK, #xxxxxxxxB
  EI
   -
   -  ;Insert service routine here
   -
  POPF
  RET
```

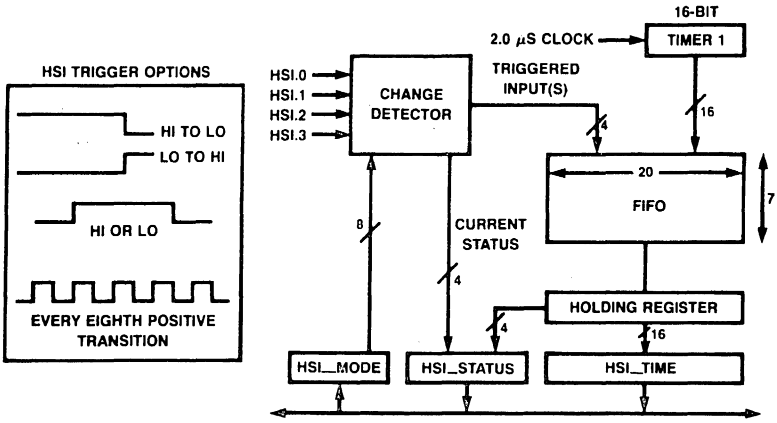
The PUSHF instruction saves the PSW including the old INT__MASK register. The PSW, including the interrupt enable bit are left cleared. If some interrupts need to be enabled while the service routine runs, the INT__MASK is loaded with a new value and interrupts are globally enabled before the service routine continues. At the end of the service routine a POPF instruction is executed to restore the old PSW. The RET instruction is executed and the code returns to the desired location. Although the POPF instruction can enable the interrupts the next instruction will always execute. This prevents unnecessary building of the stack by ensuring that the RET always executes before another interrupt vector is taken.

2.3. On-Chip I/O Section

All of the on-chip I/O features of the 8096 can be accessed through the special function registers, as shown in Figure 2-3. The advantage of using register-mapped I/O is that these registers can be used as the sources or destinations of CPU operations. There are seven major I/O functions. Each one of these will be considered with a section of code to exemplify its usage. The first section covered will be the High Speed I/O, (HSIO), subsystem. This section includes the High Speed Input (HSI) unit, High Speed Output (HSO) unit, and the Timer/Counter section.

2.3.1. TIMER/COUNTERS

The 8096 has two time bases, Timer 1 and Timer 2. Timer 1 is a 16-bit free running timer which is incremented every 8 state times. (A state time is 3 oscillator periods, or 0.25 microseconds with a 12 MHz crystal.)



270061-6

- Pulse measurement with 2.0 μsec resolution
- Input transitions trigger the recording of the reference Timer (16-bit) and triggered input(s) (4-bit)

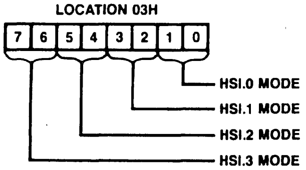
Figure 2-11. HSI Unit Block Diagram

Its value can be read at any time and used as a reference for both the HSI section and the HSO section. Timer 1 can cause an interrupt when it overflows, and cannot be modified or stopped without resetting the entire chip. Timer 2 is really an event counter since it uses an external clock source. Like Timer 1, it is 16-bits wide, can be read at any time, can be used with the HSO section, and can generate an interrupt when it overflows. Control of Timer 2 is limited to incrementing it and resetting it. Specific values can not be written to it.

Although the 8096 has only two timers, the timer flexibility is equal to a unit with many timers thanks to the HSIO unit. The HSI enables one to measure times of external events on up to four lines using Timer 1 as a timer base. The HSO unit can schedule and execute internal events and up to six external events based on the values in either Timer 1 or Timer 2. The 8096 also includes separate, dedicated timers for the baud rate generator and watchdog timer.

2.3.2. HSI

The HSI unit can be thought of as a message taker which records the line which had an event and the time at which the event occurred. Four types of events can trigger the HSI unit, as shown in the HSI block diagram in Figure 2-11. The HSI unit can measure pulse widths and record times of events with a 2



270061-7

Where each 2-bit mode control field defines one of 4 possible modes:

00	8 positive transitions
01	Each positive transition
10	Each negative transition
11	Every transition (positive and negative)

Figure 2-12. HSI Mode Register

microsecond resolution. It can look for one of four events on each of four lines simultaneously, based on the information in the HSI Mode register, shown in Figure 2-12. The information is then stored in a seven level FIFO for later retrieval. Whenever the FIFO contains information, the earliest entry is placed in the holding register. When the holding register is read, the next valid piece of information is loaded into it. Interrupts can be generated by the HSI unit at the time the holding register is loaded or when the FIFO has six or more entries.

2.3.3. HSO

Just as the HSI can be thought of as a message taker, the HSO can be thought of as a message sender. At times determined by the software, the HSO sends mes-

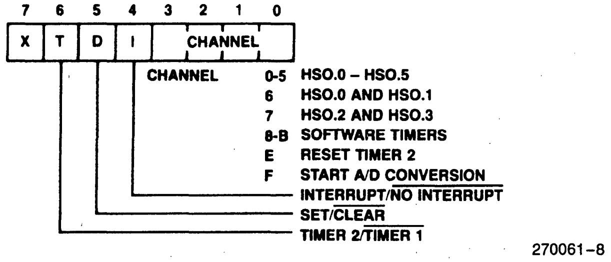


Figure 2-13. HSO Command Register

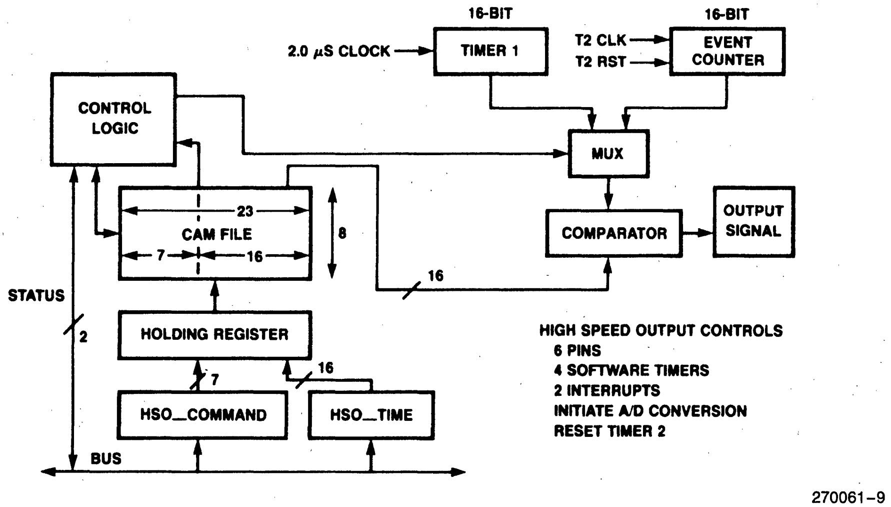


Figure 2-14. HSO Block Diagram

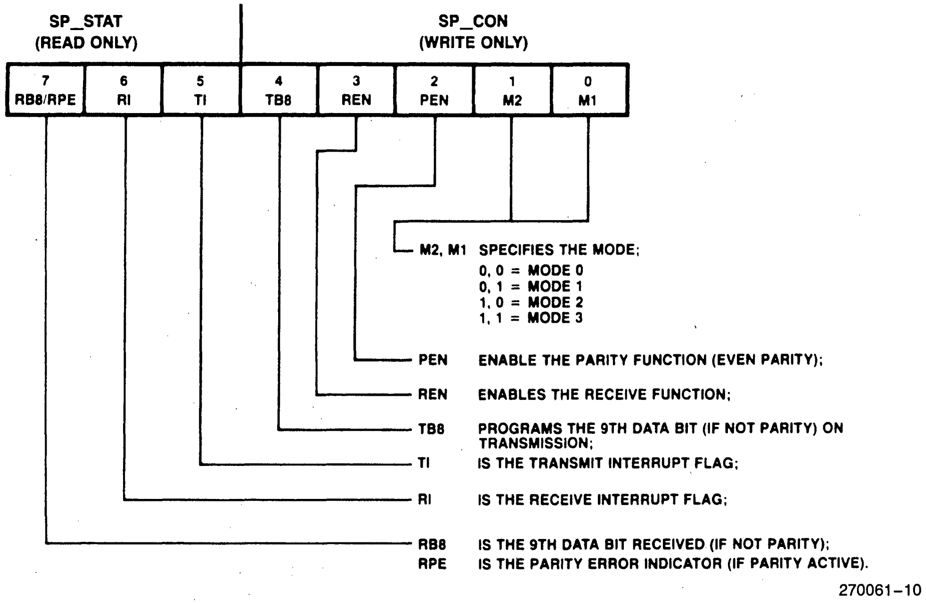
sages to various devices to have them turn on, turn off, start processing, or reset. Since the programmed times can be referenced to either Timer 1 or Timer 2, the HSO makes the two timers look like many. For example, if several events have to occur at specific times, the HSO unit can schedule all of the events based on a single timer. The events that can be scheduled to occur and the format of the command written to the HSO Command register are shown in Figure 2-13.

The software timers listed in the figure are actually 4 software flags in I/O Status Register 1 (IOS1). These flags can be set, and optionally cause an interrupt, at any time based on Timer 1 or Timer 2. In most cases these timers are used to trigger interrupt routines which must occur at regular intervals. A multitask process can easily be set up using the software timers.

A CAM (Content Addressable Memory) file is the main component of the HSO. This file stores up to eight events which are pending to occur. Every state time one location of the CAM is compared with the two timers. After 8 state times, (two microseconds with a 12 MHz clock), the entire CAM has been searched for time matches. If a match occurs the specified event will be triggered and that location of the CAM will be made available for another pending event. A block diagram of the HSO unit is shown in Figure 2-14.

2.3.4. Serial Port

Controlling a device from a remote location is a simple task that frequently requires additional hardware with many processors. The 8096 has an on-chip serial port to reduce the total number of chips required in the system.


SP_STAT
(READ ONLY)
SP_CON
(WRITE ONLY)
7 RB8/RPE
6 RI
5 TI
4 TB8
3 REN
2 PEN
1 M2
0 M1
M2, M1 SPECIFIES THE MODE;
0, 0 = MODE 0
0, 1 = MODE 1
1, 0 = MODE 2
1, 1 = MODE 3
PEN ENABLE THE PARITY FUNCTION (EVEN PARITY);
REN ENABLES THE RECEIVE FUNCTION;
TB8 PROGRAMS THE 9TH DATA BIT (IF NOT PARITY) ON TRANSMISSION;
TI IS THE TRANSMIT INTERRUPT FLAG;
RI IS THE RECEIVE INTERRUPT FLAG;
RB8 IS THE 9TH DATA BIT RECEIVED (IF NOT PARITY);
RPE IS THE PARITY ERROR INDICATOR (IF PARITY ACTIVE).
270061-10

NOTE:
TI and RI are cleared when SP__CON is read.

Figure 2-15. Serial Port Control/Status Register

The serial port is similar to that on the MCS-51 product line. It has one synchronous and three asynchronous modes. In the asynchronous modes baud rates of up to 187.5 Kbaud can be used, while in the synchronous mode rates up to 1.5 Mbaud are available. The chip has a baud rate generator which is independent of Timer 1 and Timer 2, so using the serial port does not take away any of the HSI, HSO or timer flexibility or functionality.

Control of the serial port is provided through the SPCON/SPSTAT (Serial Port CONtrol/Serial Port STATus) register. This register, shown in Figure 2-15, has some bits which are read only and others which are write only. Although the functionality of the port is similar to that of the 8051, the names of some of the modes and control bits are different. The way in which the port is used from a software standpoint is also slightly different since RI and TI are cleared after each read of the register.

The four modes of the serial port are referred to as modes 0, 1, 2 and 3. Mode 0 is the synchronous mode, and is commonly used to interface to shift registers for I/O expansion. In this mode the port outputs a pulse train on the TXD pin and either transmits or receives data on the RXD pin. Mode 1 is the standard asynchronous mode, 8 bits plus a stop and start bit are sent or received. Modes 2 and 3 handle 9 bits plus a stop and start bit. The difference between the two is, that in Mode 2 the serial port interrupt will not be activated unless the ninth data bit is a one; in Mode 3 the interrupt is activated whenever a byte is received. These two modes are commonly used for interprocessor communication.

Using XTAL1:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{4^*(B+1)};\ B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{64^*(B+1)}$$

Using T2CLK:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{B};\ B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{16^*B};\ B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Figure 2-16. Baud Rate Formulas

Baud rates for all of the modes are controlled through the Baud Rate register. This is a byte wide register which is loaded sequentially with two bytes, and internally stores the value as a word. The least significant byte is loaded to the register followed by the most significant. The most significant bit of the baud value determines the clock source for the baud rate generator. If the bit is a one, the XTAL1 pin is used as the source, if it is a zero, the T2 CLK pin is used. The formulas shown in Figure 2-16 can be used to calculate the baud rates. The variable "B" is used to represent the least significant 15 bits of the value loaded into the baud rate register.

The baud rate register values for common baud rates are shown in Figure 2-17. These values can be used when XTAL1 is selected as the clock source for serial modes other than Mode 0. The percentage deviation from theoretical is listed to help assess the reliability of a given setup. In most cases a serial link will work if there is less than a 2.5% difference between the baud rates of the two systems. This is based on the assumption that 10 bits are transmitted per frame and the last bit of the frame must be valid for at least six-eights of the bit time. If the two systems deviate from theoretical by 1.25% in opposite directions the maximum tolerance of 2.5% will be reached. Therefore, caution must be used when the baud rate deviation approaches 1.25% from theoretical. Note that an XTAL1 frequency of 11.0592 MHz can be used with the table values for 11 MHz to provide baud rates that have 0.0 percent deviation from theoretical. In most applications, however, the accuracy available when using an 11 MHz input frequency is sufficient.

Serial port Mode 1 is the easiest mode to use as there is little to worry about except initialization and loading and unloading SBUF, the Serial port BUFfer. If parity is enabled, (i.e., PEN = 1), 7 bits plus even parity are used instead of 8 data bits. The parity calculation is done in hardware for even parity. Modes 2 and 3 are similar to Mode 1, except that the ninth bit needs to be controlled and read. It is also not possible to enable parity in Mode 2. When parity is enabled in Mode 3 the ninth bit becomes the parity bit. If parity is not enabled, (i.e., PEN = 0), the TB8 bit controls the state of the ninth transmitted bit. This bit must be set prior to each transmission. On reception, if PEN = 0, the RB8 bit indicates the state of the ninth received bit. If parity is enabled, (i.e., PEN = 1), the same bit is called RPE (Receive Parity Error), and is used to indicate a parity error.

XTAL1 Frequency = 12.0 MHz		
Baud Rate	**Baud Register Value**	**Percent Error**
19.2K	8009H	+2.40
9600	8013H	+2.40
4800	8026H	−0.16
2400	804DH	−0.16
1200	809BH	−0.16
300	8270H	0.00
XTAL1 Frequency = 11.0 MHz		
19.2K	8008H	+0.54
9600	8011H	+0.54
4800	8023H	+0.54
2400	8047H	+0.54
1200	808EH	−0.16
300	823CH	+0.01
XTAL1 Frequency = 10.0 MHz		
19.2K	8007H	−1.70
9600	800FH	−1.70
4800	8020H	+1.38
2400	8040H	−0.16
1200	8081H	−0.16
300	8208H	+0.03

Figure 2-17. Baud Rate Values for 10, 11, 12 MHz

The software used to communicate between processors is simplified by making use of Modes 2 and 3. In a basic protocol the ninth bit is called the address bit. If it is set high then the information in that byte is either the address of one of the processors on the link, or a command for all the processors. If the bit is a zero, the byte contains information for the processor or processors previously addressed. In standby mode all processors wait in Mode 2 for a byte with the address bit set. When they receive that byte, the software determines if the next message is for them. The processor that is to receive the message switches to Mode 3 and receives the information. Since this information is sent with the ninth bit set to zero, none of the processors set to Mode 2 will be interrupted. By using this scheme the overall CPU time required for the serial port is minimized.

A typical connection diagram for the multi-processor mode is shown in Figure 2-18. This type of communicaton can be used to connect peripherals to a desk top computer, the axis of a multi-axis machine, or any other group of microcontrollers jointly performing a task.

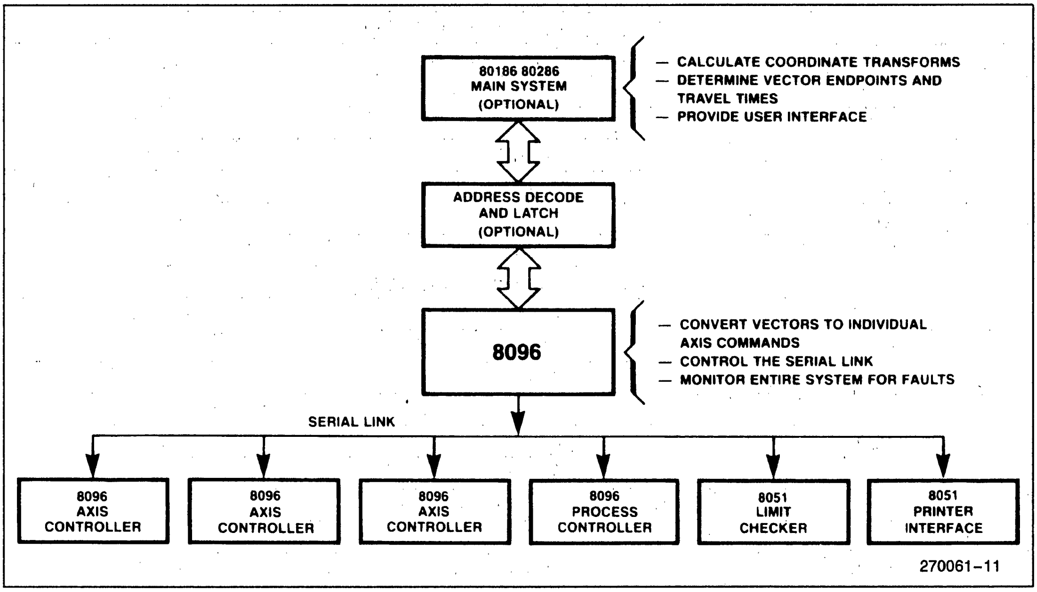


Figure 2-18. Multiprocessor Communication

Mode 0, the synchronous mode, is typically used for interfacing to shift registers for I/O expansion. The software to control this mode involves the REN (Receiver ENable) bit, the clearing of the RI bit, and writing to SBUF. To transmit to a shift register, REN is set to zero and SBUF is loaded with the information. The information will be sent and then the TI flag will be set. There are two ways to cause a reception to begin. The first is by causing a rising edge to occur on the REN bit, the second is by clearing RI with REN = 1. In either case, RI is set again when the received byte is available in SBUF.

2.3.5. A to D CONVERTER

Analog inputs are frequently required in a microcontroller application. The 8097 has a 10-bit A to D converter that can use any one of eight input channels. The conversions are done using the successive approximation method, and require 168 state times (42 microseconds with a 12 MHz clock.)

The results are guaranteed monotonic by design of the converter. This means that if the analog input voltage changes, even slightly, the digital value will either stay the same or change in the same direction as the analog input. When doing process control algorithms, it is frequently the changes in inputs that are required, not the absolute accuracy of the value. For this reason, even if the absolute accuracy of a 10-bit converter is the same as that of an 8-bit converter, the 10-bit monotonic converter is much more useful.

Since most of the analog inputs which are monitored by a microcontroller change very slowly relative to the 42 microsecond conversion time, it is acceptable to use a capacitive filter on each input instead of a sample and hold. The 8097 does not have an internal sample and hold, so it is necessary to ensure that the input signal does not change during the conversion time. The input to the A/D must be between ANGND and VREF. ANGND must be within a few millivolts of VSS and VREF must be within a few tenths of a volt of VCC.

Using the A to D converter on the 8097 can be a very low software overhead task because of the interrupt and HSO unit structure. The A to D can be started by the HSO unit at a preset time. When the conversion is complete it is possible to generate an interrupt. By using these features the A to D can be run under complete interrupt control. The A to D can also be directly

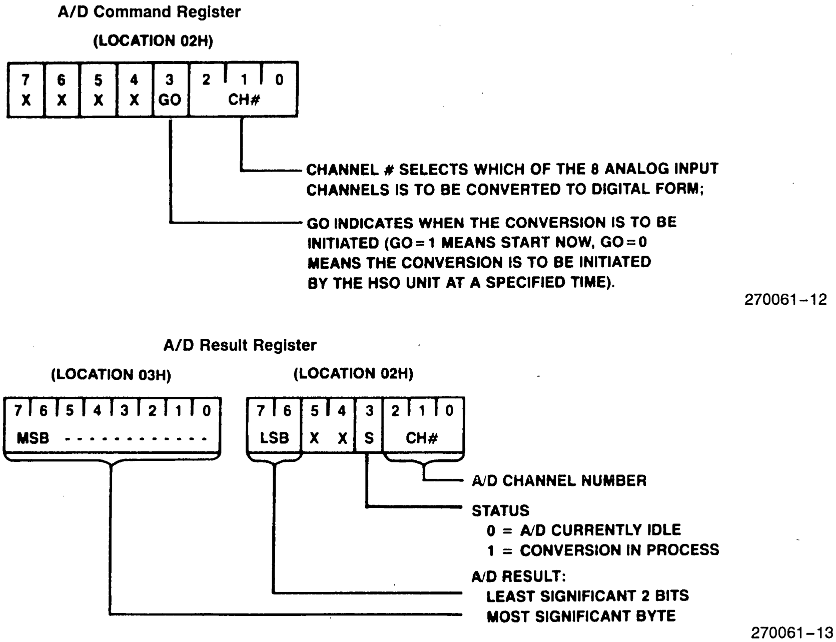

A/D Command Register
(LOCATION 02H)
7 6 5 4 3 2 1 0
X X X X GO CH#
CHANNEL # SELECTS WHICH OF THE 8 ANALOG INPUT CHANNELS IS TO BE CONVERTED TO DIGITAL FORM;
GO INDICATES WHEN THE CONVERSION IS TO BE INITIATED (GO=1 MEANS START NOW, GO=0 MEANS THE CONVERSION IS TO BE INITIATED BY THE HSO UNIT AT A SPECIFIED TIME).
270061-12
A/D Result Register
(LOCATION 03H)
(LOCATION 02H)
7 6 5 4 3 2 1 0
MSB - - - - - - - - - - - -
7 6 5 4 3 2 1 0
LSB X X S CH#
A/D CHANNEL NUMBER
STATUS
0 = A/D CURRENTLY IDLE
1 = CONVERSION IN PROCESS
A/D RESULT:
LEAST SIGNIFICANT 2 BITS
MOST SIGNIFICANT BYTE
270061-13

Figure 2-19. A to D Result/Command Register

controlled by software flags which are located in the AD__RESULT/AD__COMMAND Register, shown in Figure 2-19.

2.3.6. PWM REGISTER

Analog outputs are just as important as analog inputs when connecting to a piece of equipment. True digital to analog converters are difficult to make on a microprocessor because of all of the digital noise and the necessity of providing an on chip, relatively high current, rail to rail driver. They also take up a fair amount of silicon area which can be better used for other features. The A to D converter does use a D to A, but the currents involved are very small.

For many applications an analog output signal can be replaced by a Pulse Width Modulated (PWM) signal. This signal can be easily generated in hardware, and takes up much less silicon area than a true D to A. The signal is a variable duty cycle, fixed frequency waveform that can be integrated to provide an approximation to an analog output. The frequency is fixed at a period of 64 microseconds for a 12 MHz clock speed. Controlling the PWM simply requires writing the desired duty cycle value (an 8-bit value) to the PWM Register. Some typical output waveforms that can be generated are shown in Figure 2-20.

Converting the PWM signal to an analog signal varies in difficulty, depending upon the requirements of the system. Some systems, such as motors or switching power supplies actually require a PWM signal, not a true analog one. For many other cases it is necessary only to amplify the signal so that it switches rail-to-rail, and then filter it. Switching rail-to-rail means that the output of the amplifier will be a reference value when the input is a logical one, and the output will be zero when the input is a logical zero. The filter can be a simple RC network or an active filter. If a large amount of current is needed a buffer is also required. For low output currents, (less than 100 microamps or so), the circuit shown in Figure 2-21 can be used.

The RC network determines how quiet the output is, but the quieter the output, the slower it can change. The design of high accuracy voltage followers and active filters is beyond the scope of this paper, however many books on the subject are available.

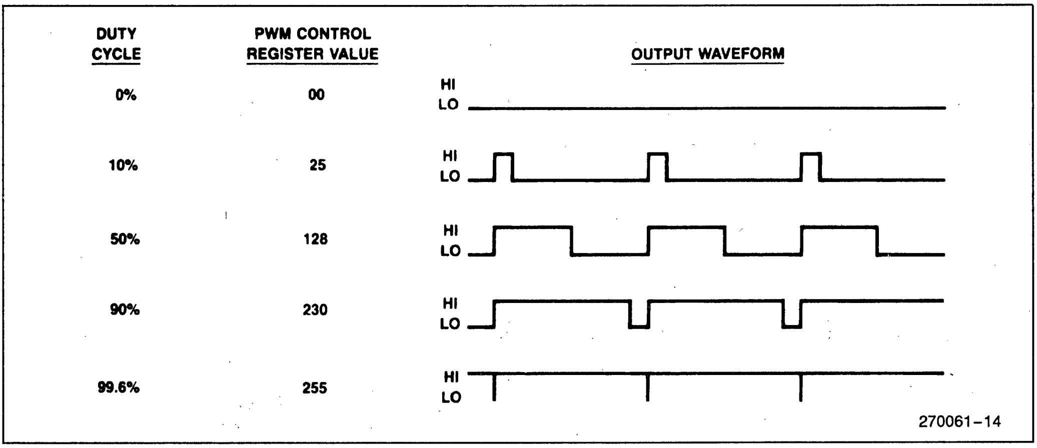
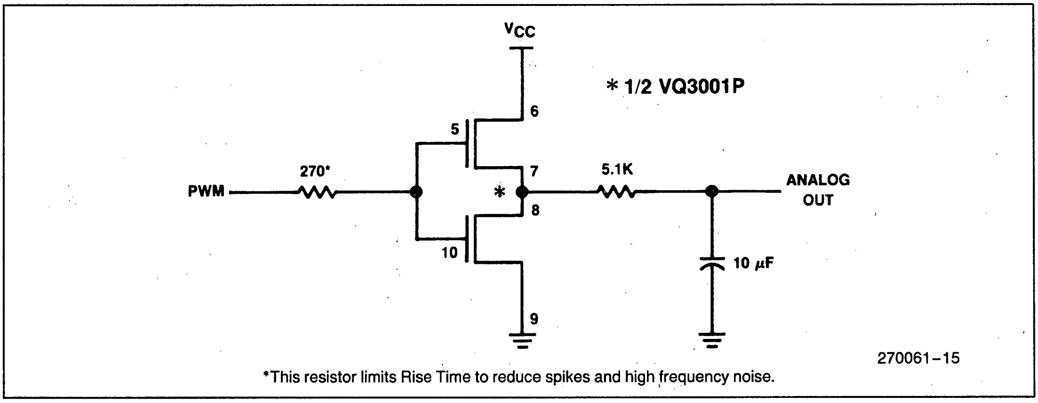


Figure 2-20. PWM Output Waveforms



*This resistor limits Rise Time to reduce spikes and high frequency noise.

Figure 2-21. PWM to Analog Conversion Circuitry

3.0 BASIC SOFTWARE EXAMPLES

The examples in this section show how to use each I/O feature individually. Examples of using more than one feature at a time are described in section 4. All of the examples in this ap-note are set up to be used as listed. If run through ASM96 they will load and run on an SBE-96. In order to insure that the programs work, the stack pointer is initialized at the beginning of each program. If the programs are going to be used as modules of other programs, the stack pointer initialization should only be used at the beginning of the main program.

To avoid repetitive declarations the "include" file "DEMO96.INC", shown in Listing 3-1, is used. ASM-96 will insert this file into the code file whenever the directive "INCLUDE DEMO96.INC" is used. The file contains the definitions for the SFRs and other variables. The include statement has been placed in all of the examples. It should be noted that some of the labels in this file are different from those in the file 8096.INC that is provided in the ASM-96 package.

3.1. Using the 8096's Processing Section

3.1.1. TABLE INTERPOLATION

A good way of increasing speed for many processing tasks is to use table lookup with interpolation. This can eliminate lengthy calculations in many algorithms. Frequently it is used in programs that generate sine waveforms, use exponents in calculations, or require some non-linear function of a given input variable. Table lookup can also be used without interpolation to determine the output state of I/O devices for a given state of a set of input devices. The procedure is also a good example of 8096 code as it uses many of the software features. Two ways of making a lookup table are described, one way uses more calculation time, the second way uses more table space.

```
;*****************************************************************************
;
; DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096
;
;*****************************************************************************
;
ZERO            EQU     00h:WORD        ; R/W
AD_COMMAND      EQU     02H:BYTE        ;   W
AD_RESULT_LO    EQU     02H:BYTE        ; R
AD_RESULT_HI    EQU     03H:BYTE        ; R
HSI_MODE        EQU     03H:BYTE        ;   W
HSO_TIME        EQU     04H:WORD        ;   W
HSI_TIME        EQU     04H:WORD        ; R
HSO_COMMAND     EQU     06H:BYTE        ;   W
HSI_STATUS      EQU     06H:BYTE        ; R
SBUF            EQU     07H:BYTE        ; R/W
INT_MASK        EQU     08H:BYTE        ; R/W
INT_PENDING     EQU     09H:BYTE        ; R/W
SPCON           EQU     11H:BYTE
SPSTAT          EQU     11H:BYTE
WATCHDOG        EQU     0AH:BYTE        ;   W  WATCHDOG TIMER
TIMER1          EQU     0AH:WORD        ; R
TIMER2          EQU     0CH:WORD        ; R
PORT0           EQU     0EH:BYTE        ; R
BAUD_REG        EQU     0EH:BYTE        ;   W
PORT1           EQU     0FH:BYTE        ; R/W
PORT2           EQU     10H:BYTE        ; R/W
IOC0            EQU     15H:BYTE        ;   W
IOS0            EQU     15H:BYTE        ; R
IOC1            EQU     16H:BYTE        ;   W
IOS1            EQU     16H:BYTE        ; R
PWM_CONTROL     EQU     17H:BYTE        ;   W
SP              EQU     18H:WORD        ; R/W STACK POINTER


RSEG at 1CH

        AX:     DSW     1
        DX:     DSW     1
        BX:     DSW     1
        CX:     DSW     1

        AL      EQU     AX      :BYTE
        AH      EQU     (AX+1)  :BYTE
```

270061-16

Listing 3-1. Include File DEMO.96.INC

In both methods the procedure is similar. Values of a function are stored in memory for specific input values. To compute the output function for an input that is not listed, a linear approximation is made based on the nearest inputs and nearest outputs. As an example, consider the table below.

If the input value was one of those listed then there would be no problem. Unfortunately the real world is never so kind. The input number will probably be 259 or something similar. If this is the case linear interpolation would provide a reasonable result. The formula is:

$$\text{Delta Out} = \frac{\text{UpperOutput-Lower Output}}{\text{Upper Input-Lower Input}} * (\text{Actual Input-Lower Input})$$

$$\text{Actual Output} = \text{Lower Output} + \text{Delta Out}$$

For the value of 259 the solution is:

$$\text{Delta Out} = \frac{900\text{-}400}{300\text{-}200} * (259\text{-}200) = \frac{500}{100} * 59 = 5 * 59 = 295$$

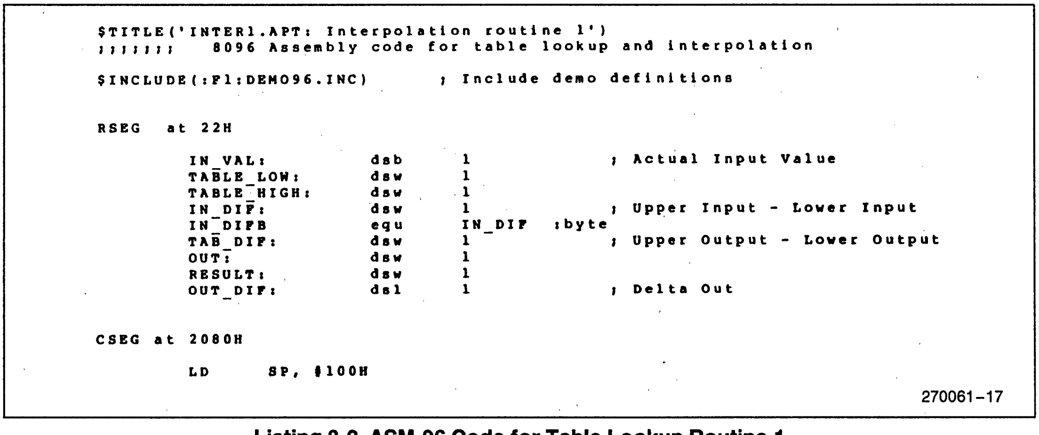
$$\text{Actual Output} = 400 + 295 = 695$$

To make the algorithm easier, (and therefore faster), it is appropriate to limit the range and accuracy of the function to only what is needed. It is also advantageous to make the input step (Upper Input-Lower Input) equal to a power of 2. This allows the substitution of multiple right shifts for a divide operation, thus speeding up throughput. The 8096 allows multiple arithmetic right shifts with a single instruction providing a very fast divide if the divisor is a power of two.

For the purpose of an example, a program with a 12-bit output and an 8-bit input has been written. An input step of 16 (2**4) was selected. To cover the input range 17 words are needed, 255/16 + 1 word to handle values in the last 15 bytes of input range. Although only 12 bits are required for the output, the 16-bit architecture offers no penalty for using 16 instead of 12 bits.

The program for this example, shown in Listing 3-2, uses the definitions and equates from Listing 3-1, only the additional equates and definitions are shown in the code.

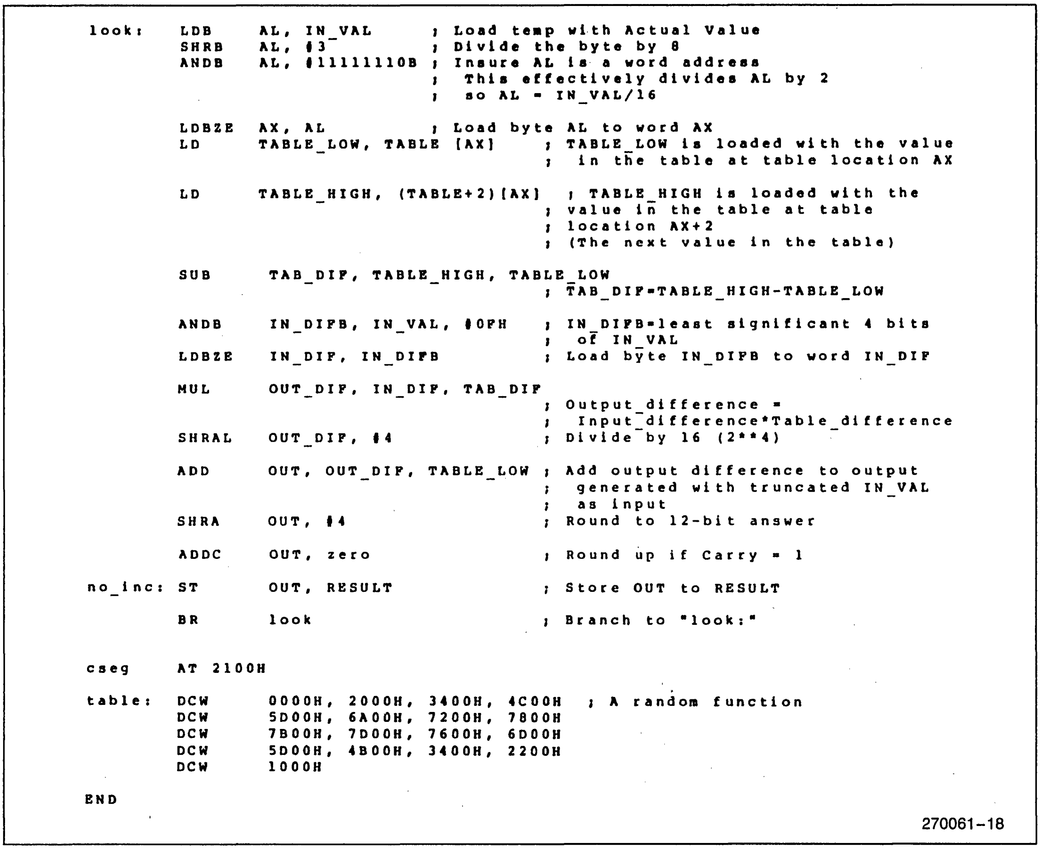
Input Value	Relative Table Address	Table Value
100	0001H	100
200	0002H	400
300	0003H	900
400	0004H	1600



```
$TITLE('INTER1.APT: Interpolation routine 1')
;;;;;;;     8096 Assembly code for table lookup and interpolation

$INCLUDE(:F1:DEMO96.INC)        ; Include demo definitions


RSEG   at  22H

          IN_VAL:         dsb      1              ; Actual Input Value
          TABLE_LOW:      dsw      1
          TABLE_HIGH:     dsw      1
          IN_DIF:         dsw      1              ; Upper Input - Lower Input
          IN_DIFB         equ      IN_DIF   :byte
          TAB_DIF:        dsw      1              ; Upper Output - Lower Output
          OUT:            dsw      1
          RESULT:         dsw      1
          OUT_DIF:        dsl      1              ; Delta Out


CSEG  at  2080H

          LD       SP, #100H
```

270061-17

Listing 3-2. ASM-96 Code for Table Lookup Routine 1

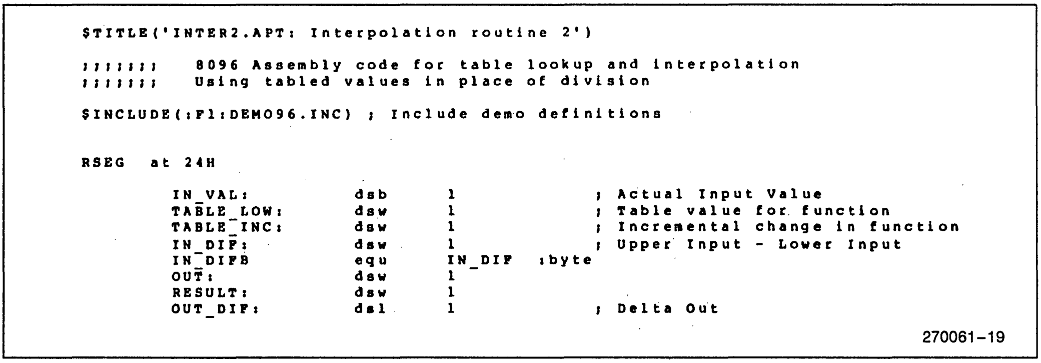


```
look:   LDB     AL, IN_VAL         ; Load temp with Actual Value
        SHRB    AL, #3             ; Divide the byte by 8
        ANDB    AL, #11111110B     ; Insure AL is a word address
                                   ;  This effectively divides AL by 2
                                   ;  so AL = IN_VAL/16

        LDBZE   AX, AL             ; Load byte AL to word AX
        LD      TABLE_LOW, TABLE [AX]      ; TABLE_LOW is loaded with the value
                                           ;  in the table at table location AX

        LD      TABLE_HIGH, (TABLE+2)[AX]   ; TABLE_HIGH is loaded with the
                                           ; value in the table at table
                                           ; location AX+2
                                           ; (The next value in the table)

        SUB      TAB_DIF, TABLE_HIGH, TABLE_LOW
                                           ; TAB_DIF=TABLE_HIGH-TABLE_LOW

        ANDB     IN_DIFB, IN_VAL, #0FH     ; IN_DIFB=least significant 4 bits
                                           ;  of IN_VAL
        LDBZE    IN_DIF, IN_DIFB           ; Load byte IN_DIFB to word IN_DIF

        MUL      OUT_DIF, IN_DIF, TAB_DIF
                                           ; Output_difference =
                                           ;  Input_difference*Table_difference
        SHRAL    OUT_DIF, #4               ; Divide by 16 (2**4)

        ADD      OUT, OUT_DIF, TABLE_LOW   ; Add output difference to output
                                           ;  generated with truncated IN_VAL
                                           ;  as input
        SHRA     OUT, #4                   ; Round to 12-bit answer

        ADDC     OUT, zero                 ; Round up if Carry = 1

no_inc: ST       OUT, RESULT               ; Store OUT to RESULT

        BR       look                      ; Branch to "look:"


cseg    AT 2100H

table:  DCW      0000H, 2000H, 3400H, 4C00H   ; A random function
        DCW      5D00H, 6A00H, 7200H, 7800H
        DCW      7B00H, 7D00H, 7600H, 6D00H
        DCW      5D00H, 4B00H, 3400H, 2200H
        DCW      1000H

END
```

270061-18

Listing 3-2. ASM-96 Code for Table Lookup Routine 1 (Continued)

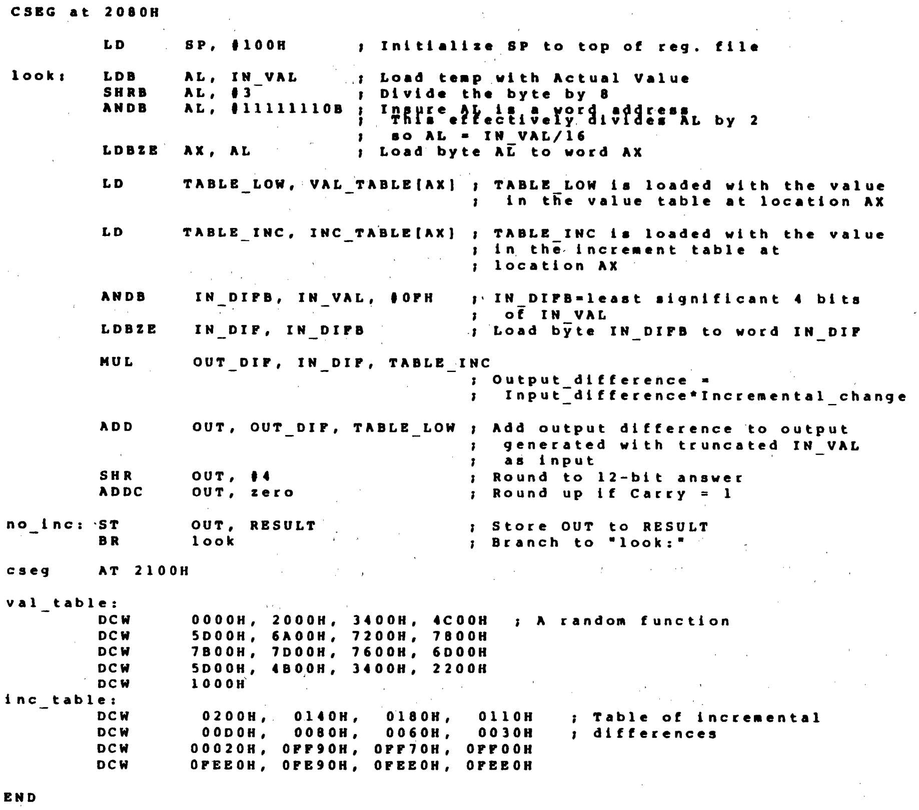
If the function is known at the time of writing the software it is also possible to calculate in advance the change in the output function for a given change in the input. This method can save a divide and a few other instructions at the expense of doubling the size of the lookup table. There are many applications where time is critical and code space is overly abundant. In these cases the code in Listing 3-3 will work to the same specifications as the previous example.



```
$TITLE('INTER2.APT: Interpolation routine 2')

;;;;;;;    8096 Assembly code for table lookup and interpolation
;;;;;;;    Using tabled values in place of division

$INCLUDE(:F1:DEMO96.INC) ; Include demo definitions


RSEG   at 24H

        IN_VAL:          dsb      1              ; Actual Input Value
        TABLE_LOW:       dsw      1              ; Table value for function
        TABLE_INC:       dsw      1              ; Incremental change in function
        IN_DIF:          dsw      1              ; Upper Input - Lower Input
        IN_DIFB          equ      IN_DIF   ;byte
        OUT:             dsw      1
        RESULT:          dsw      1
        OUT_DIF:         dsl      1              ; Delta Out
```

270061-19

Listing 3-3. ASM-96 Code For Table Lookup Routine 2



```
CSEG at 2080H

        LD      SP, #100H          ; Initialize SP to top of reg. file

look:   LDB     AL, IN_VAL         ; Load temp with Actual Value
        SHRB    AL, #3             ; Divide the byte by 8
        ANDB    AL, #11111110B     ; Insure AL is a word address
                                   ;  This effectively divides AL by 2
                                   ;  so AL = IN_VAL/16
        LDBZE   AX, AL             ; Load byte AL to word AX

        LD      TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value
                                         ;  in the value table at location AX

        LD      TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value
                                         ; in the increment table at
                                         ; location AX

        ANDB     IN_DIFB, IN_VAL, #0FH   ; IN_DIFB=least significant 4 bits
                                         ;  of IN_VAL
        LDBZE    IN_DIF, IN_DIFB         ; Load byte IN_DIFB to word IN_DIF

        MUL      OUT_DIF, IN_DIF, TABLE_INC
                                         ; Output_difference =
                                         ;  Input_difference*Incremental_change

        ADD      OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                                         ;  generated with truncated IN_VAL
                                         ;  as input
        SHR      OUT, #4                 ; Round to 12-bit answer
        ADDC     OUT, zero               ; Round up if Carry = 1

no_inc: ST       OUT, RESULT             ; Store OUT to RESULT
        BR       look                    ; Branch to "look:"

cseg    AT 2100H

val_table:
        DCW      0000H, 2000H, 3400H, 4C00H  ; A random function
        DCW      5D00H, 6A00H, 7200H, 7800H
        DCW      7B00H, 7D00H, 7600H, 6D00H
        DCW      5D00H, 4B00H, 3400H, 2200H
        DCW      1000H
inc_table:
        DCW       0200H,  0140H,  0180H,  0110H   ; Table of incremental
        DCW       00D0H,  0080H,  0060H,  0030H   ; differences
        DCW      00020H, 0FF90H, 0FF70H, 0FF00H
        DCW      0FEE0H, 0FE90H, 0FEE0H, 0FEE0H

END
```

270061-20

Listing 3-3. ASM-96 Code for Table Lookup Routine 2 (Continued)

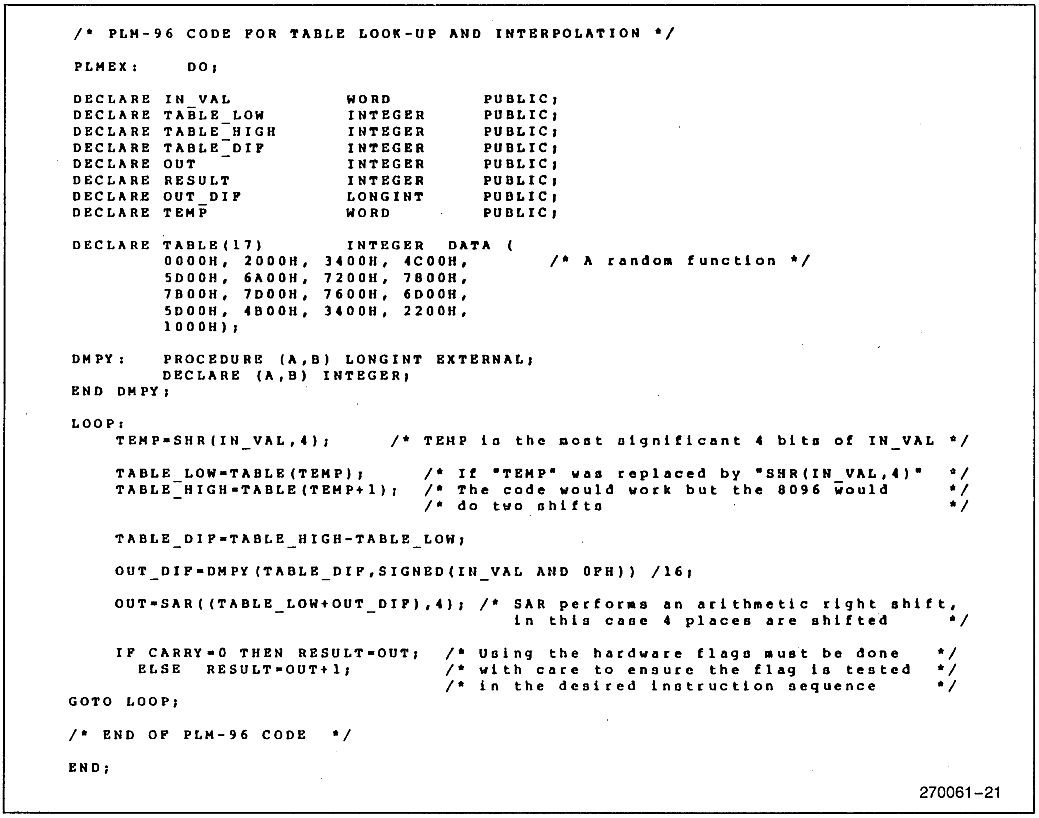
By making use of the second lookup table, one word of RAM was saved and 16 state times. In most cases this time savings would not make much of a difference, but when pushing the processor to the limit, microseconds can make or break a design.

3.1.2. PL/M-96

Intel provides high level language support for most of its micro processors and microcontrollers in the form of PL/M. Specifically, PL/M refers to a family of languages, each similar in syntax, but specialized for the device for which it generates code. The PL/M syntax is similar to PL/1, and is easy to learn. PLM-96 is the version of PL/M used for the 8096. It is very code efficient as it was written specifically for the MCS-96 family. PLM-96 most closely resembles PLM-86, although it has bit and I/O functions similar to PLM-51. One line of PL/M-code can take the place of many lines of assembly code. This is advantageous to the programmer, since code can usually be written at a set number of lines per hour, so the less lines of code that need to be written, the faster the task can be completed.

If the first example of interpolation is considered, the PLM-96 code would be written as shown in Listing 3-4. Note that version 1.0 of PLM-96 does not support 32-bit results of 16 by 16 multiplies, so the ASM-96 procedure "DMPY" is used. Procedure DMPY, shown in Listing 3-5, must be assembled and linked with the compiled PLM-96 program using RL-96, the relocator and linker. The command line to be used is:

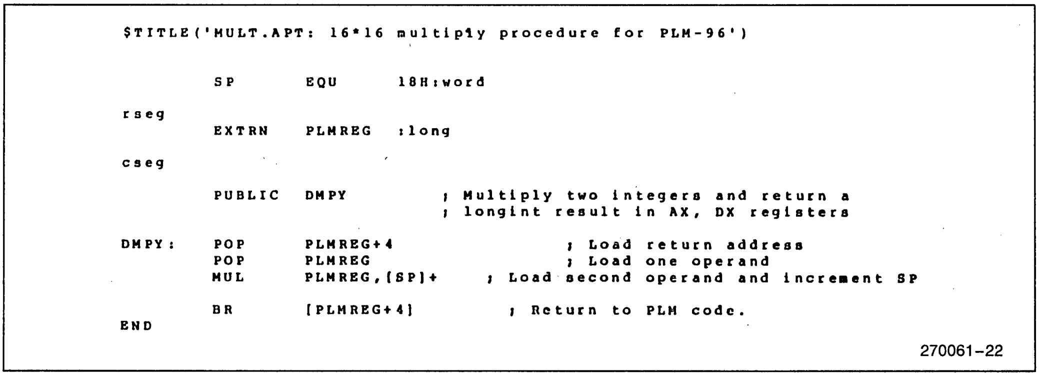
```
RL96 PLMEX1.OBJ, DMPY.OBJ, PLM96.LIB &
to PLMOUT.OBJ ROM (2080H-3FFFH)
```



```
/* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION */

PLMEX:     DO;

DECLARE IN_VAL          WORD         PUBLIC;
DECLARE TABLE_LOW       INTEGER      PUBLIC;
DECLARE TABLE_HIGH      INTEGER      PUBLIC;
DECLARE TABLE_DIF       INTEGER      PUBLIC;
DECLARE OUT             INTEGER      PUBLIC;
DECLARE RESULT          INTEGER      PUBLIC;
DECLARE OUT_DIF         LONGINT      PUBLIC;
DECLARE TEMP            WORD         PUBLIC;

DECLARE TABLE(17)       INTEGER  DATA (
        0000H, 2000H, 3400H, 4C00H,        /* A random function */
        5D00H, 6A00H, 7200H, 7800H,
        7B00H, 7D00H, 7600H, 6D00H,
        5D00H, 4B00H, 3400H, 2200H,
        1000H);

DMPY:   PROCEDURE (A,B) LONGINT EXTERNAL;
        DECLARE (A,B) INTEGER;
END DMPY;

LOOP:
    TEMP=SHR(IN_VAL,4);       /* TEMP is the most significant 4 bits of IN_VAL */

    TABLE_LOW=TABLE(TEMP);      /* If "TEMP" was replaced by "SHR(IN_VAL,4)"  */
    TABLE_HIGH=TABLE(TEMP+1);   /* The code would work but the 8096 would     */
                                /* do two shifts                              */

    TABLE_DIF=TABLE_HIGH-TABLE_LOW;

    OUT_DIF=DMPY(TABLE_DIF,SIGNED(IN_VAL AND 0FH)) /16;

    OUT=SAR((TABLE_LOW+OUT_DIF),4); /* SAR performs an arithmetic right shift,
                                       in this case 4 places are shifted      */

    IF CARRY=0 THEN RESULT=OUT;   /* Using the hardware flags must be done   */
      ELSE  RESULT=OUT+1;         /* with care to ensure the flag is tested  */
                                  /* in the desired instruction sequence     */
GOTO LOOP;

/* END OF PLM-96 CODE  */

END;
```

270061-21

Listing 3-4. PLM-96 Code For Table Lookup Routine 1



```
$TITLE('MULT.APT: 16*16 multiply procedure for PLM-96')

        SP      EQU     18H:word

rseg
        EXTRN   PLMREG  :long

cseg

        PUBLIC  DMPY            ; Multiply two integers and return a
                                ; longint result in AX, DX registers

DMPY:   POP     PLMREG+4                  ; Load return address
        POP     PLMREG                    ; Load one operand
        MUL     PLMREG,[SP]+     ; Load second operand and increment SP

        BR      [PLMREG+4]         ; Return to PLM code.
END
```

270061-22

Listing 3-5. 32-Bit Result Multiply Procedure For PLM-96

Using PLM, code requires less lines, is much faster to write, and easier to maintain, but may take slightly longer to run. For this example, the assembly code generated by the PLM-96 compiler takes 56.75 microseconds to run instead of 30.75 microseconds. If PLM-96 performed the 32-bit result multiply instead of using the ASM-96 routine the PLM code would take 41.5 microseconds to run. The actual code listings are shown in Appendix A.

3.2. Using the I/O Section

3.2.1. USING THE HSI UNIT

One of the most frequent uses of the HSI is to measure the time between events. This can be used for frequency determination in lab instruments, or speed/acceleration information when connected to pulse type encoders. The code in Listing 3-6 can be used to determine the high and low times of the signals on two lines. This code can be easily expanded to 4 lines and can also be modified to work as an interrupt routine.

Frequently it is also desired to keep track of the number of events which have occurred, as well as how often they are occurring. By using a software counter this feature can be added to the above code. This code depends on the software responding to the change in line state before the line changes again. If this cannot be guaranteed then it may be necessary to use 2 HSI lines for each incoming line. In this case one HSI line would look for falling edges while the other looks for rising edges. The code in Listing 3-7 includes both the counter feature and the edge detect feature.

The uses for this type of routine are almost endless. In instrumentation it can be used to determine frequency on input lines, or perhaps baud rate for a self adjusting serial port. Section 4.2 contains an example of making a software serial port using the HSI unit. Interfacing to some form of mechanically generated position information is a very frequent use of the HSI. The applications in this category include motor control, precise positioning (print heads, disk drives, etc.), engine control and

```
$TITLE('PULSE.APT: Measuring pulses using the HSI unit')

$INCLUDE(DEMO96.INC)

rseg     at  28H

         HIGH_TIME:      dsw      1
         LOW_TIME:       dsw      1
         PERIOD:         dsw      1
         HI_EDGE:        dsw      1
         LO_EDGE:        dsw      1


cseg     at  2080H


         LD      SP, #100H
         LDB     IOC0, #00000001B          ; Enable HSI 0
         LDB     HSI_MODE, #00001111B      ; HSI 0 look for either edge

wait:    ADD     PERIOD, HIGH_TIME, LOW_TIME
         JBS     IOS1, 6, contin           ; If FIFO is full
         JBC     IOS1, 7, wait   ; Wait while no pulse is entered

contin:  LDB     AL, HSI_STATUS            ; Load status; Note that reading
                                           ;   HSI_TIME clears HSI_STATUS

         LD      BX, HSI_TIME              ; Load the HSI_TIME

         JBS     AL, 1, hsi_hi             ; Jump if HSI.0 is high

hsi_lo:  ST      BX, LO_EDGE
         SUB     HIGH_TIME, LO_EDGE, HI_EDGE
         BR      wait


hsi_hi:  ST      BX, HI_EDGE
         SUB     LOW_TIME, HI_EDGE, LO_EDGE
         BR      wait

         END
```

270061-23

Listing 3-6. Measuring Pulses Using The HSI Unit

transmission control. The HSI unit is used extensively in the example in section 4.3.

3.2.2. USING THE HSO UNIT

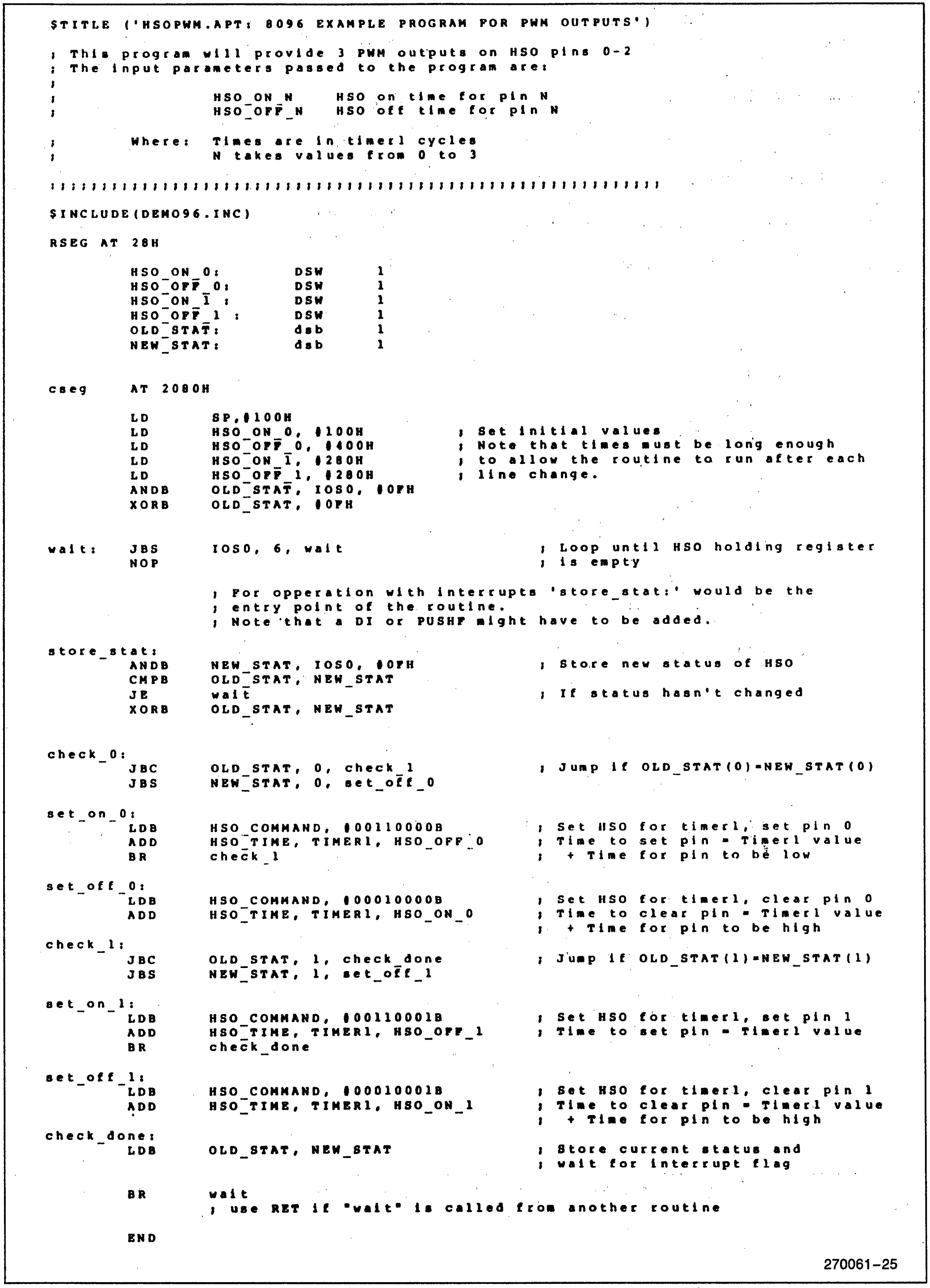
Although the HSO has many uses, the best example is that of a multiple PWM output. This program, shown in Listing 3-8, is simple enough to be easily understood, yet it shows how to use the HSO for a task which can be complex. In order for this program to operate, another program needs to set up the on and off time variables for each line. The program also requires that a HSO line not change so quickly that it changes twice between consecutive reads of I/O Status Register 0, (IOS0).

A very eye catching example can be made by having the program output waveforms that vary over time. The driver routine in Listing 3-10 can be linked to the above program to provide this function. Linking is accomplished using RL96, the relocatable linker for the 8096. Information for using RL96 can be found in the "MCS-96 Utilities Users Guide", listed in the bibliography. In order for the program to link, the register dec-

```
$TITLE ('ENHSI.APT: ENHANCED HSI PULSE ROUTINE')

$INCLUDE(DEMO96.INC)

RSEG AT 28H

        TIME:           DSW 1
        LAST_RISE:      DSW 1
        LAST_FALL:      DSW 1
        HSI_S0:         DSB 1
        IOS1_BAK:       DSB 1
        PERIOD:         DSW 1
        LOW_TIME:       DSW 1
        HIGH_TIME:      DSW 1
        COUNT:          DSW 1

cseg    at      2080H

init:   LD      SP,#100H

        LDB     IOC1,#00100101B ; Disable HSO.4,HSO.5, HSI_INT=first,
                                ; Enable PWM,TXD,TIMER1_OVRFLOW_INT

        LDB     HSI_MODE,#10011001B     ; set hsi.1 -; hsi.0 +
        LDB     IOC0,#00000111B         ; Enable  hsi 0,1
                                        ; T2 CLOCK=T2CLK, T2RST=T2RST
                                        ; Clear timer2


wait:   ANDB    IOS1_BAK,#01111111B     ; Clear IOS1_BAK.7
        ORB     IOS1_BAK,IOS1           ; Store into temp to avoid clearing
                                        ; other flags which may be needed
        JBC     IOS1_BAK,7,wait         ; If hsi is not triggered then
                                        ; jump to wait

        ANDB    HSI_S0,HSI_STATUS,#01010101B
        LD      TIME, HSI_TIME

        JBS     HSI_S0,0,a_rise
        JBS     HSI_S0,2,a_fall
        BR      no_cnt

a_rise: SUB     LOW_TIME, TIME, LAST_FALL
        SUB     PERIOD, TIME,LAST_RISE
        LD      LAST_RISE, TIME
        BR      increment

a_fall: SUB     HIGH_TIME, TIME, LAST_RISE
        SUB     PERIOD, TIME,LAST_FALL
        LD      LAST_FALL, TIME

increment:
        INC     COUNT

no_cnt: BR      wait

        END
```

270061-24

Listing 3-7. Enhanced HSI Pulse Measurement Routine



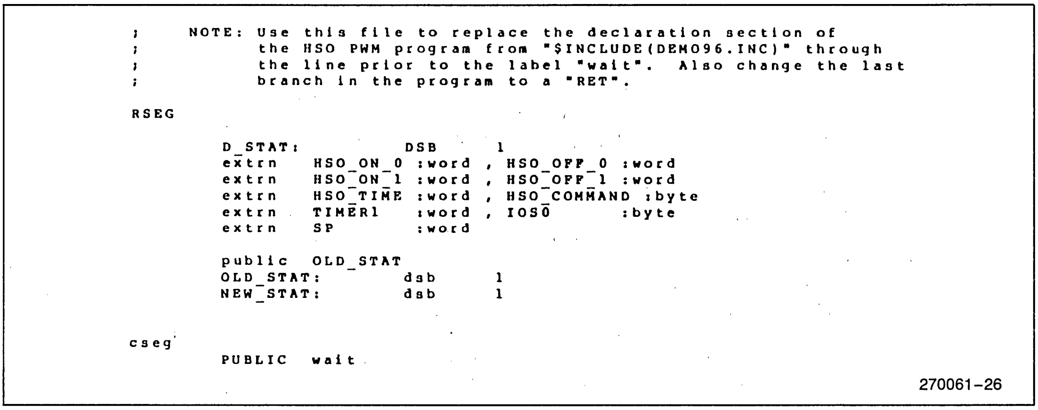
```
$TITLE ('HSOPWM.APT: 8096 EXAMPLE PROGRAM FOR PWM OUTPUTS')

; This program will provide 3 PWM outputs on HSO pins 0-2
; The input parameters passed to the program are:
;
;                  HSO_ON_N     HSO on time for pin N
;                  HSO_OFF_N    HSO off time for pin N

;         Where:   Times are in timer1 cycles
;                  N takes values from 0 to 3

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

$INCLUDE(DEMO96.INC)

RSEG AT 28H

          HSO_ON_0:          DSW       1
          HSO_OFF_0:         DSW       1
          HSO_ON_1 :         DSW       1
          HSO_OFF_1 :        DSW       1
          OLD_STAT:          dsb       1
          NEW_STAT:          dsb       1


cseg      AT 2080H

          LD       SP,#100H
          LD       HSO_ON_0, #100H          ; Set initial values
          LD       HSO_OFF_0, #400H         ; Note that times must be long enough
          LD       HSO_ON_1, #280H          ; to allow the routine to run after each
          LD       HSO_OFF_1, #280H         ; line change.
          ANDB     OLD_STAT, IOS0, #0FH
          XORB     OLD_STAT, #0FH


wait:     JBS      IOS0, 6, wait                      ; Loop until HSO holding register
          NOP                                         ; is empty

                   ; For opperation with interrupts 'store_stat:' would be the
                   ; entry point of the routine.
                   ; Note that a DI or PUSHF might have to be added.

store_stat:
          ANDB     NEW_STAT, IOS0, #0FH               ; Store new status of HSO
          CMPB     OLD_STAT, NEW_STAT
          JE       wait                               ; If status hasn't changed
          XORB     OLD_STAT, NEW_STAT


check_0:
          JBC      OLD_STAT, 0, check_1               ; Jump if OLD_STAT(0)=NEW_STAT(0)
          JBS      NEW_STAT, 0, set_off_0

set_on_0:
          LDB      HSO_COMMAND, #00110000B            ; Set HSO for timer1, set pin 0
          ADD      HSO_TIME, TIMER1, HSO_OFF_0        ; Time to set pin = Timer1 value
          BR       check_1                            ;  + Time for pin to be low

set_off_0:
          LDB      HSO_COMMAND, #00010000B            ; Set HSO for timer1, clear pin 0
          ADD      HSO_TIME, TIMER1, HSO_ON_0         ; Time to clear pin = Timer1 value
                                                      ;  + Time for pin to be high
check_1:
          JBC      OLD_STAT, 1, check_done            ; Jump if OLD_STAT(1)=NEW_STAT(1)
          JBS      NEW_STAT, 1, set_off_1

set_on_1:
          LDB      HSO_COMMAND, #00110001B            ; Set HSO for timer1, set pin 1
          ADD      HSO_TIME, TIMER1, HSO_OFF_1        ; Time to set pin = Timer1 value
          BR       check_done

set_off_1:
          LDB      HSO_COMMAND, #00010001B            ; Set HSO for timer1, clear pin 1
          ADD      HSO_TIME, TIMER1, HSO_ON_1         ; Time to clear pin = Timer1 value
                                                      ;  + Time for pin to be high
check_done:
          LDB      OLD_STAT, NEW_STAT                 ; Store current status and
                                                      ; wait for interrupt flag

          BR       wait
                   ; use RET if "wait" is called from another routine

          END
```

270061-25

Listing 3-8. Generating a PWM with the HSO

laration section (i.e., the section between “RSEG” and “CSEG”) in Listing 3-8 must be changed to that in Listing 3-9.

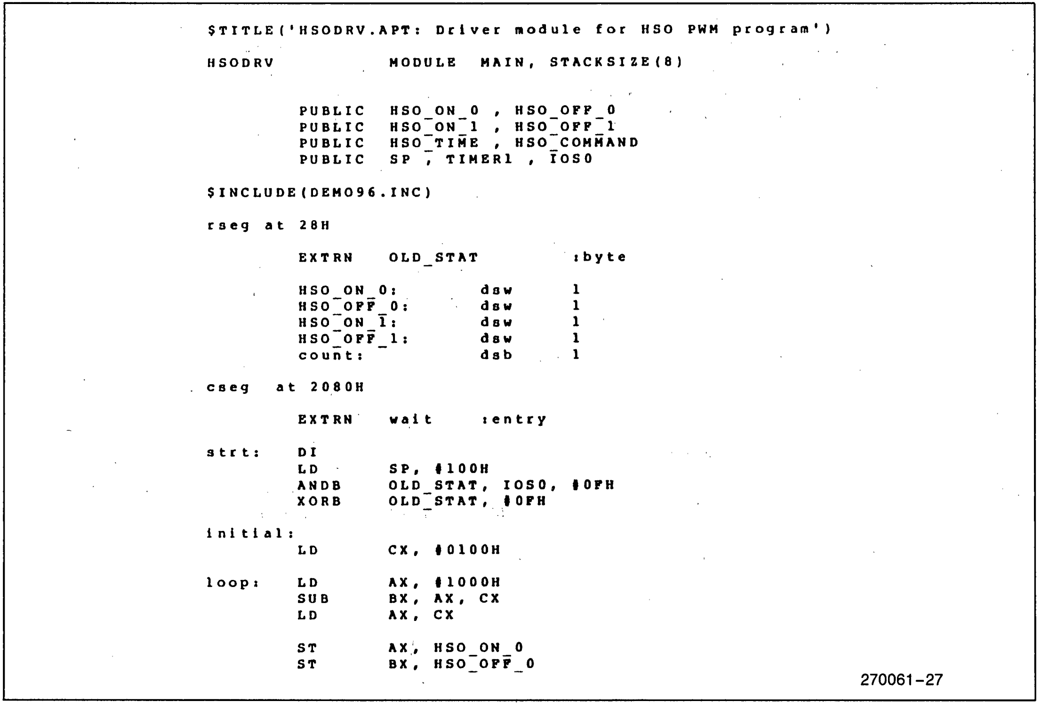
The driver routine simply changes the duty cycle of the waveform and sets the second HSO output to a frequency twice that of the first one. A slightly different driver routine could easily be the basis for a switching power supply or a variable frequency/variable voltage motor driver. The listing of the driver routine is shown in Listing 3-10.



```
;       NOTE: Use this file to replace the declaration section of
;             the HSO PWM program from "$INCLUDE(DEMO96.INC)" through
;             the line prior to the label "wait".  Also change the last
;             branch in the program to a "RET".

RSEG

        D_STAT:          DSB     1
        extrn   HSO_ON_0  :word , HSO_OFF_0 :word
        extrn   HSO_ON_1  :word , HSO_OFF_1 :word
        extrn   HSO_TIME  :word , HSO_COMMAND :byte
        extrn   TIMER1    :word , IOS0        :byte
        extrn   SP        :word

        public  OLD_STAT
        OLD_STAT:        dsb     1
        NEW_STAT:        dsb     1

cseg
        PUBLIC  wait
```

270061-26

Listing 3-9. Changes to Declarations for HSO Routine



```
$TITLE('HSODRV.APT: Driver module for HSO PWM program')

HSODRV          MODULE  MAIN, STACKSIZE(8)


        PUBLIC  HSO_ON_0 , HSO_OFF_0
        PUBLIC  HSO_ON_1 , HSO_OFF_1
        PUBLIC  HSO_TIME , HSO_COMMAND
        PUBLIC  SP , TIMER1 , IOS0

$INCLUDE(DEMO96.INC)

rseg  at  28H

        EXTRN   OLD_STAT            :byte

        HSO_ON_0:       dsw     1
        HSO_OFF_0:      dsw     1
        HSO_ON_1:       dsw     1
        HSO_OFF_1:      dsw     1
        count:          dsb     1

cseg    at 2080H

        EXTRN   wait    :entry

strt:   DI
        LD      SP, #100H
        ANDB    OLD_STAT, IOS0, #0FH
        XORB    OLD_STAT, #0FH

initial:
        LD      CX, #0100H

loop:   LD      AX, #1000H
        SUB     BX, AX, CX
        LD      AX, CX

        ST      AX, HSO_ON_0
        ST      BX, HSO_OFF_0
```

270061-27

Listing 3-10. Driver Module for HSO PWM Program

```
        SHR     AX,#1
        SHR     BX,#1
        ST      AX, HSO_ON_1
        ST      BX, HSO_OFF_1

        CALL    wait

        INC     CX
        CMP     CX, #00F00H
        BNE     loop

        BR      initial

        END
```

270061-28

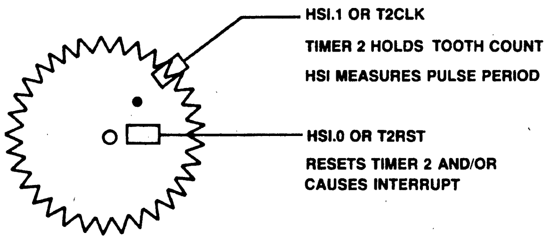
Listing 3-10. Driver Module for HSO PWM Program (Continued)

Since the 8096 needs to keep track of events which often repeat at set intervals it is convenient to be able to have Timer 2 act as a programmable modulo counter. There are several ways of doing this. The first is to program the HSO to reset Timer 2 when Timer 2 equals a set value. A software timer set to interrupt at Timer 2 equals zero could be used to reload the CAM. This software method takes up two locations in the CAM and does not synchronize Timer 2 to the external world.

To synchronize Timer 2 externally the T2 RST (Timer 2 ReSeT) pin can be used. In this way Timer 2 will get reset on each rising edge of T2 RST. If it is desired to have an interrupt generated and time recorded when Timer 2 gets reset, the signal for its reset can be taken from HSI.0 instead of T2RST. The HSI.0 pin has its own interrupt vector which functions independently of the HSI unit.

Another option available is to use the HSI.1 pin to clock Timer 2. By using this approach it is possible to use the HSI to measure the period of events on the input to Timer 2. If both of the HSI pins are used instead of the T2RST and T2CLK pins the HSIO unit can keep track of speed and position of the rotating device with very little software overhead. This type of setup is ideal for a system like the one shown in Figure 3-1, and similar to the one used in section 4.3.

In this system a sequence of events is required based on the position of the gear which represents any piece of rotating machinery. Timer 2 holds the count of the number of tooth edges passed since the index mark. By using HSI.1 as the input to Timer 2, instead of T2 CLK, it is possible to determine tooth count and time information through the HSI. From this information instantaneous velocity and acceleration can be calculated. Having the tooth edge count in Timer 2 means



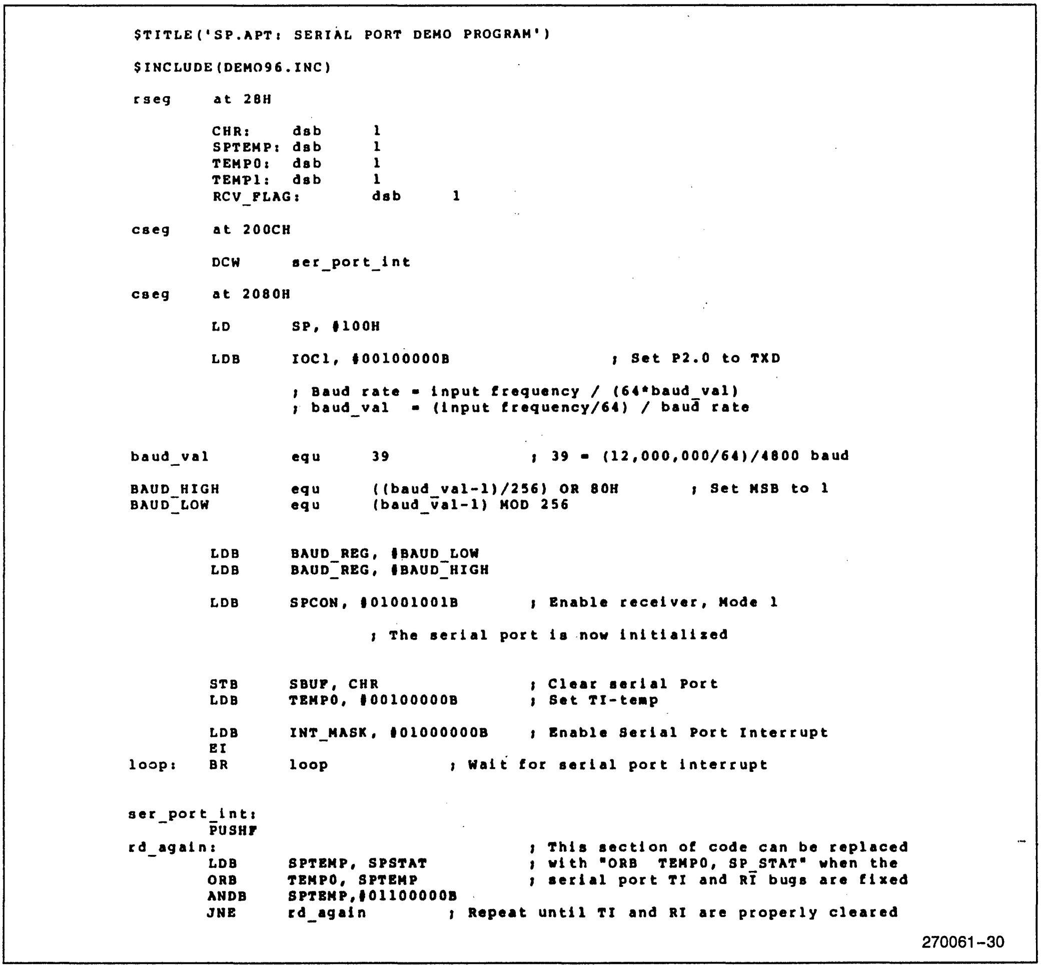
270061-29

Figure 3-1. Using the HSIO to Monitor Rotating Machinery

that the HSO unit can be used to initiate the desired tasks at the appropriate tooth count. The interrupt routine initiated by HSI.0 can be used to perform any software task required every revolution. In this system, the overhead which would normally require extensive software has been done with the hardware on the 8096, thus making more software time available for control programs.

3.2.3. USING THE SERIAL PORT IN MODE 1

Mode 1 of the serial port supports the basic asynchronous 8-bit protocol and is used to interface to most CRTs and printers. The example in Listing 3-11 shows a simple routine which receives a character and then transmits the same character. The code is set up so that minor modifications could make it run on an interrupt basis. Note that it is necessary to set up some flags as initial conditions to get the routine to run properly. If it was desired to send 7 bits of data plus parity instead of 8 bits of data the PEN bit would be set to a one. Interprocessor communication, as described in section 2.3.4, can be set up by simply adding code to change RB8 and the port mode to the listing below. The hardware shown in Figure 3-2 can be used to convert the logic level output of the 8096 to ±12 or 15 volt levels to connect to a CRT. This circuit has been found to work with most RS-232 devices, although it does not conform to strict RS-232 specifications. If true RS-232 conformance is required then any standard RS-232 driver can be used.



```
$TITLE('SP.APT: SERIAL PORT DEMO PROGRAM')

$INCLUDE(DEMO96.INC)

rseg    at 28H

        CHR:     dsb     1
        SPTEMP:  dsb     1
        TEMPO:   dsb     1
        TEMP1:   dsb     1
        RCV_FLAG:        dsb     1

cseg    at 200CH

        DCW     ser_port_int

cseg    at 2080H

        LD      SP, #100H

        LDB     IOC1, #00100000B                ; Set P2.0 to TXD

                ; Baud rate = input frequency / (64*baud_val)
                ; baud_val  = (input frequency/64) / baud rate


baud_val        equ     39              ; 39 = (12,000,000/64)/4800 baud

BAUD_HIGH       equ     ((baud_val-1)/256) OR 80H       ; Set MSB to 1
BAUD_LOW        equ     (baud_val-1) MOD 256


        LDB     BAUD_REG, #BAUD_LOW
        LDB     BAUD_REG, #BAUD_HIGH

        LDB     SPCON, #01001001B       ; Enable receiver, Mode 1

                        ; The serial port is now initialized


        STB     SBUF, CHR               ; Clear serial Port
        LDB     TEMPO, #00100000B       ; Set TI-temp

        LDB     INT_MASK, #01000000B    ; Enable Serial Port Interrupt
        EI
loop:   BR      loop            ; Wait for serial port interrupt


ser_port_int:
        PUSHF
rd_again:                               ; This section of code can be replaced
        LDB     SPTEMP, SPSTAT          ; with "ORB  TEMPO, SP_STAT" when the
        ORB     TEMPO, SPTEMP           ; serial port TI and RI bugs are fixed
        ANDB    SPTEMP,#01100000B
        JNE     rd_again        ; Repeat until TI and RI are properly cleared
```

270061-30

Listing 3-11. Using the Serial Port in Mode 1

```
get_byte:
        JBC     TEMP0, 6, put_byte       ; If RI-temp is not set
        STB     SBUF, CHR                ; Store byte
        ANDB    TEMP0, #10111111B        ; CLR RI-temp
        LDB     RCV_FLAG, #0FFH          ; Set bit-received flag

put_byte:
        JBC     RCV_FLAG, 0, continue    ; If receive flag is cleared
        JBC     TEMP0, 5, continue       ; If TI was not set
        LDB     SBUF, CHR                ; Send byte
        ANDB    TEMP0, #11011111B        ; CLR TI-temp

        ANDB    CHR, #01111111B          ; This section of code appends
        CMPB    CHR,#0DH                 ; an LF after a CR is sent
        JNE     clr_rcv
        LDB     CHR, #0AH
        BR      continue

clr_rcv:
        CLRB    RCV_FLAG                 ; Clear bit-received flag

continue:
        POPF
        RET

        END
```

270061-31

Listing 3-11. Using the Serial Port in Mode 1 (Continued)

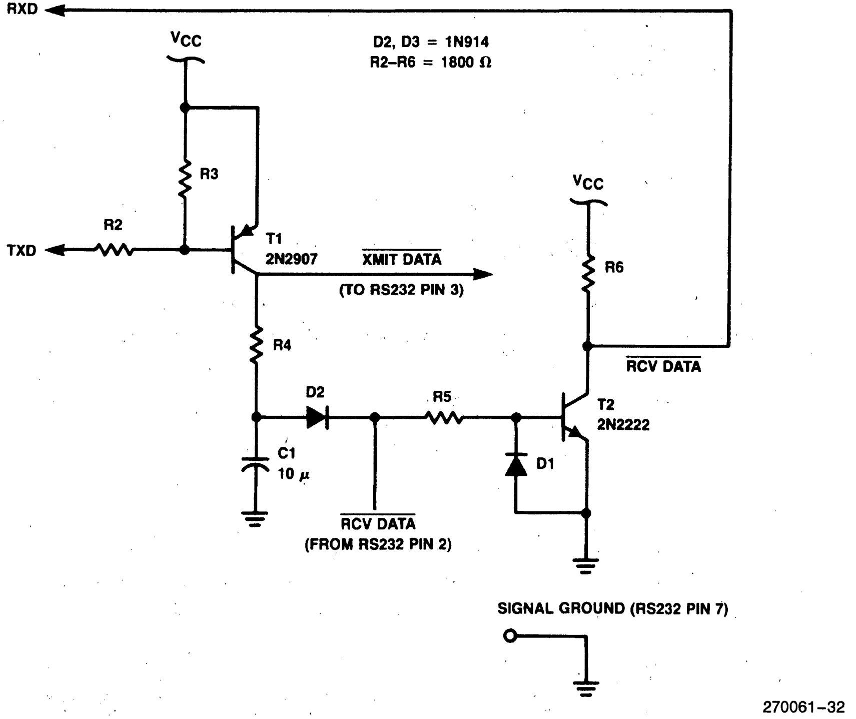


Figure 3-2. Serial Port Level Conversion

3.2.4. USING THE A TO D

The code in Listing 3-12 makes use of the software flags to implement a non-interrupt driven routine which scans A to D channels 0 through 3 and stores them as words in RAM. An interrupt driven routine is shown in section 4.1. When using the A to D it is important to always read the value using the byte read commands, and to give the converter 8 state times to start converting before reading the status bit.

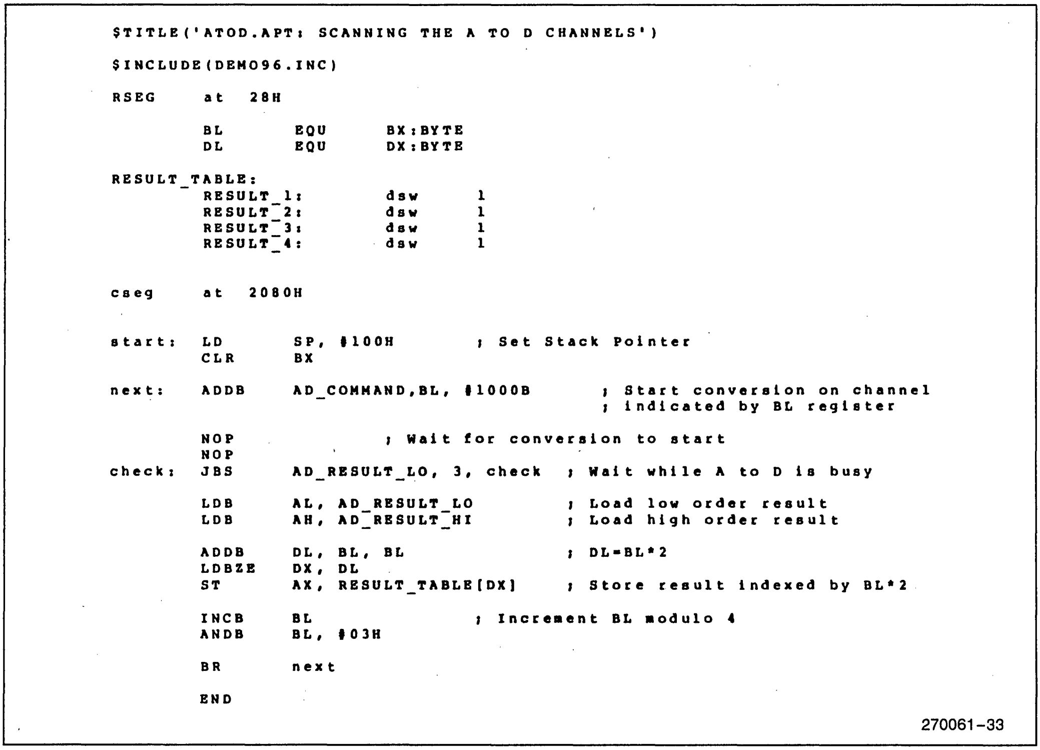
Since there is no sample and hold on the A to D converter it may be desirable to use an RC filter on each input. A 100Ω resistor in series with a 0.22 uf capacitor to ground has been used successfully in the lab. This circuit gives a time constant of around 22 microseconds which should be long enough to get rid of most noise, without overly slowing the A to D response time.

4.0 ADVANCED SOFTWARE EXAMPLES

Using the 8096 for applications which consist only of the brief examples in the previous section does not really make use of its full capabilities. The following examples use some of the code blocks from the previous section to show how several I/O features can be used together to accomplish a practical task. Three examples will be shown. The first is simply a combination of several of the section 3 examples run under an interrupt system. Next, a software serial port using the HSIO unit is described. The concluding example is one of interfacing the HSI unit to an optical encoder to control a motor.

4.1. Simultaneous I/O Routines under Interrupt Control

A four channel analog to PWM converter can easily be made using the 8096. In the example in Listing 4 analog channels are read and 3 PWM waveforms are generated on the HSO lines and one on the PWM pin. Each analog channel is used to set the duty cycle of its associated output pin. The interrupt system keeps the whole program humming, providing time for a background task which is simply a 32 bit software counter. To show which routines are executing and in which



```
$TITLE('ATOD.APT: SCANNING THE A TO D CHANNELS')

$INCLUDE(DEMO96.INC)

RSEG    at   28H

        BL        EQU        BX:BYTE
        DL        EQU        DX:BYTE

RESULT_TABLE:
        RESULT_1:          dsw     1
        RESULT_2:          dsw     1
        RESULT_3:          dsw     1
        RESULT_4:          dsw     1


cseg    at   2080H

start:  LD        SP, #100H          ; Set Stack Pointer
        CLR       BX

next:   ADDB      AD_COMMAND,BL, #1000B       ; Start conversion on channel
                                              ; indicated by BL register

        NOP                ; Wait for conversion to start
        NOP
check:  JBS       AD_RESULT_LO, 3, check    ; Wait while A to D is busy

        LDB       AL, AD_RESULT_LO          ; Load low order result
        LDB       AH, AD_RESULT_HI          ; Load high order result

        ADDB      DL, BL, BL                ; DL=BL*2
        LDBZE     DX, DL
        ST        AX, RESULT_TABLE[DX]      ; Store result indexed by BL*2

        INCB      BL                 ; Increment BL modulo 4
        ANDB      BL, #03H

        BR        next

        END
```

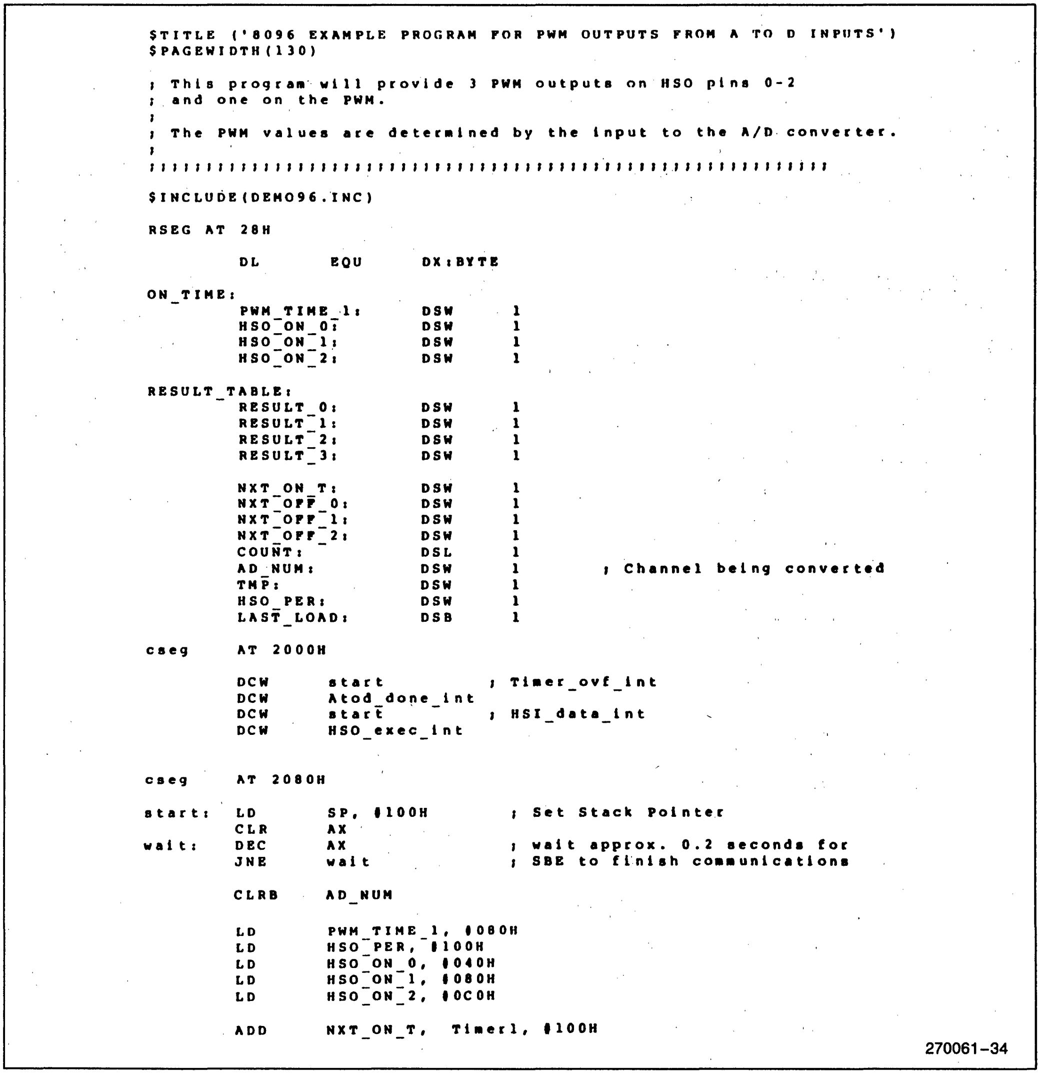
270061-33

Listing 3-12. Scanning the A to D Channels

order, Port 1 output pins are used to indicate the current status of each task. The actual code listing is included in Appendix B.

The initialization section, shown in Listing 4-1a, clears a few variables and then loads the first set of on and off times to the HSO unit. Note that 8 state times must be waited between consecutive loads of the HSO. If this is not done it is possible to overwrite the contents of the CAM holding register. An A/D interrupt is forced by setting the bit in the Interrupt Pending register. This causes the first A/D interrupt to occur just after the Interrupt Mask register is set and interrupts are enabled.

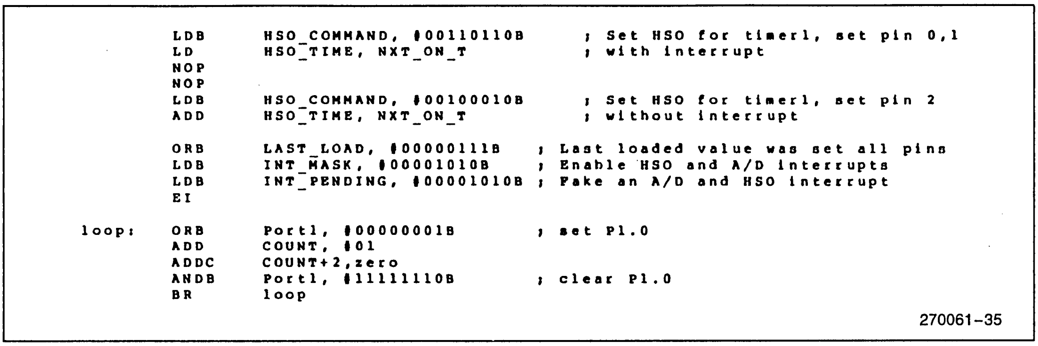
Listing 4-1. Using Multiple I/O Devices



```
$TITLE ('8096 EXAMPLE PROGRAM FOR PWM OUTPUTS FROM A TO D INPUTS')
$PAGEWIDTH(130)

; This program will provide 3 PWM outputs on HSO pins 0-2
; and one on the PWM.
;
; The PWM values are determined by the input to the A/D converter.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

$INCLUDE(DEMO96.INC)

RSEG AT 28H

        DL          EQU         DX:BYTE

ON_TIME:
        PWM_TIME_1:             DSW     1
        HSO_ON_0:               DSW     1
        HSO_ON_1:               DSW     1
        HSO_ON_2:               DSW     1

RESULT_TABLE:
        RESULT_0:               DSW     1
        RESULT_1:               DSW     1
        RESULT_2:               DSW     1
        RESULT_3:               DSW     1

        NXT_ON_T:               DSW     1
        NXT_OFF_0:              DSW     1
        NXT_OFF_1:              DSW     1
        NXT_OFF_2:              DSW     1
        COUNT:                  DSL     1
        AD_NUM:                 DSW     1          ; Channel being converted
        TMP:                    DSW     1
        HSO_PER:                DSW     1
        LAST_LOAD:              DSB     1

cseg    AT 2000H

        DCW     start           ; Timer_ovf_int
        DCW     Atod_done_int
        DCW     start           ; HSI_data_int
        DCW     HSO_exec_int


cseg    AT 2080H

start:  LD      SP, #100H       ; Set Stack Pointer
        CLR     AX
wait:   DEC     AX              ; wait approx. 0.2 seconds for
        JNE     wait            ; SBE to finish communications

        CLRB    AD_NUM

        LD      PWM_TIME_1, #080H
        LD      HSO_PER, #100H
        LD      HSO_ON_0, #040H
        LD      HSO_ON_1, #080H
        LD      HSO_ON_2, #0C0H

        ADD     NXT_ON_T,  Timer1, #100H
```

270061-34

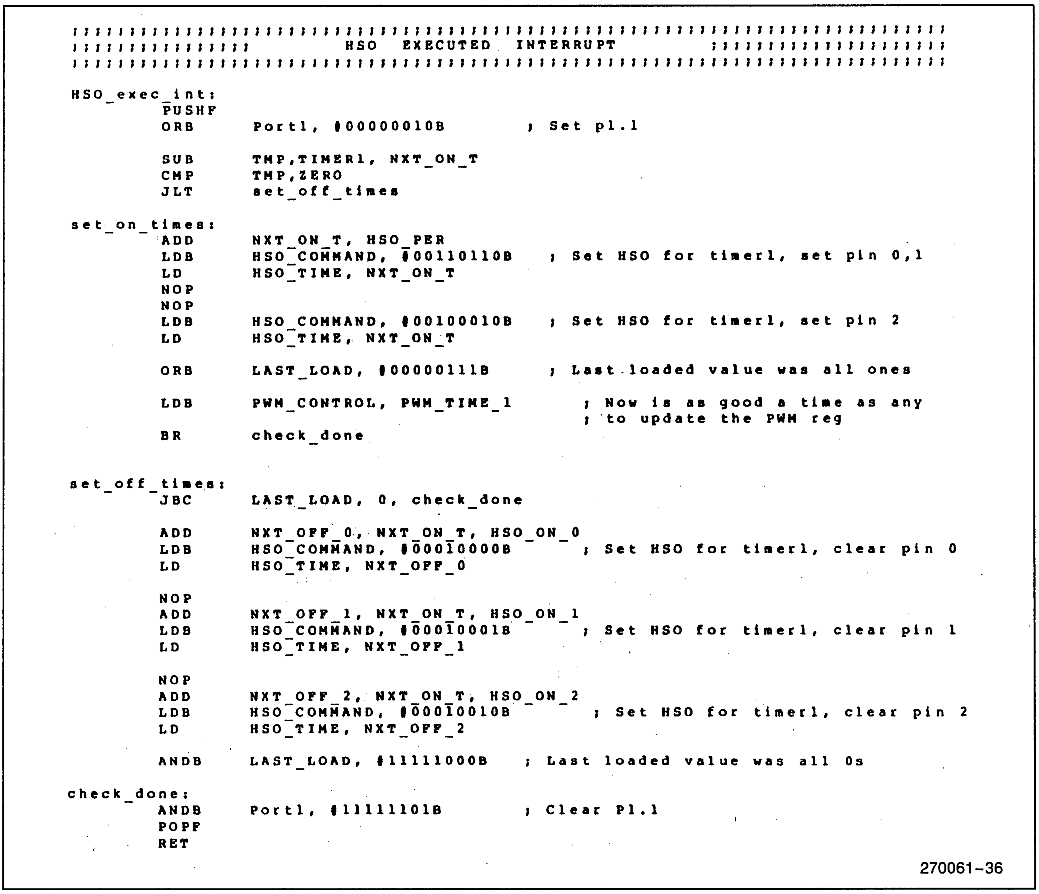
Listing 4-1a. Initializing the A to D to PWM Program



```
        LDB     HSO_COMMAND, #00110110B      ; Set HSO for timer1, set pin 0,1
        LD      HSO_TIME, NXT_ON_T           ; with interrupt
        NOP
        NOP
        LDB     HSO_COMMAND, #00100010B      ; Set HSO for timer1, set pin 2
        ADD     HSO_TIME, NXT_ON_T           ; without interrupt

        ORB     LAST_LOAD, #00000111B    ; Last loaded value was set all pins
        LDB     INT_MASK, #00001010B     ; Enable HSO and A/D interrupts
        LDB     INT_PENDING, #00001010B  ; Fake an A/D and HSO interrupt
        EI

loop:   ORB     Port1, #00000001B        ; set P1.0
        ADD     COUNT, #01
        ADDC    COUNT+2,zero
        ANDB    Port1, #11111110B        ; clear P1.0
        BR      loop
```

270061-35

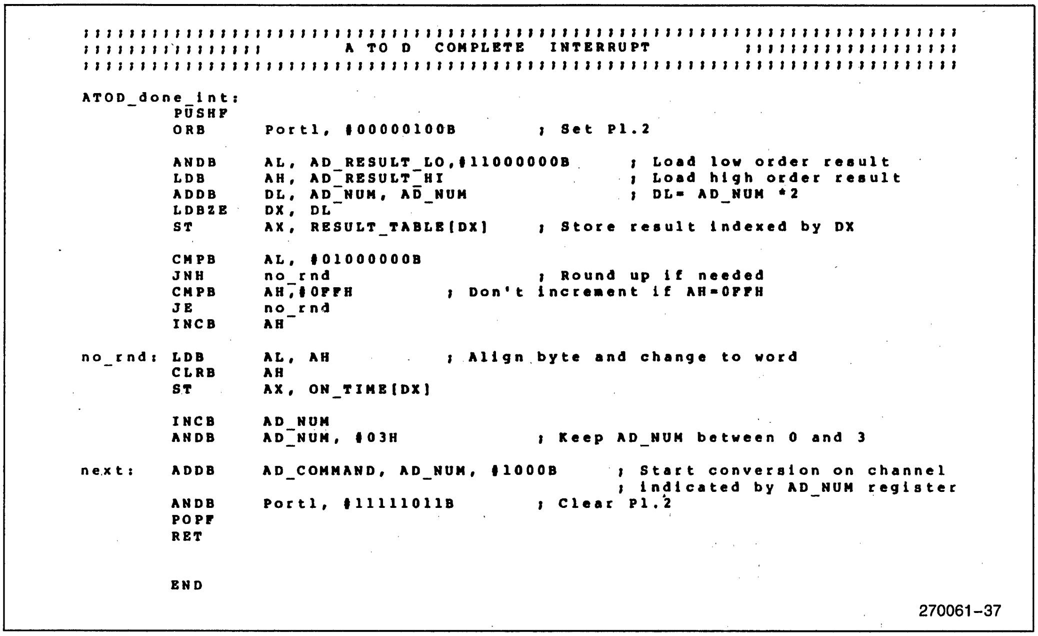
Listing 4-1a. Initializing the A to D to PWM program (Continued)



```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;       HSO  EXECUTED  INTERRUPT        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

HSO_exec_int:
        PUSHF
        ORB     Port1, #00000010B        ; Set p1.1

        SUB     TMP,TIMER1, NXT_ON_T
        CMP     TMP,ZERO
        JLT     set_off_times

set_on_times:
        ADD     NXT_ON_T, HSO_PER
        LDB     HSO_COMMAND, #00110110B    ; Set HSO for timer1, set pin 0,1
        LD      HSO_TIME, NXT_ON_T
        NOP
        NOP
        LDB     HSO_COMMAND, #00100010B    ; Set HSO for timer1, set pin 2
        LD      HSO_TIME, NXT_ON_T

        ORB     LAST_LOAD, #00000111B      ; Last loaded value was all ones

        LDB     PWM_CONTROL, PWM_TIME_1        ; Now is as good a time as any
                                               ; to update the PWM reg
        BR      check_done


set_off_times:
        JBC     LAST_LOAD, 0, check_done

        ADD     NXT_OFF_0, NXT_ON_T, HSO_ON_0
        LDB     HSO_COMMAND, #00010000B        ; Set HSO for timer1, clear pin 0
        LD      HSO_TIME, NXT_OFF_0

        NOP
        ADD     NXT_OFF_1, NXT_ON_T, HSO_ON_1
        LDB     HSO_COMMAND, #00010001B        ; Set HSO for timer1, clear pin 1
        LD      HSO_TIME, NXT_OFF_1

        NOP
        ADD     NXT_OFF_2, NXT_ON_T, HSO_ON_2
        LDB     HSO_COMMAND, #00010010B          ; Set HSO for timer1, clear pin 2
        LD      HSO_TIME, NXT_OFF_2

        ANDB    LAST_LOAD, #11111000B     ; Last loaded value was all 0s

check_done:
        ANDB    Port1, #11111101B         ; Clear P1.1
        POPF
        RET
```

270061-36

Listing 4-1b. Interrupt Driven HSO Routine



```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;          A TO D  COMPLETE  INTERRUPT          ;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ATOD_done_int:
        PUSHF
        ORB     Port1, #00000100B           ; Set P1.2

        ANDB    AL, AD_RESULT_LO,#11000000B       ; Load low order result
        LDB     AH, AD_RESULT_HI                  ; Load high order result
        ADDB    DL, AD_NUM, AD_NUM                ; DL= AD_NUM *2
        LDBZE   DX, DL
        ST      AX, RESULT_TABLE[DX]        ; Store result indexed by DX

        CMPB    AL, #01000000B
        JNH     no_rnd                      ; Round up if needed
        CMPB    AH,#0FFH              ; Don't increment if AH=0FFH
        JE      no_rnd
        INCB    AH

no_rnd: LDB     AL, AH                ; Align byte and change to word
        CLRB    AH
        ST      AX, ON_TIME[DX]

        INCB    AD_NUM
        ANDB    AD_NUM, #03H                ; Keep AD_NUM between 0 and 3

next:   ADDB    AD_COMMAND, AD_NUM, #1000B        ; Start conversion on channel
                                                  ; indicated by AD_NUM register
        ANDB    Port1, #11111011B           ; Clear P1.2
        POPF
        RET


        END
```

270061-37

Listing 4-1c. Interrupt Driven A to D Routine

The HSO routine shown in Listing 4-1b is slightly different than the one in section 3. All of the HSO lines turn on at the same time, only the turn-off-time is varied between lines. This action is what is most commonly required for multiple PWM outputs and simplifies the software. A comparison is made between Timer1 and the next HSO turn on time at the beginning of the routine. If the next turn on time has passed, then the on-times are loaded into the CAM, otherwise the off times are loaded.

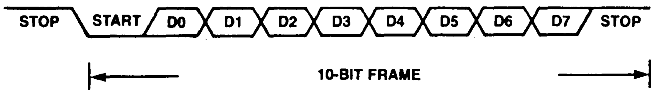
The maximum number of events in the CAM at any given time is 7. This occurs when the first line to turn off does so, causing the off-times for all of the lines to be loaded. For two of the lines there will be an offtime, an on-time, and the just loaded off-time. The other line (the one that just turned off) will have only the on-time and the just loaded off-time.

A/D conversions are performed by the code in Listing 4-1c about every 60 microseconds, 42 for the conversion, the rest for overhead. The A/D routine sets up the HSO and PWM on and off times. Since the A/D has a ten bit output, the most significant 8 bits are rounded up or down based on the least significant two bits.

4.2. Software Serial Port Using the HSIO Unit

There are many systems which require more than one serial port, an example is a system which must communicate with other computers and have an additional port for a local console. If the on-board UART is being used as an inter-processor link, the HSIO unit can be used to interface the 8096 to an additional asynchronous line.

Figure 4-1 shows the format of a standard 10-bit asynchronous frame. The start bit is used to synchronize the receiver to the transmitter; at the leading edge of the START bit the receiver must set up its timing logic to sample the incoming line in the center of each bit. Following the start bit are the eight data bits which are transmitted least significant bit first. The STOP bit is set to the opposite state of the START bit to guar-



270061-38

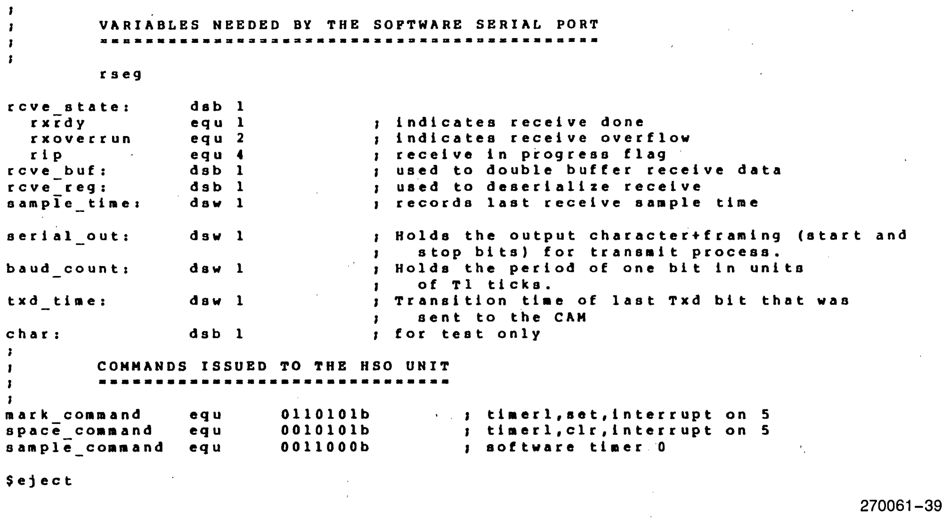
Figure 4-1. 10-bit Asynchronous Frame

antee that the leading edge of the START bit will cause a transition on the line; it also provides for a dead time on the line so that the receiver can maintain its synchronization.

The remainder of this section will show how a full-duplex asynchronous port can be built from the HSIO unit. There are four sections to this code:

1. Interface routines. These routines provide a procedural interface between the interrupt driven core of the software serial port and the remainder of the application software.
2. Initialization routine. This routine is called during the initialization of the overall system and sets up the various variables used by the software port.
3. Transmit ISR. This routine runs as an ISR (interrupt service routine) in response to an HSO interrupt interrupt. Its function is to serialize the data passed to it by the interface routines.
4. Receive ISRs. There are two ISRs involved in the receive process. One of them runs in response to an HSI interrupt and is used to synchronize the receive process at the leading edge of the start bit. The second receive ISR runs in response to an HSO generated software timer interrupt, this routine is scheduled to run at the center of each bit and is used to deserialize the incoming data.

The routines share the set of variables that are shown in Listing 4-2. These variables should be accessed only by the routines which make up the software serial port.



```
;
;       VARIABLES NEEDED BY THE SOFTWARE SERIAL PORT
;       ==========================================
;
        rseg

rcve_state:     dsb 1
  rxrdy         equ 1           ; indicates receive done
  rxoverrun     equ 2           ; indicates receive overflow
  rip           equ 4           ; receive in progress flag
rcve_buf:       dsb 1           ; used to double buffer receive data
rcve_reg:       dsb 1           ; used to deserialize receive
sample_time:    dsw 1           ; records last receive sample time

serial_out:     dsw 1           ; Holds the output character+framing (start and
                                ;   stop bits) for transmit process.
baud_count:     dsw 1           ; Holds the period of one bit in units
                                ;   of T1 ticks.
txd_time:       dsw 1           ; Transition time of last Txd bit that was
                                ;   sent to the CAM
char:           dsb 1           ; for test only
;
;       COMMANDS ISSUED TO THE HSO UNIT
;       ===============================
;
mark_command    equ     0110101b        ; timer1,set,interrupt on 5
space_command   equ     0010101b        ; timer1,clr,interrupt on 5
sample_command  equ     0011000b        ; software timer 0

$eject
```

270061-39

Listing 4-2. Software Serial Port Declarations

The table also shows the declarations for the commands issued to the HSO unit. In this example HSI.2 is used for receive data and HSO.5 is used for transmit data, although other HSI and HSO lines could have been used.

The interface routines are shown in Listing 4-3. Data is passed to the port by pushing the eight-bit character into the stack and calling *char_out,* which waits for any in-process transmission to complete and stores the character into the variable *serial_out.* As the data is stored the START and STOP bits are added to the data bits. The routine *char—in* is called when the application software requires a character from the port. The data is returned in the *ax* register in conformance to PLM 96 calling conventions. The routine *csts* can be called to determine if a character is available at the port before calling *char_in.* (If no character is available *char_in* will wait indefinitely).

The initialization routine is shown in Listing 4-4. This routine is called with the required baud rate in the

```
;
char_out:
; Output character to the software serial port
;
        pop     cx              ; the return address
        pop     bx              ; the character for output
        ldb     (bx+1),#01h     ; add the start and stop bits
        add     bx,bx           ;   to the char and leave as 16 bit
wait_for_xmit:
        cmp     serial_out,0    ; wait for serial_out=0 (it will be cleared by
        bne     wait_for_xmit   ;   the hso interrupt process)
        st      bx,serial_out   ; put the formatted character in serial_out
        br      [cx]            ; return to caller
;
csts:
; Returns "true" (ax<>0) if char_in has a character.
;
        clr     ax
        bbc     rcve_state,0,csts_exit
        inc     ax
csts_exit:
        ret
;
char_in:
; Get a character from the software serial port
;
                                ; wait for character ready
        bbc     rcve_state,0,char_in
        pushf                   ; set up a critical region
        andb    rcve_state,#not(rxrdy)
        ldbze   al,rcve_buf
        popf                    ; leave the critical region
        ret
```

270061-40

Listing 4-3. Software Serial Port Interface Routines

```
;
setup_serial_port:
; Called on system reset to intiate the software serial port.
;
        pop     cx              ; the return address
        pop     bx              ; the baud rate (in decimal)
        ld      dx,#0007h       ; dx:ax:=500,000 (assumes 12 Mhz crystal)
        ld      ax,#0A120h
        divu    ax,bx           ; calculate the baud count (500,000/baudrate)
        st      ax,baud_count
        st      0,serial_out    ; clear serial out
        ldb     iocl,#01100000b ; Enable HSO.5 and Txd
        bbs     ios0,6,$        ; Wait for room in the HSO CAM
                                ; and issue a MARK command.
        add     txd_time,timer1,20
        ldb     hso_command,#mark_command
        ld      hso_time,txd_time
        clrb    rcve_buf        ; clear out the receive variables
        clrb    rcve_reg
        clrb    rcve_state
        call    init_receive    ; setup to detect a start bit
        br      [cx]            ; return
```

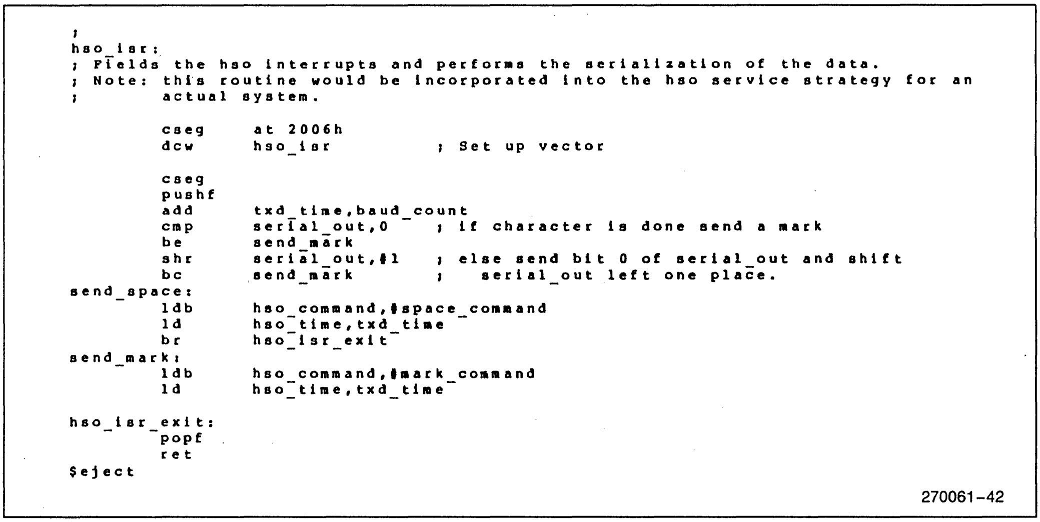
270061-41

Listing 4-4. Software Serial Port Initialization Routine

stack; it calculates the bit time from the baud rate and stores it in the variable *baud__count* in units of TIMER1 ticks. An HSO command is issued which will initiate the transmit process and then the remainder of the variables owned by the port are initialized. The routine *init__receive* is called to setup the HSI unit to look for the leading edge of the START bit.

The transmit process is shown in Listing 4-5. The HSO unit is used to generate an output command to the transmit pin once per bit time. If the *serial__out* register is zero a MARK (idle condition) is output. If the *serial__out* register contains data then the least significant bit is output and the register shifted right one place. The framing information (START and STOP bits) are appended to the actual data by the interface routines. Note that this routine will be executed once per bit time whether or not data is being transmitted. It would be possible to use this routine for additional low resolution timing functions with minimal overhead.

The receive process consists of an initialization routine and two interrupt service routines, *hsi__isr* and *software__timer__isr.* The listings of these routines are shown in Listings 4-6a,4-6b, and 4-6c respectively. The

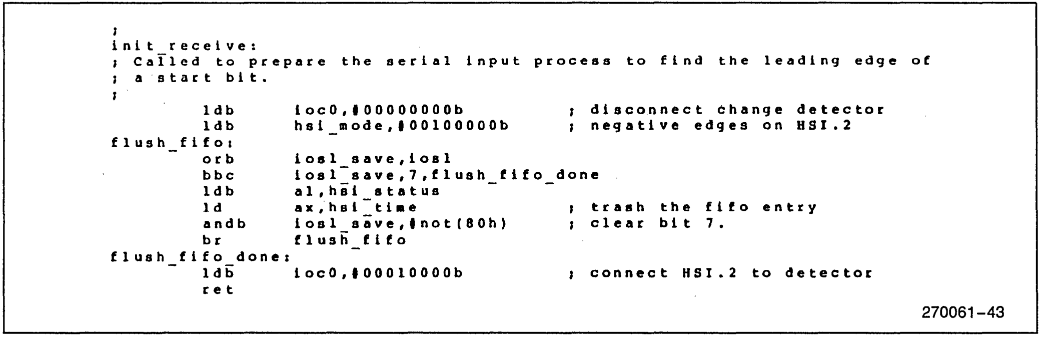


```
;
hso_isr:
; Fields the hso interrupts and performs the serialization of the data.
; Note: this routine would be incorporated into the hso service strategy for an
;       actual system.

        cseg    at 2006h
        dcw     hso_isr         ; Set up vector

        cseg
        pushf
        add     txd_time,baud_count
        cmp     serial_out,0    ; if character is done send a mark
        be      send_mark
        shr     serial_out,#1   ; else send bit 0 of serial_out and shift
        bc      send_mark       ;   serial_out left one place.
send_space:
        ldb     hso_command,#space_command
        ld      hso_time,txd_time
        br      hso_isr_exit
send_mark:
        ldb     hso_command,#mark_command
        ld      hso_time,txd_time

hso_isr_exit:
        popf
        ret
$eject
```

270061-42

Listing 4-5. Software Serial Port Transmit Process

Listing 4-6. Receive Process



```
;
init_receive:
; Called to prepare the serial input process to find the leading edge of
; a start bit.
;
        ldb     ioc0,#00000000b         ; disconnect change detector
        ldb     hsi_mode,#00100000b     ; negative edges on HSI.2
flush_fifo:
        orb     iosl_save,iosl
        bbc     iosl_save,7,flush_fifo_done
        ldb     al,hsi_status
        ld      ax,hsi_time             ; trash the fifo entry
        andb    iosl_save,#not(80h)     ; clear bit 7.
        br      flush_fifo
flush_fifo_done:
        ldb     ioc0,#00010000b         ; connect HSI.2 to detector
        ret
```

270061-43

Listing 4-6a. Software Serial Port Receive Initialization

```
;
hsi_isr:
; Fields interrupts from the HSI unit, used to detect the leading edge
; of the START bit
; Note: this routine would be incorporated into the HSI strategy of an actual
; system.
;
        cseg at 2004h
        dcw     hsi_isr                 ; setup the interrupt vector

        cseg
        pushf
        push    ax
        ldb     al,hsi_status
        ld      sample_time,hsi_time
        bbc     al,4,exit_hsi
        bbs     ios0,7,$                ; wait for room in HSO holding reg
        ld      ax,baud_count           ; send out sample command in 1/2
        shr     ax,#1                   ; bit time
        add     sample_time,ax
        ldb     hso_command,#sample_command
        st      sample_time,hso_time
        ldb     ioc0,#00000000b         ; disconnect hsi.2 from change detector
exit_hsi:
        pop     ax
        popf
        ret
```

270061-44

Listing 4-6b. Software Serial Port Start Bit Detect

```
;
software_timer_isr:
; Fields the software timer interrupt, used to deserialize the incomming data.
; Note: this routine would be incorporated into the software timer stategy
; in an actual system.
;
        cseg at 200ah
        dcw     software_timer_isr      ; setup vector

        cseg
        pushf
        orb     ios1_save,ios1
        andb    ios1_save,#not(01h)     ; clear bit 0
        andb    0,rcve_state,#0fch      ; All bits except rxrdy and overrun=0
        bne     process_data
process_start_bit:
        bbc     hsi_status,5,start_ok
        call    init_receive
        br      software_timer_exit
start_ok:
        orb     rcve_state,#rip ; set receive in progress flag
        br      schedule_sample

process_data:
        bbs     rcve_state,7,check_stopbit
        shrb    rcve_reg,#1
        bbc     hsi_status,5,datazero
        orb     rcve_reg,#80h   ; set the new data bit
datazero:
        addb    rcve_state,#10h ; increment bit count
        br      schedule_sample

check_stopbit:
        bbc     hsi_status,5,$  ; DEBUG ONLY
        ldb     rcve_buf,rcve_reg
        orb     rcve_state,#rxrdy
        andb    rcve_state,#03h ; Clear all but ready and overrun bits
        call    init_receive
        br      software_timer_exit

schedule_sample:
        bbs     ios0,7,$        ; wait for holding reg empty
        ldb     hso_command,#sample_command
        add     sample_time,baud_count
        st      sample_time,hso_time

software_timer_exit:
        popf
        ret
```

270061-45

Listing 4-6c. Software Serial Port Data Reception

start is detected by the *hsi_isr* which schedules a software timer interrupt in one-half of a bit time. This first sample is used to verify that the START bit has not ended prematurely (a protection against a noisy line). The software timer service routine uses the variable *rcve_state* to determine whether it should check for a valid START bit, deserialize data, or check for a valid STOP bit. When a complete character has been received it is moved to the receive buffer and *init_receive* is called to set up the receive process for the next character. This routine is also called when an error (e.g., invalid START bit) is detected.

Appendix C contains the complete listing of the routines and the simple loop which was used to initialize them and verify their operation. The test was run for several hours at 9600 baud with no apparent malfunction of the port.

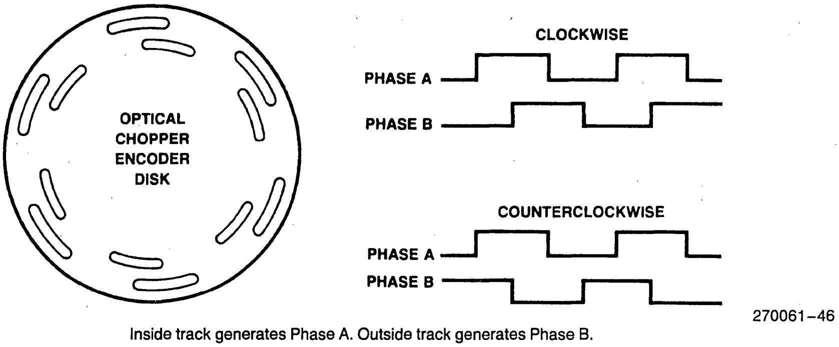
4.3. Interfacing an Optical Encoder to the HSI Unit

Optical encoders are among one of the more popular devices used to determine position of rotating equipment. These devices output two pulse trains with edges that occur from 2 to 4000 times a revolution.

Frequently there is a third line which generates one pulse per revolution for indexing purposes. Figure 4-2 shows a six line encoder and typical waveforms. As can be seen, the two waveforms provide the ability to determine both position and direction. Since a microcontroller can perform real time calculations it is possible to determine velocity and acceleration from the position and time information.

Interfacing to the encoder can be an interesting problem, as it requires connecting mechanically generated electrical signals to the HSI unit. The problems arise because it is difficult to obtain the exact nature of the signals under all conditions.

The equipment used in the lab was a Pittman 9400 series gearmotor with a 600 line optical encoder from Vernitech. The encoder has to be carefully attached to the shaft to minimize any runout or endplay. Fortunately, Pitmann has started marketing their motors with ball bearings and optical encoders already installed. It is recommended that the encoder be mounted to the motor using the exact specifications of the encoder manufacturer and/or a good machine shop.



Inside track generates Phase A. Outside track generates Phase B.

Figure 4-2. Optical Encoder and Waveforms

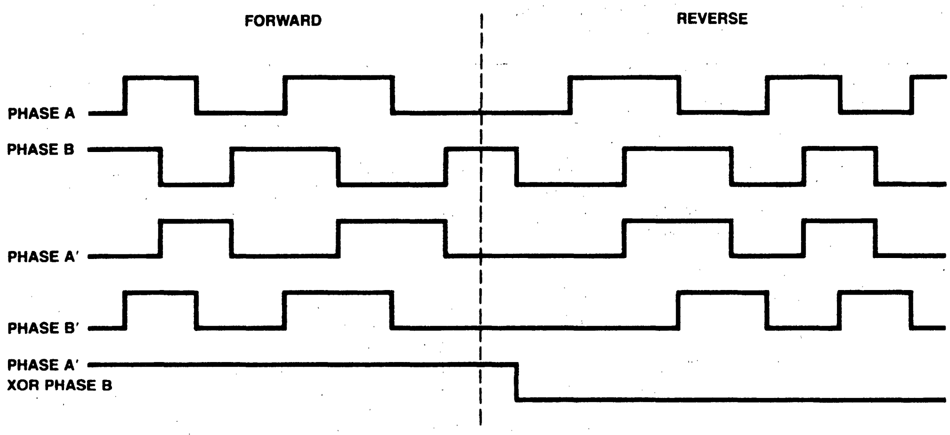
Digital filtering external to the 8096 is used on the encoder signals. The idealized signals coming from the encoder and after the digital filter are shown in Figure 4-3. The circuitry connecting the encoder to the 8096 requires only two chips. A one-shot constructed of XOR gates generates pulses on each edge of each signal. The pulses generated by Phase A are used to clock the signal from Phase B and vice versa. The hardware is shown in Figure 4-4. CMOS parts are used to reduce loading on the encoder so that buffers are not needed. Note that T2CLK is clocked on both edges of both filtered phases.

By using this method repetitive edges on a single phase without an edge on the other phase will not be passed on to the 8096. Repetitive edges on a phase can occur when the motor is stopped and vibrates or when it is changing direction. The digital filtering technique causes a little more delay in the signal at slow speeds than an analog filter would, but the simplicity trade off is worthwhile. The net effect of digital filtering is losing the ability to determine the first edge after a direction change. This does not affect the count since the first edge in both directions is lost.

If it is desired to determine when each edge occurs before filtering, the encoder outputs can be attached directly to the 8096. As these would be input signals, Port 0 is the most likely choice for connection. It would not be required to connect these lines to the HSI unit, as the information on them would only be needed when the motor is going very slowly.

The motor is driven using the PWM output pin for power control and a port pin for direction control. The 8096 drives a 7438 which drives 2 opto-isolators. These in turn drive two VFETs. A MOV (Metal Oxide Varistor, a type of transient absorber) is used to protect the VFETs, and a capacitor filters the PWM to get the best motor performance. Figure 4-5 shows the driver circuitry. To avoid noise getting into the 8096 system, the ±15 volt power supply is isolated from the 8096 logic power supply.

This is the extent of the external circuitry required for this example. All of the counting and direction detection are done by the 8096. There are two sections to the example: driving the motor and interfacing to the encoder. The motor driver uses proportional control with



NOTES:
Phase A' is Phase A clocked by Phase B
Phase B' is Phase B clocked by Phase A

Figure 4-3. Filtered Encoder Waveforms

some modifications and a braking algorithm. Since the main point of this example is I/O interfacing, the motor driver will be briefly described at the end of this section.

In order to interface to the encoder it is necessary to know the types of waveforms that can be expected. The motor was accelerated and decelerated many times using different maximum voltages. It was found that the

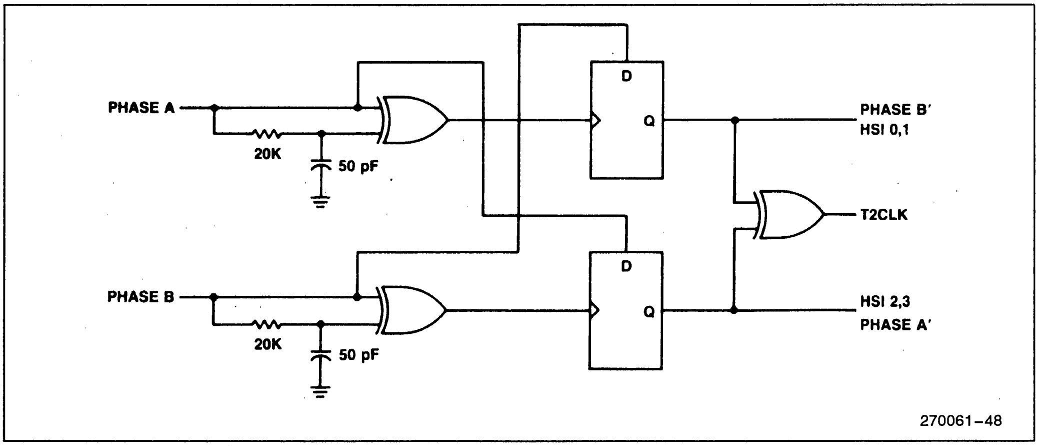


Figure 4-4. Schematic of Optical Encoder to 8096 Interface

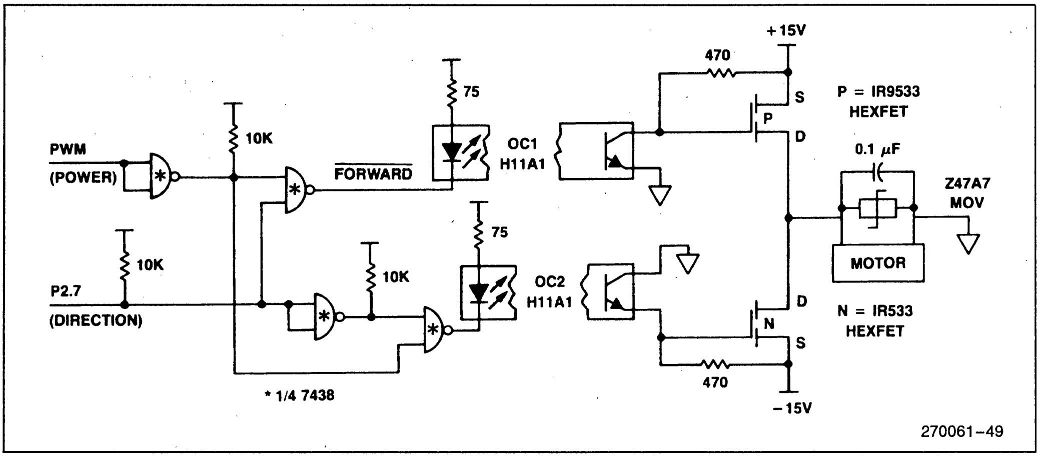


Figure 4-5. Motor Driver Circuitry

motor would decelerate smoothly until the time between encoder edges was around 100 microseconds. At this point the motor would either continue to decelerate slowly, or would suddenly stop and reverse. The latter case is the one that was most problematic.

After a brief overview, each section of the program will be described separately, with the complete listing included in the Appendix D. In order to make debugging easier, as well as to provide insight into how the program is working, I/O port 1 is used to indicate the program status. This information consists of which routine the program is in and under which mode it is operating. The main program sections are: Main loop, HSI interrupt, Timer 2 check, and Motor drive. There are also minor sections such as initialization, timer overflow handling, and software timer handling. Tying everything together is some overhead and glue. Where the glue is not obvious it will be discussed, otherwise it can be derived from the listings.

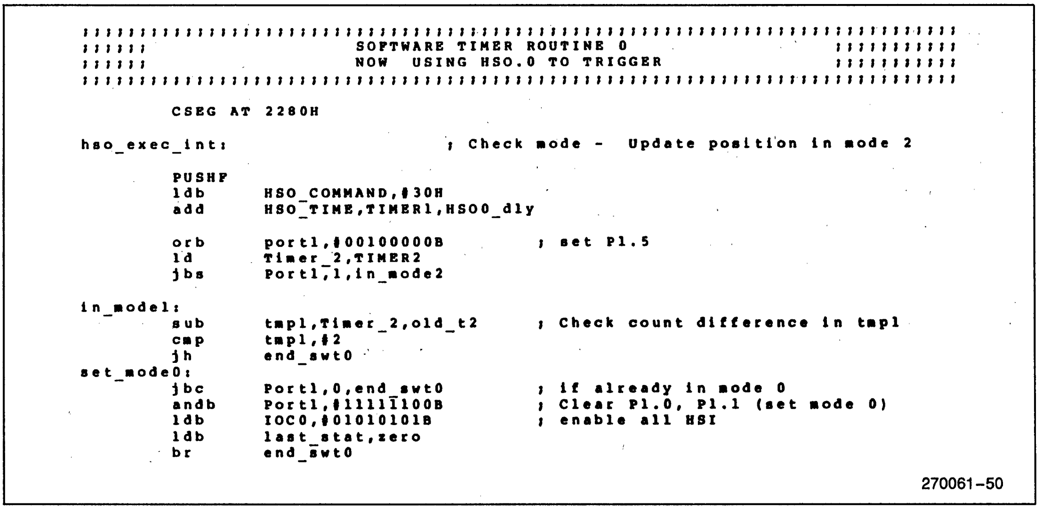
The program is a main loop which does nothing except serve as a place for the program to go when none of the interrupt routines are being run. All of the processing is done on an interrupt basis.

There are three basic software modes which are invoked depending on the speed of the motor. The modes referred to as 0, 1 and 2, in order from slowest to fastest operation. When the program is running the operating mode is indicated by the lower 2 bits of Port 1, with the following coding:

P1.0	P1.1	Mode	Description
0	0	0	HSI looks at every edge
1	0	1	HSI looks at Phase A edges only
0	1	2	Timer 2 used instead of HSI
1	1	2	(alternate form of above)

The example is easiest to see if mode 2 is described first, followed by mode 1 then mode 0. In mode 2 Timer 2 is used to count edges on the incoming signal. A software timer routine, which is actually run using HSO.0, uses the Timer 2 value to update a LONG (32-bit) software counter labeled *POSITION*. The HSO routine runs every 260 microseconds. The HSO.0 interrupt is used instead of an actual software timer because of the ability to easily unmask it while other software timer routines are running.

In the code in Listing 4-7, the mode is first determined. For the first pass ignore the code starting with the label *in__mode__1*. Starting with *in__mode__2* the counter is incremented or decremented based on bit zero of DIRECT. If DIRECT.0 = 0 the motor is going backward, if it is a 1 the motor is going forward. Next the count difference is checked to see if it is slow enough to go into mode 1. If not the routine returns to the code it was running when the interrupt occurred.



```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;                      SOFTWARE TIMER ROUTINE 0              ;;;;;;;;;;;;
;;;;;;                      NOW  USING HSO.0 TO TRIGGER           ;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        CSEG AT 2280H

hso_exec_int:                   ; Check mode -  Update position in mode 2

        PUSHF
        ldb     HSO_COMMAND,#30H
        add     HSO_TIME,TIMER1,HSO0_dly

        orb     port1,#00100000B        ; set P1.5
        ld      Timer_2,TIMER2
        jbs     Port1,1,in_mode2

in_model:
        sub     tmp1,Timer_2,old_t2     ; Check count difference in tmp1
        cmp     tmp1,#2
        jh      end_swt0
set_mode0:
        jbc     Port1,0,end_swt0        ; if already in mode 0
        andb    Port1,#11111100B        ; Clear P1.0, P1.1 (set mode 0)
        ldb     IOC0,#01010101B         ; enable all HSI
        ldb     last_stat,zero
        br      end_swt0
```

270061-50

Listing 4-7. Motor Control HSO.0 Timer Routine

```
in_mode2:
        sub     delta_p,timer_2,tmr2_old            ; get timer2 count difference
        ld      tmr2_old,timer_2

        jbc     direct,0,in_rev

in_fwd: add     position,delta_p
        addc    position+2,zero
        br      chk_mode

in_rev: sub     position,delta_p
        subc    position+2,zero

chk_mode:
        sub     tmpl,Timer_2,old_t2         ; Check count difference in tmpl
        cmp     tmpl,#5                     ; set model if count is too low
        jgt     end_swt0                    ; count <= 5

set_model:
        andb    Portl,#11111101B            ; Clear Pl.1, set Pl.0 (set mode 1)
        orb     Portl,#00000001B
        ldb     IOC0,#00000101B             ; enable HSI 0 and 1
        ld      zero, HSI_TIME
        sub     lastl_time,Timerl,min_hsil
                        ; set up so (time-last2_time)>min_hsil on next HSI
clr_hsi:
        ld      ZERO, HSI_TIME
        andb    iosl_bak,#01111111B                 ; clear bit 7
        orb     iosl_bak,iosl
        jbs     iosl_bak,7,clr_hsi          ; If hsi is triggered then clear hsi

end_swt0:
        ld      old_t2,TIMER_2
        andb    portl,#11011111B            ; clear Pl.5
        POPF
        ret
```

270061-51

Listing 4-7. Motor Control HSO.0 Timer Routine (Continued)

If the pulse rate is slow enough to go to mode 1, the transition is made by enabling HSI.0 and HSI.1. Both of these lines are connected to the same encoder line, with HSI.0 looking for rising edges and HSI.1 looking for falling edges. The *HSI__TIME* register is read to speed up clearing the HSI FIFO and the *LAST1__TIME* value is set up so the mode 1 routine does not immediately put the program into another mode. The HSI FIFO is then cleared, the Timer 2 value used throughout this routine is saved, and the routine returns.

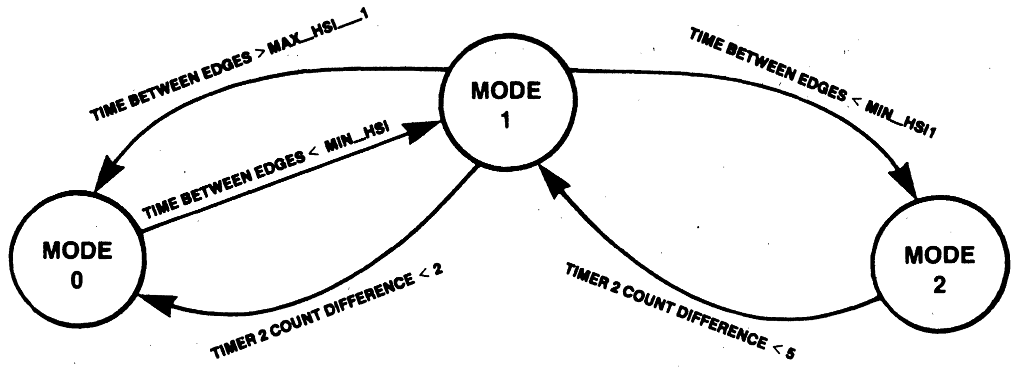
This routine still runs in modes 0 and 1, but in an abbreviated form. The section of code starting with the label *in__model* checks to see if the pulses are coming in so slowly that both HSI lines can be checked. If this is the case then all of the HSIs are enabled and the program returns. This routine is the secondary method for going from mode 1 to mode 0, the primary method is by checking the time between edges during the HSI routine, which will be described later.

The HSO routine will enable mode 0 from mode 1 if two edges are not received every 260 microseconds. The primary method, (under the HSI routine), can only enable mode 0 after an edge is received. This could cause a problem if the last 2 edges on Phase A before the encoder stops were too close to enable mode 0. If this happened, mode 0 would not be enabled until after the encoder started again, resulting in missed edges on Phase B. Using the HSO routine to switch from mode 1 to mode 0 eliminates this problem.

Figure 4-6 shows a state diagram of how the mode switching is done. As can be seen, there are two sources for most of the mode decisions. This helps avoid problems such as the one mentioned above.

When either Mode 1 or Mode 0 is enabled the HSI interrupt routine performs the counting of edges, while the HSO routine only ensures that the correct mode is running. The routines for modes 0 and 1 share the same initialization and completion sections, with the main body of code being different.

The initialization routine is similar to many HSI routines. The flags are checked to ensure that the HSI FIFO data is valid, and then the FIFO is read. Next, the main body of code (for either mode 0 or mode 1) is


MODE 0
MODE 1
MODE 2
TIME BETWEEN EDGES > MAX_HSI_1
TIME BETWEEN EDGES < MIN_HSI
TIMER 2 COUNT DIFFERENCE < 2
TIME BETWEEN EDGES < MIN_HSI1
TIMER 2 COUNT DIFFERENCE < 5

270061-52

NOTES:
Mode 0: HSI Examines edges on Phase A and B
Mode 1: HSI Examines edges on Phase A only
Mode 2: TIMER 2 stores edgecount

Figure 4-6. Mode State Diagram

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;           HSI DATA AVAILABLE INTERRUPT ROUTINE          ;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;  This routine keeps track of the current time and position of the motor.
;  The upper word of information is provided by the timer overflow routine.

        CSEG AT 2400H
now_mode_1:     br      in_mode_1       ; used to save execution time for
no_intl:        br      no_int          ; worst case loop

hsi_data_int:   pushf
        orb     portl,#01000000B        ; set P1.6
        andb    iosl_bak,#01111111B     ; Clear iosl_bak.7
        orb     iosl_bak,iosl
        jbc     iosl_bak,7,no_intl      ; If hsi is not triggered then
                                        ; jump to no_int
get_values:
        ld      timer_2,TIMER2
        andb    hsi_s0,HSI_STATUS,#01010101B
        ld      time, HSI_TIME

        jbs     portl,0,now_mode_1      ; jump if in mode 1

In_mode_0:


;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;             INSERT  BODY  OF  ROUTINE
;;;;;;;;;;;;;;;;;;;


load_lasts:
        ld      tmr2_old,timer_2
no_cnt: andb    iosl_bak,#01111111B     ; clr bit 7
        orb     iosl_bak,iosl
        jbc     iosl_bak,7,no_int
again:  br      get_values

no_int: andb    portl,#10111111B        ; Clear P1.6
        popf
        ret     ; end of hsi_data interrupt routine
                ; Routine for mode 1 follows and then returns to "load_lasts"
$EJECT
```

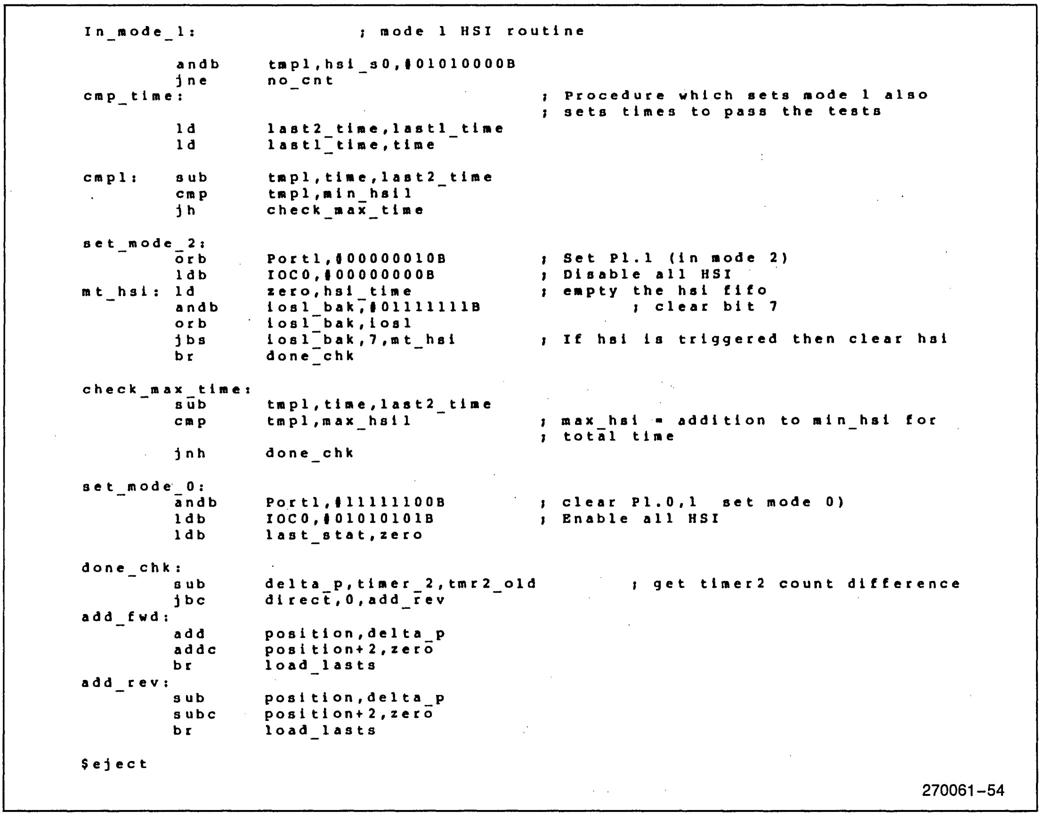
270061-53

Listing 4-8. Motor Control HSI Data Available Routine

run. At the end time and count values are saved and the holding register is checked for another event. Listing 4-8 contains the initialization and completion sections of the HSI routine.

Listing 4-9 is the main body of the Mode 1 routine. Before any calculations are done in Mode 1, the incoming pulse period is measured to see if it is too fast or too slow for mode 1. The time period between two edges is used so that the duty cycle of the waveform will not affect mode switching. If it is determined that Mode 2 should be set, Port 1.1 is set, all of the HSI lines are disabled, and the HSI fifo is cleared. If Mode 0 is to be set all of the HSI lines are enabled and the variable *LAST_STAT* is cleared. LAST_STAT = 0 is used as a flag to indicate the first HSI interrupt in Mode 0 after Mode 1. After the mode checking and setting are complete the incremental value in Timer 2 is used to update *POSITION*. The program then returns to the completion section of the routine.

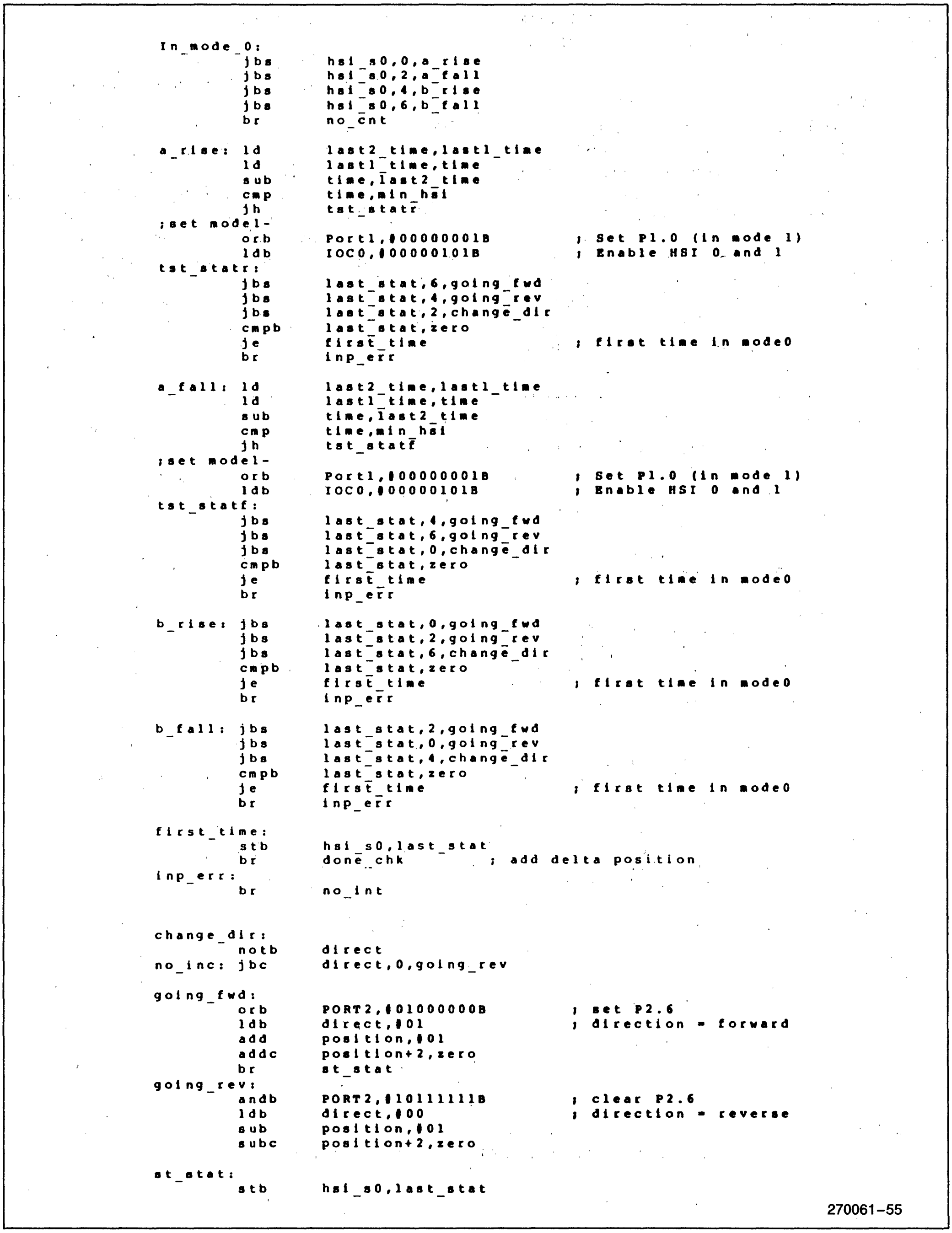
There is a lot more code used in Mode 0 than in Mode 1, most of which is due to the multiple jump statements that determine the current and previous state of the HSI pins. In order to save execution time several blocks of code are repeated as can be seen in Listing 4-10. The first determination is that of which edge had occurred. If a Phase A edge was detected the *LAST1_TIME* and *LAST2_TIME* variables are updated so a reference to the pulse frequency will be available. These are the same variables used under Mode 1. A test is also made to see if the edges are coming fast enough to warrant being in Mode 1, if they are, the switch is made. If the last edge detected was on Phase B, the information is used only to determine direction.



```
In_mode_1:                 ; mode 1 HSI routine

          andb     tmp1,hsi_s0,#01010000B
          jne      no_cnt
cmp_time:                                       ; Procedure which sets mode 1 also
                                                ; sets times to pass the tests
          ld       last2_time,last1_time
          ld       last1_time,time

cmp1:     sub      tmp1,time,last2_time
          cmp      tmp1,min_hsi1
          jh       check_max_time

set_mode_2:
          orb      Port1,#00000010B             ; Set P1.1 (in mode 2)
          ldb      IOC0,#00000000B              ; Disable all HSI
mt_hsi:   ld       zero,hsi_time                ; empty the hsi fifo
          andb     ios1_bak,#01111111B                ; clear bit 7
          orb      ios1_bak,ios1
          jbs      ios1_bak,7,mt_hsi            ; If hsi is triggered then clear hsi
          br       done_chk

check_max_time:
          sub      tmp1,time,last2_time
          cmp      tmp1,max_hsi1                ; max_hsi = addition to min_hsi for
                                                ; total time
          jnh      done_chk

set_mode_0:
          andb     Port1,#11111100B             ; clear P1.0,1  set mode 0)
          ldb      IOC0,#01010101B              ; Enable all HSI
          ldb      last_stat,zero

done_chk:
          sub      delta_p,timer_2,tmr2_old           ; get timer2 count difference
          jbc      direct,0,add_rev
add_fwd:
          add      position,delta_p
          addc     position+2,zero
          br       load_lasts
add_rev:
          sub      position,delta_p
          subc     position+2,zero
          br       load_lasts

$eject
```

270061-54

Listing 4-9. Motor Control Mode 1 Routines



```
In_mode_0:
                jbs     hsi_s0,0,a_rise
                jbs     hsi_s0,2,a_fall
                jbs     hsi_s0,4,b_rise
                jbs     hsi_s0,6,b_fall
                br      no_cnt

a_rise:         ld      last2_time,last1_time
                ld      last1_time,time
                sub     time,last2_time
                cmp     time,min_hsi
                jh      tst_statr
;set mode1-
                orb     Port1,#00000001B        ; Set P1.0 (in mode 1)
                ldb     IOC0,#00000101B         ; Enable HSI 0 and 1
tst_statr:
                jbs     last_stat,6,going_fwd
                jbs     last_stat,4,going_rev
                jbs     last_stat,2,change_dir
                cmpb    last_stat,zero
                je      first_time              ; first time in mode0
                br      inp_err

a_fall:         ld      last2_time,last1_time
                ld      last1_time,time
                sub     time,last2_time
                cmp     time,min_hsi
                jh      tst_statf
;set mode1-
                orb     Port1,#00000001B        ; Set P1.0 (in mode 1)
                ldb     IOC0,#00000101B         ; Enable HSI 0 and 1
tst_statf:
                jbs     last_stat,4,going_fwd
                jbs     last_stat,6,going_rev
                jbs     last_stat,0,change_dir
                cmpb    last_stat,zero
                je      first_time              ; first time in mode0
                br      inp_err

b_rise:         jbs     last_stat,0,going_fwd
                jbs     last_stat,2,going_rev
                jbs     last_stat,6,change_dir
                cmpb    last_stat,zero
                je      first_time              ; first time in mode0
                br      inp_err

b_fall:         jbs     last_stat,2,going_fwd
                jbs     last_stat,0,going_rev
                jbs     last_stat,4,change_dir
                cmpb    last_stat,zero
                je      first_time              ; first time in mode0
                br      inp_err

first_time:
                stb     hsi_s0,last_stat
                br      done_chk        ; add delta position
inp_err:
                br      no_int


change_dir:
                notb    direct
no_inc:         jbc     direct,0,going_rev

going_fwd:
                orb     PORT2,#01000000B        ; set P2.6
                ldb     direct,#01              ; direction = forward
                add     position,#01
                addc    position+2,zero
                br      st_stat
going_rev:
                andb    PORT2,#10111111B        ; clear P2.6
                ldb     direct,#00              ; direction = reverse
                sub     position,#01
                subc    position+2,zero

st_stat:
                stb     hsi_s0,last_stat
```

270061-55

Listing 4-10. Motor Control Mode 0 Routines

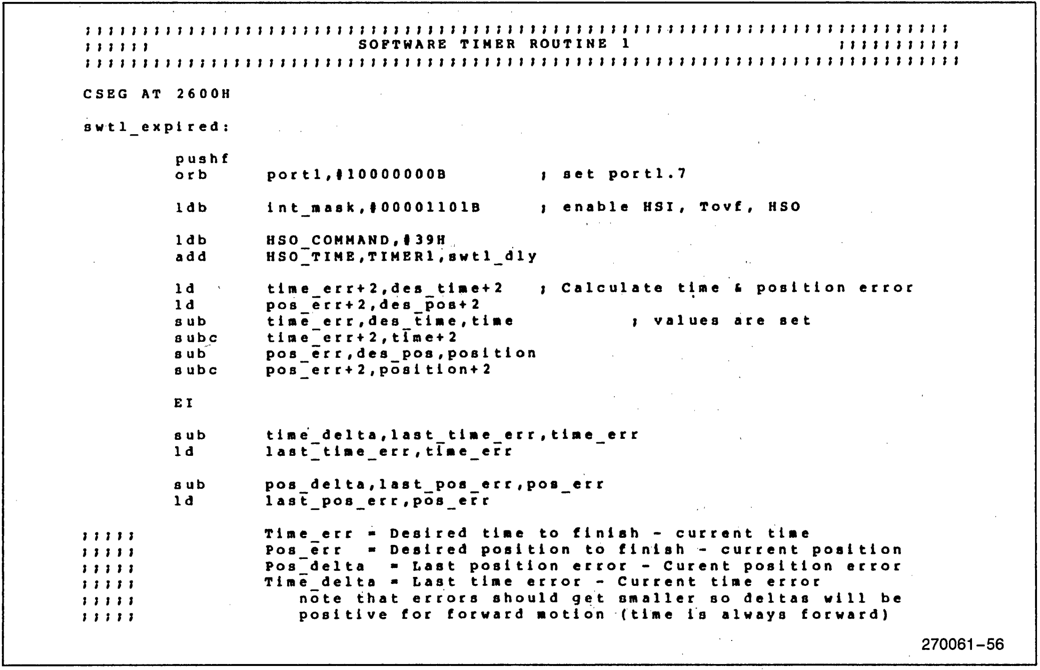
After mode correctness is confirmed and the *LASTx__TIME* values are updated the *LAST__STAT* (Last Status) variable is used to determine the current direction of travel. The POSITION value is then updated in the direction specified by the last two edges and the status is stored. Note that the first time in Mode 0 after being in Mode 1, the Mode 1 *done__chk* routine is used to update POSITION, instead of the routines *going__fwd* and *going__rev* from the Mode 0 section of code. The completion section of code is then executed.

Providing the PWM value to drive the motor is done by a routine running under Software Timer 1. The first section of code, shown in Listing 4-11a, has to do with calculating the position and timer errors. Listing 4-11b shows the next section of code where the power to be supplied to the motor is calculated. First the direction is checked and if the direction is reverse the absolute value of the error is taken. If the error is greater than 64K counts, the PWM routine is loaded with the maximum value. The next check is made to see if the motor is close enough to the desired location that the power to it should be reversed, (i.e., enter the Braking mode). If the motor is very close to the position or has slowed to the point that is likely to turn around, the *Hold__Position mode is entered.*

The determination of which modes are selected under what conditions was done empirically. All of the parameters used to determine the mode are kept in RAM so they can be easily changed on the fly instead of by re-assembling the program. The parameters in the listing have been selected to make the motor run, but have not been optimized for speed or stability. A diagram of the modes is shown in Figure 4-7.

In the *Hold__Position* mode power is eased onto the motor to lock it into position. Since the motor could be stopped in this mode, some integral control is needed, as proportional control alone does not work well when the error is small and the load is large. The BOOST variable provides this integral control by increasing the output a fixed amount every time period in which the

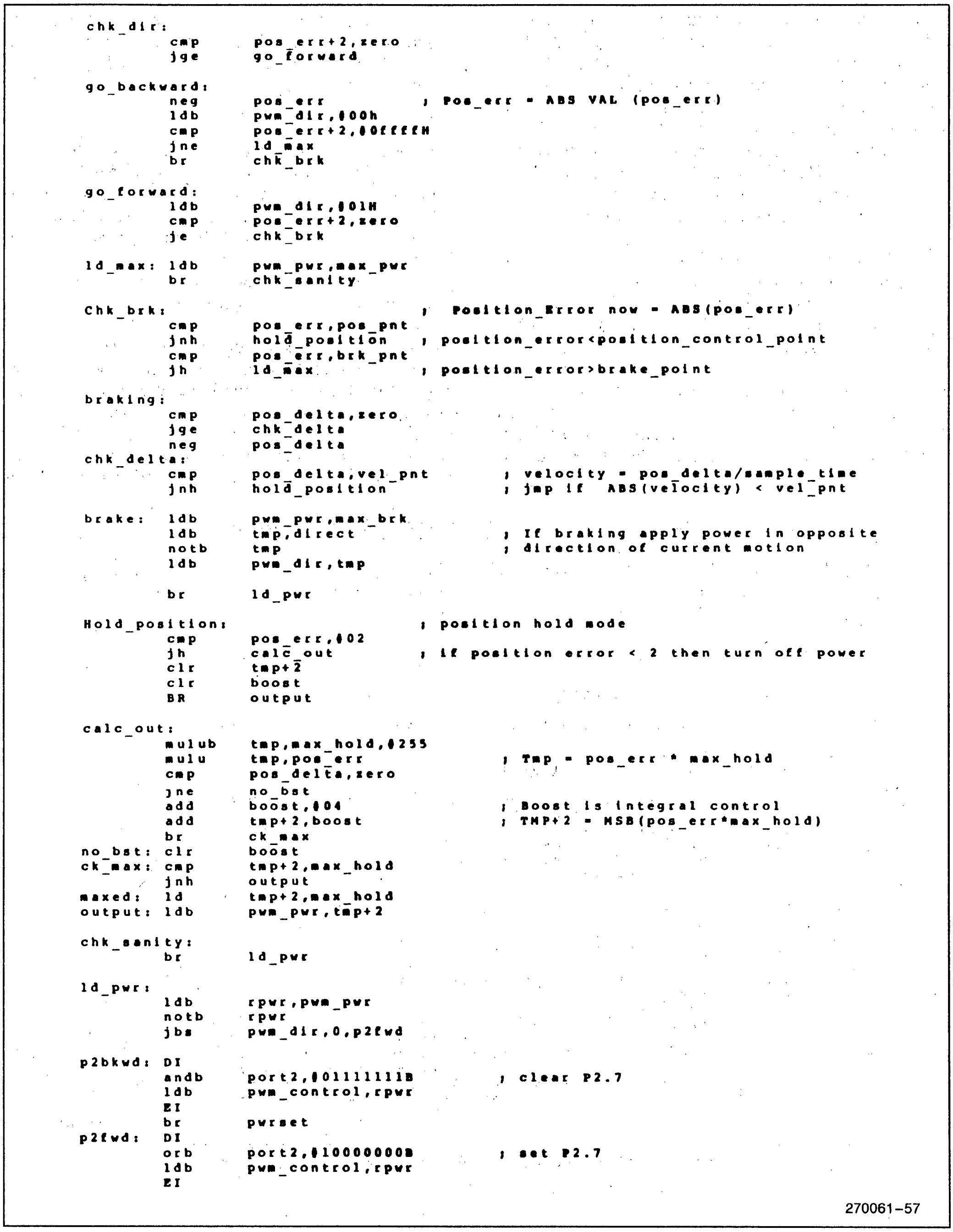
Listing 4-11. Motor Control Software Timer 1 Routine



```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;                      SOFTWARE TIMER ROUTINE 1                ;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CSEG AT 2600H

swtl_expired:

        pushf
        orb     portl,#10000000B        ; set portl.7

        ldb     int_mask,#00001101B     ; enable HSI, Tovf, HSO

        ldb     HSO_COMMAND,#39H
        add     HSO_TIME,TIMER1,swtl_dly

        ld      time_err+2,des_time+2   ; Calculate time & position error
        ld      pos_err+2,des_pos+2
        sub     time_err,des_time,time          ; values are set
        subc    time_err+2,time+2
        sub     pos_err,des_pos,position
        subc    pos_err+2,position+2

        EI

        sub     time_delta,last_time_err,time_err
        ld      last_time_err,time_err

        sub     pos_delta,last_pos_err,pos_err
        ld      last_pos_err,pos_err

;;;;;           Time_err  = Desired time to finish - current time
;;;;;           Pos_err   = Desired position to finish - current position
;;;;;           Pos_delta  = Last position error - Curent position error
;;;;;           Time_delta = Last time error - Current time error
;;;;;              note that errors should get smaller so deltas will be
;;;;;              positive for forward motion (time is always forward)
```

270061-56

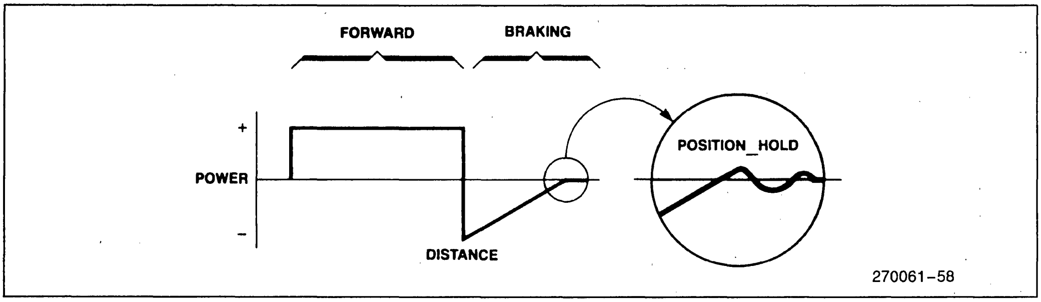
Listing 4-11a. Motor Control Software Position Counter



```
chk_dir:
        cmp     pos_err+2,zero
        jge     go_forward

go_backward:
        neg     pos_err          ; Pos_err = ABS VAL (pos_err)
        ldb     pwm_dir,#00h
        cmp     pos_err+2,#0ffffH
        jne     ld_max
        br      chk_brk

go_forward:
        ldb     pwm_dir,#01H
        cmp     pos_err+2,zero
        je      chk_brk

ld_max: ldb     pwm_pwr,max_pwr
        br      chk_sanity

Chk_brk:                         ; Position_Error now = ABS(pos_err)
        cmp     pos_err,pos_pnt
        jnh     hold_position    ; position_error<position_control_point
        cmp     pos_err,brk_pnt
        jh      ld_max           ; position_error>brake_point

braking:
        cmp     pos_delta,zero
        jge     chk_delta
        neg     pos_delta
chk_delta:
        cmp     pos_delta,vel_pnt      ; velocity = pos_delta/sample_time
        jnh     hold_position          ; jmp if  ABS(velocity) < vel_pnt

brake:  ldb     pwm_pwr,max_brk        ; If braking apply power in opposite
        ldb     tmp,direct             ; direction of current motion
        notb    tmp
        ldb     pwm_dir,tmp

        br      ld_pwr

Hold_position:                   ; position hold mode
        cmp     pos_err,#02
        jh      calc_out         ; if position error < 2 then turn off power
        clr     tmp+2
        clr     boost
        BR      output

calc_out:
        mulub   tmp,max_hold,#255
        mulu    tmp,pos_err            ; Tmp = pos_err * max_hold
        cmp     pos_delta,zero
        jne     no_bst
        add     boost,#04              ; Boost is integral control
        add     tmp+2,boost            ; TMP+2 = MSB(pos_err*max_hold)
        br      ck_max
no_bst: clr     boost
ck_max: cmp     tmp+2,max_hold
        jnh     output
maxed:  ld      tmp+2,max_hold
output: ldb     pwm_pwr,tmp+2

chk_sanity:
        br      ld_pwr

ld_pwr:
        ldb     rpwr,pwm_pwr
        notb    rpwr
        jbs     pwm_dir,0,p2fwd

p2bkwd: DI
        andb    port2,#01111111B       ; clear P2.7
        ldb     pwm_control,rpwr
        EI
        br      pwrset
p2fwd:  DI
        orb     port2,#10000000B       ; set P2.7
        ldb     pwm_control,rpwr
        EI
```

270061-57

Listing 4-11b: Motor Control Power Algorithm

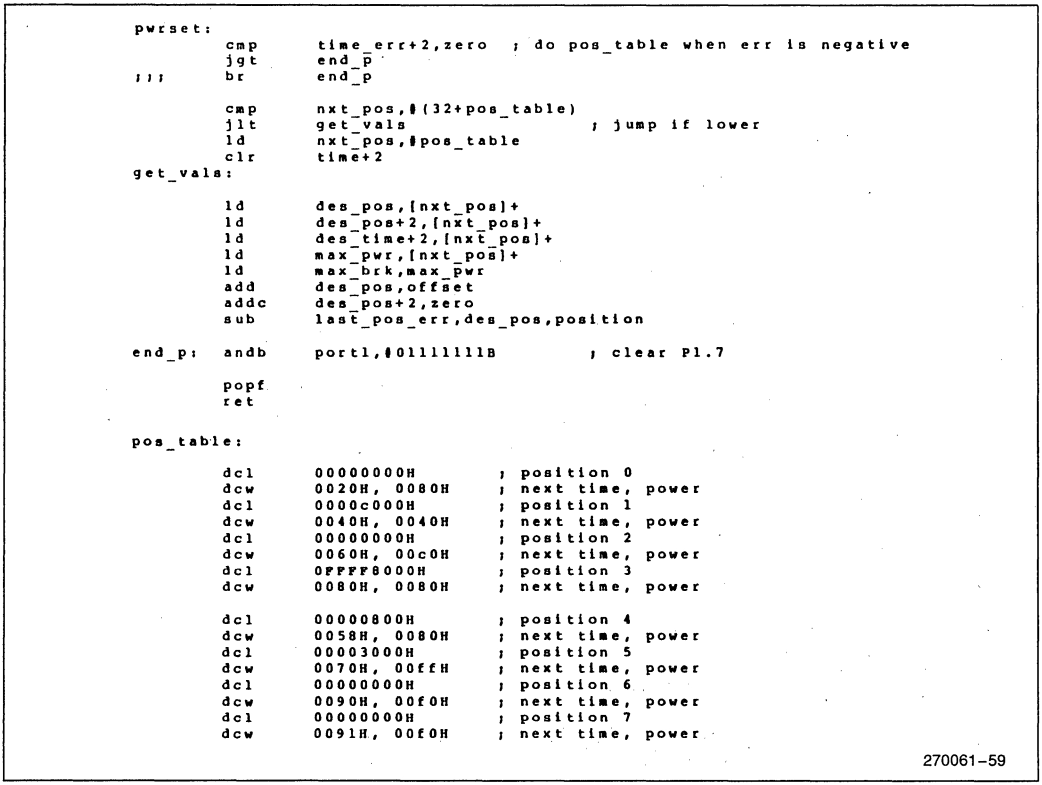


270061-58

Figure 4-7. Motor Control Modes

error does not get smaller. Once the error does get smaller, usually because the motor starts moving, BOOST is cleared.

A sanity check can be performed at this point to double check that the 8096 has proper control of the motor. In the example the worst that can happen is the proto-



```
pwrset:
          cmp      time_err+2,zero   ; do pos_table when err is negative
          jgt      end_p
;;;       br       end_p

          cmp      nxt_pos,#(32+pos_table)
          jlt      get_vals                    ; jump if lower
          ld       nxt_pos,#pos_table
          clr      time+2
get_vals:

          ld       des_pos,[nxt_pos]+
          ld       des_pos+2,[nxt_pos]+
          ld       des_time+2,[nxt_pos]+
          ld       max_pwr,[nxt_pos]+
          ld       max_brk,max_pwr
          add      des_pos,offset
          addc     des_pos+2,zero
          sub      last_pos_err,des_pos,position

end_p:    andb     portl,#01111111B            ; clear P1.7

          popf
          ret

pos_table:

          dcl      00000000H        ; position 0
          dcw      0020H, 0080H     ; next time, power
          dcl      0000c000H        ; position 1
          dcw      0040H, 0040H     ; next time, power
          dcl      00000000H        ; position 2
          dcw      0060H, 00c0H     ; next time, power
          dcl      0FFFF8000H       ; position 3
          dcw      0080H, 0080H     ; next time, power

          dcl      00000800H        ; position 4
          dcw      0058H, 0080H     ; next time, power
          dcl      00003000H        ; position 5
          dcw      0070H, 00ffH     ; next time, power
          dcl      00000000H        ; position 6
          dcw      0090H, 00f0H     ; next time, power
          dcl      00000000H        ; position 7
          dcw      0091H, 00f0H     ; next time, power
```

270061-59

Listing 4-12. Motor Control Next Position Lookup

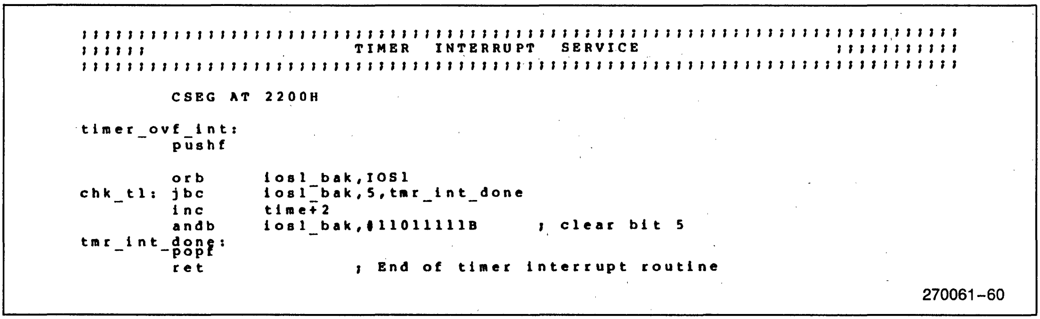
type will need to be reset, so the sanity check was not used. If one were desired, it could be as simple as checking a hardware generated direction indicator, or as complex as checking motor condition and other environmental factors.

After all checks have been made, the power value is loaded to the RPWR register using a software inversion to compensate for the hardware inversion. Direction is determined next and the power and direction are changed in adjacent instructions with interrupts disabled to prevent changing power without direction and vice versa.

To exercise the program logic the desired position is changed based on the time value using the code and lookup table shown in Listing 4-12.

The remaining sections of the program are relatively simple, but worth discussing briefly. The initialization routine initializes the I/O features and places several variables from ROM into RAM. Having these variables in RAM makes it easier to tweak the algorithm. Timer 1 is expanded into a 32-bit timer by the interrupt routine shown in Listing 4-13.

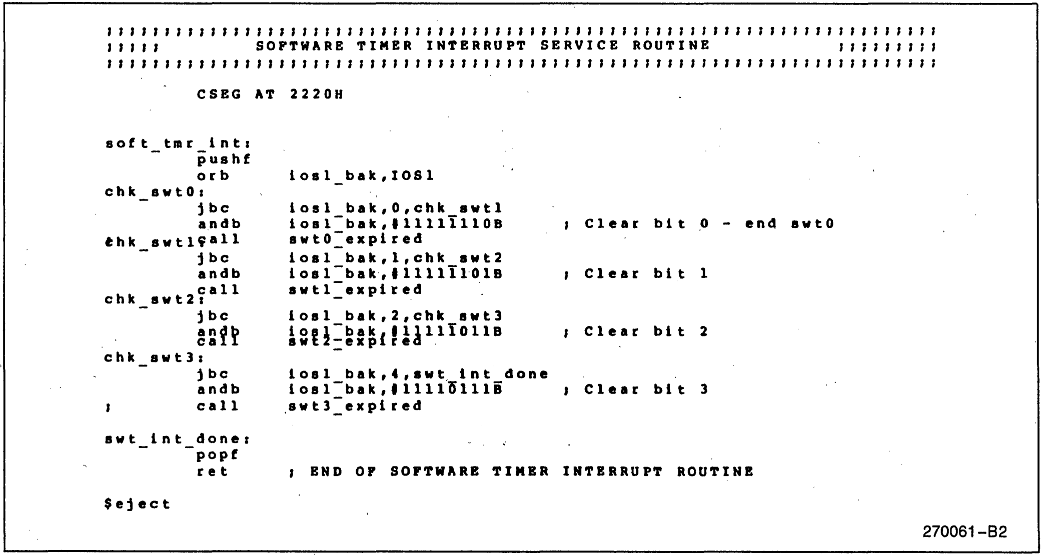
Software timer overhead is handled by the routine shown in Listing 4-14. In this routine the status of each timer bit is checked in a shadow register. If any of the timers have expired the appropriate routine is called.



```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;                          TIMER  INTERRUPT  SERVICE                ;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

          CSEG AT 2200H

timer_ovf_int:
          pushf

          orb       iosl_bak,IOS1
chk_tl:   jbc       iosl_bak,5,tmr_int_done
          inc       time+2
          andb      iosl_bak,#11011111B      ; clear bit 5
tmr_int_done:
          popf
          ret                 ; End of timer interrupt routine
```

270061-60

Listing 4-13. Motor Control Timer Interrupt Routine



```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;          SOFTWARE TIMER INTERRUPT SERVICE ROUTINE                   ;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

          CSEG AT 2220H


soft_tmr_int:
          pushf
          orb       iosl_bak,IOS1
chk_swt0:
          jbc       iosl_bak,0,chk_swtl
          andb      iosl_bak,#11111110B      ; Clear bit 0 - end swt0
chk_swtl: call      swt0_expired
          jbc       iosl_bak,1,chk_swt2
          andb      iosl_bak,#11111101B      ; Clear bit 1
          call      swtl_expired
chk_swt2:
          jbc       iosl_bak,2,chk_swt3
          andb      iosl_bak,#11111011B      ; Clear bit 2
          call      swt2_expired
chk_swt3:
          jbc       iosl_bak,4,swt_int_done
          andb      iosl_bak,#11110111B      ; Clear bit 3
;         call      swt3_expired

swt_int_done:
          popf
          ret       ; END OF SOFTWARE TIMER INTERRUPT ROUTINE

$eject
```

270061-B2

Listing 4-14. Motor Control Software Timer Interrupt Handler

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;                   SOFTWARE TIMER ROUTINE 2                 ;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        CSEG AT 2380H

swt2_expired:
        pushf
        ldb     hso_command,#3AH        ; set swt_2
        add     hso_time,timer1,swt2_dly

        orb     port1,#00000100B        ; set port 1.2
        cmp     out_ptr,#7ffH
        bnh     pulsing
        ld      out_ptr,#1f0H

pulsing:
        jbc     tr_col,0,swt2_done

        st      position+2,[out_ptr]+   ;  position high, position low
        st      position,[out_ptr]+

        st      direct,[out_ptr]+
        st      pwm_pwr,[out_ptr]+

                                        ; store 8 bytes externally

swt2_done:
        sub     tmp1,timer1,last1_time
        cmp     tmp1,#1800H
        jnh     swt2_ret         ; keep (time_last4_time)<7000H

        add     last1_time,#1000H
swt2_ret:
        andb    port1,#11111011B        ; clear port1.2
        popf
        ret
```

270061-61

Listing 4-15: Motor Control Software Timer 2 Routine

The last routine, shown in Listing 4-15, is the Software Timer 2 routine which outputs some variables to external RAM. It also keeps LAST1__Time within 1800H of Timer1 to prevent overflows from occurring when the Mode 0 and Mode 1 software check this variable.

A complete listing of the program as it is used in our lab can be found in Appendix D. For a given motor or encoder it will probably be necessary to change some of the time constants on the first page of the listing. With the motor used in our experimentation, pulses are missed from time to time when direction changes quickly. If the motor were not as fast to turn around or the encoder were mounted better these problems should disappear. The missing pulses occur when switching from Mode 1 to Mode 0, other than that no anomalies were found in the lab.

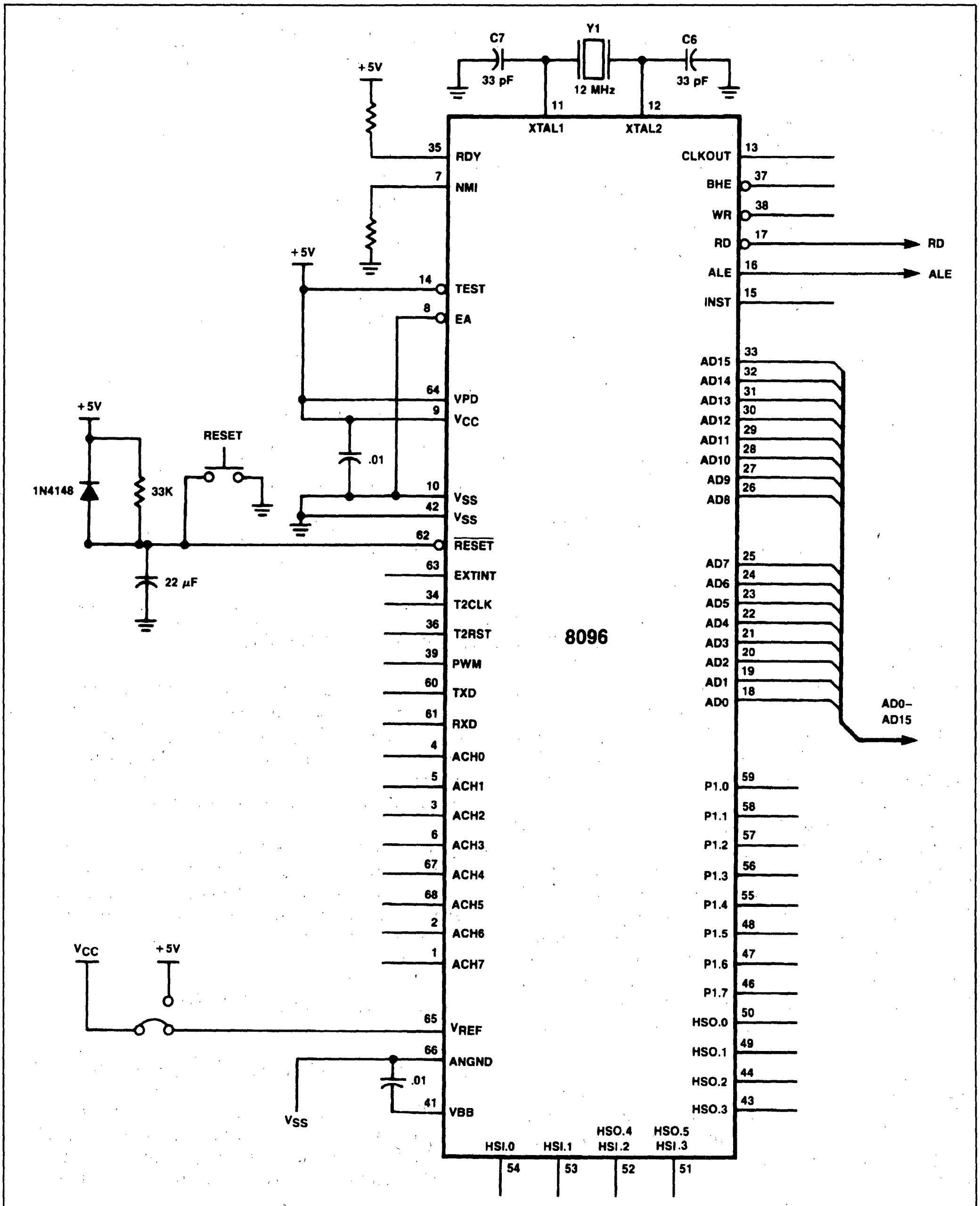
Prior to the version of code just discussed, several attempts were made, one of which could be used under certain constraints. It is possible to use only modes 2 and 0 to monitor the encoder, provided the encoder always operates smoothly and provides at least 200 microseconds between the last several edges of Phase A before reversing. This idea was originally tried because the motor was not characterized thoroughly at first, and caused problems because of the motors tendency to stop suddenly when its speed was low.

If an encoder has a lower line count and therefore more time between output pulses the two mode solution can be used. The software for the two mode version can be easily extracted form the three mode version, so it will not be presented.

5.0 HARDWARE EXAMPLE

5.1. EPROM Only Minimum System

The diagram in Figure 5-1 illustrates how to connect an 8096 in a minimum configuration system. Either 2764s or 27128s can be used in the system. Note that the lower EPROM contains the even bytes while the upper



270061-62

one contains the odd bytes, and the addressing is not fully decoded. This means that the addressing on a 2764 will be such that the lower 4K of each EPROM is mapped at 0000H and 4000H while the upper 4K is mapped at 2000H. If the program being loaded is 16 Kbytes long the first half is loaded into the second half of the 2764s and vice versa. A similar situation exists when using 27128s.

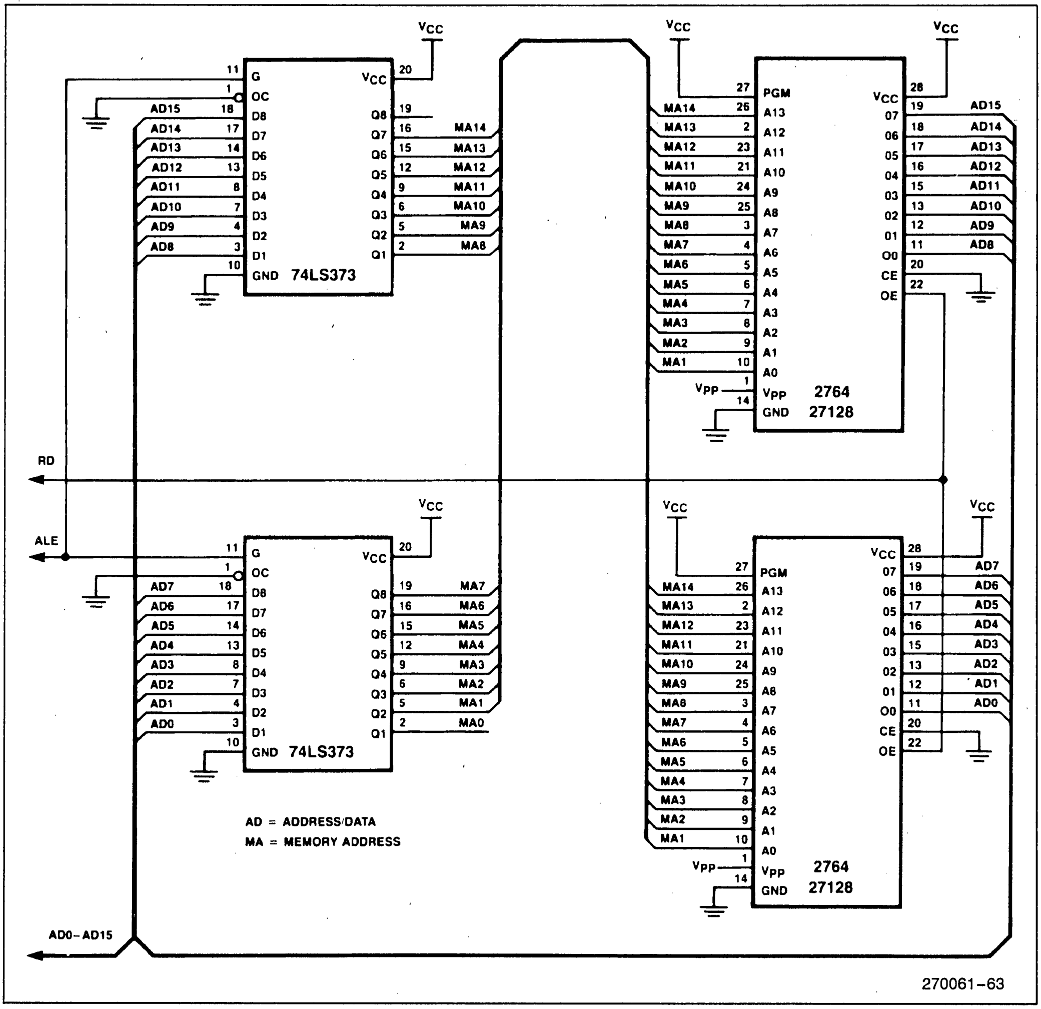


Figure 5-1 (2 of 2).

This circuit will allow most of the software presented in this ap-note to be run. In a system designed for prototyping in the lab it may be desirable to buffer the I/O ports to reduce the risk of burning out the chip during experimentation. One may also want to enhance the system by providing RC filters on the A to D inputs, a precision VREF power supply, and additional RAM.

5.2. Port Reconstruction

If it is desired to fully emulate a 8396 then I/O ports 3 and 4 must be reconstructed. It is easiest to do this if the usage of the lines can be restricted to inputs or outputs on a port by port rather than line by line basis. The ports are reconstructed by using standard memory-mapped I/O techniques, (i.e., address decoders and latches), at the appropriate addresses. If no external RAM is being used in the system then the address decoding can be partial, resulting in less complex logic.

The reconstructed I/O ports will work with the same code as the on chip ports. The only difference will be the propagation delay in the external circuitry.

6.0 CONCLUSION

An overview of the MCS-96 family has been presented along with several simple examples and a few more complex ones. The source code for all of these programs are available in the Insite Users Library using order code AE-16. Additional information on the 8096 can be found in the Microcontroller Handbook and it is recommended that this book be in your possession before attempting any work with the MCS-96 family of products. Your local Intel sales office can assist you in getting more information on the 8096 and its hardware and software development tools.

7.0 BIBLOGRAPHY

1. MSC-96 Macro Assembler User's Guide, Intel Corporation, 1983.

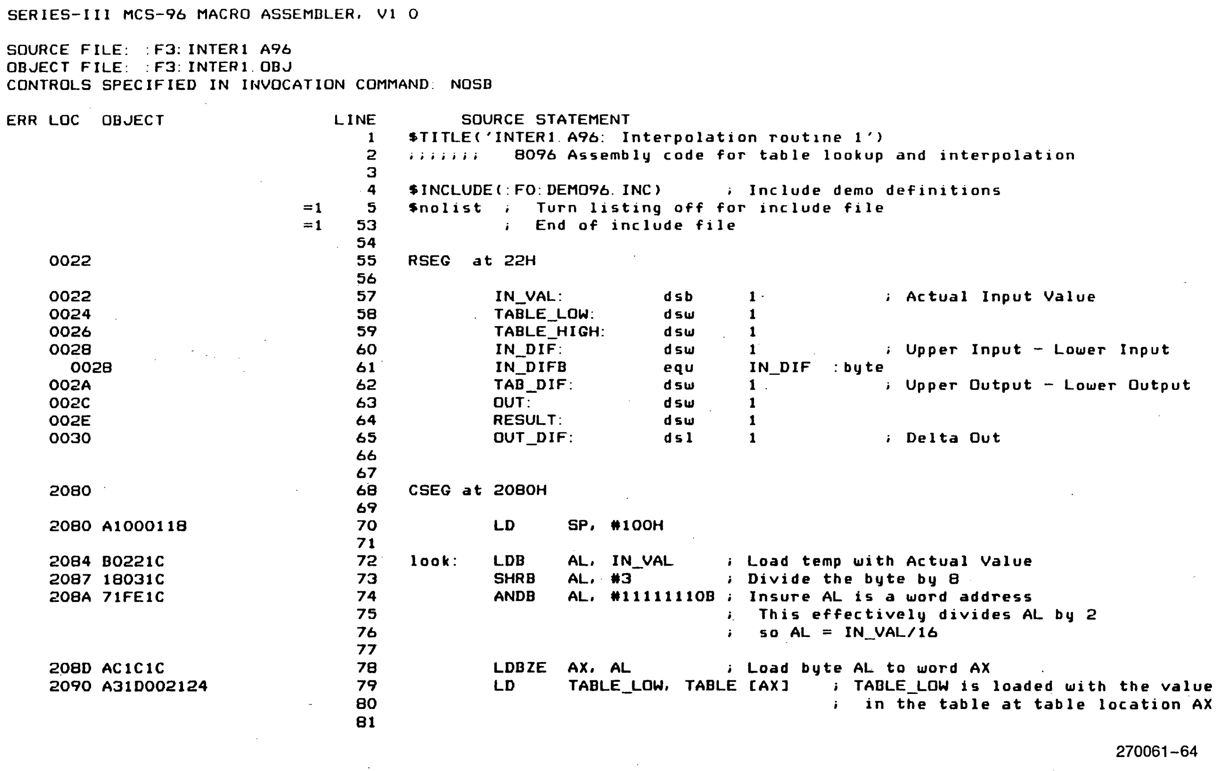
   Order number 122048-001.
2. Microcontroller Handbook (1985), Intel Corporation, 1984.

   Order number 210918-002.
3. MSC-96 Utilities User's Guide, Intel Corporation, 1983.

   Order number 122049-001.
4. PL/M-96 User's Guide, Intel Corporation, 1983.

   Order number 122134-001.

APPENDIX A
BASIC SOFTWARE EXAMPLES



```
SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE: :F3:INTER1 A96
OBJECT FILE: :F3:INTER1.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                    1   $TITLE('INTER1.A96: Interpolation routine 1')
                                    2   ;;;;;;;    8096 Assembly code for table lookup and interpolation
                                    3
                                    4   $INCLUDE(:FO:DEMO96.INC)       ; Include demo definitions
                              =1    5   $nolist  ;  Turn listing off for include file
                              =1   53            ;  End of include file
                                   54
    0022                           55   RSEG  at 22H
                                   56
    0022                           57           IN_VAL:          dsb     1             ; Actual Input Value
    0024                           58           TABLE_LOW:       dsw     1
    0026                           59           TABLE_HIGH:      dsw     1
    0028                           60           IN_DIF:          dsw     1             ; Upper Input - Lower Input
      0028                         61           IN_DIFB          equ     IN_DIF  :byte
    002A                           62           TAB_DIF:         dsw     1             ; Upper Output - Lower Output
    002C                           63           OUT:             dsw     1
    002E                           64           RESULT:          dsw     1
    0030                           65           OUT_DIF:         dsl     1             ; Delta Out
                                   66
                                   67
    2080                           68   CSEG at 2080H
                                   69
    2080 A1000118                  70           LD      SP, #100H
                                   71
    2084 B0221C                    72   look:   LDB     AL, IN_VAL      ; Load temp with Actual Value
    2087 18031C                    73           SHRB    AL, #3          ; Divide the byte by 8
    208A 71FE1C                    74           ANDB    AL, #11111110B ; Insure AL is a word address
                                   75                                   ;  This effectively divides AL by 2
                                   76                                   ;  so AL = IN_VAL/16
                                   77
    208D AC1C1C                    78           LDBZE   AX, AL          ; Load byte AL to word AX
    2090 A31D002124                79           LD      TABLE_LOW, TABLE [AX]     ; TABLE_LOW is loaded with the value
                                   80                                             ;  in the table at table location AX
                                   81
```

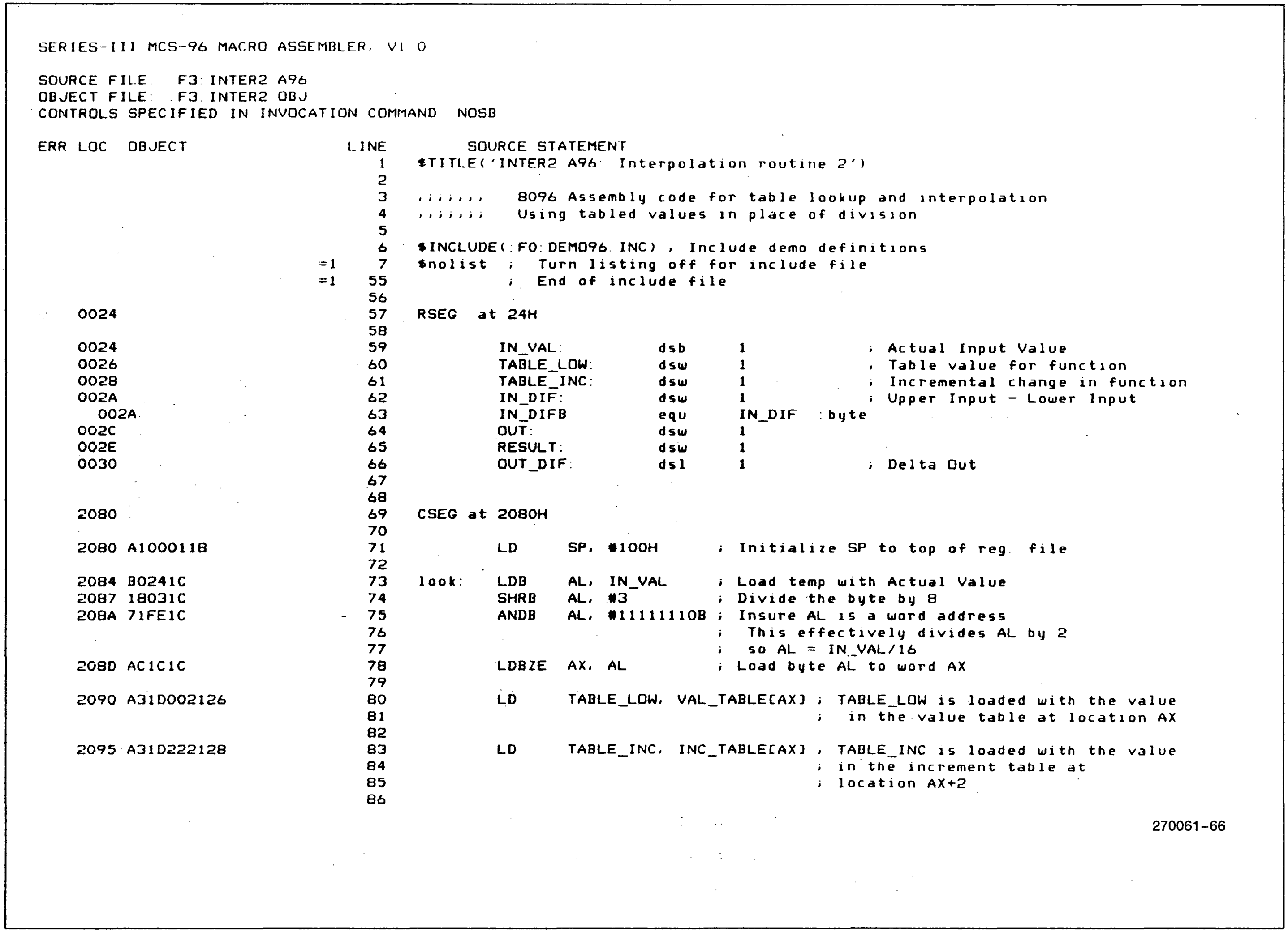
270061-64

A.1. Table Lookup 1

```
2095 A31D022126          82         LD     TABLE_HIGH, (TABLE+2)[AX]  ; TABLE_HIGH is loaded with the
                         83                                           ; value in the table at table
                         84                                           ; location AX+2
                         85                                           ; (The next value in the table)
                         86
209A 4824262A            87         SUB     TAB_DIF, TABLE_HIGH, TABLE_LOW
                         88                                           ; TAB_DIF=TABLE_HIGH-TABLE_LOW
                         89
209E 510F2228            90         ANDB    IN_DIFB, IN_VAL, #0FH     ; IN_DIFB=least significant 4 bits
                         91                                           ;  of IN_VAL
20A2 AC2828              92         LDBZE   IN_DIF, IN_DIFB           ; Load byte IN_DIFB to word IN_DIF
                         93
20A5 FE4C2A2830          94         MUL     OUT_DIF, IN_DIF, TAB_DIF
                         95                                           ; Output_difference =
                         96                                           ;  Input_difference*Table_difference
20AA 0E0430              97         SHRAL   OUT_DIF, #4               ; Divide by 16 (2**4)
                         98
20AD 4424302C            99         ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                        100                                           ;  generated with truncated IN_VAL
                        101                                           ;  as input
20B1 0A042C             102         SHRA    OUT, #4                   ; Round to 12-bit answer
20B4 A4002C             103         ADDC    OUT, zero                 ; Round up if Carry = 1
                        104
20B7 C02E2C             105 no_inc: ST      OUT, RESULT               ; Store OUT to RESULT
                        106
20BA 27C8               107         BR      look                      ; Branch to "look:"
                        108
                        109
2100                    110 cseg    AT 2100H
                        111
2100 000000200034004C   112 table:  DCW     0000H, 2000H, 3400H, 4C00H  ; A random function
2108 005D006A00720078   113         DCW     5D00H, 6A00H, 7200H, 7800H
2110 007B007D0076006D   114         DCW     7B00H, 7D00H, 7600H, 6D00H
2118 005D004B00340022   115         DCW     5D00H, 4B00H, 3400H, 2200H
2120 0010               116         DCW     1000H
                        117
2122                    118 END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-65

A.1. Table Lookup 1 (Continued)



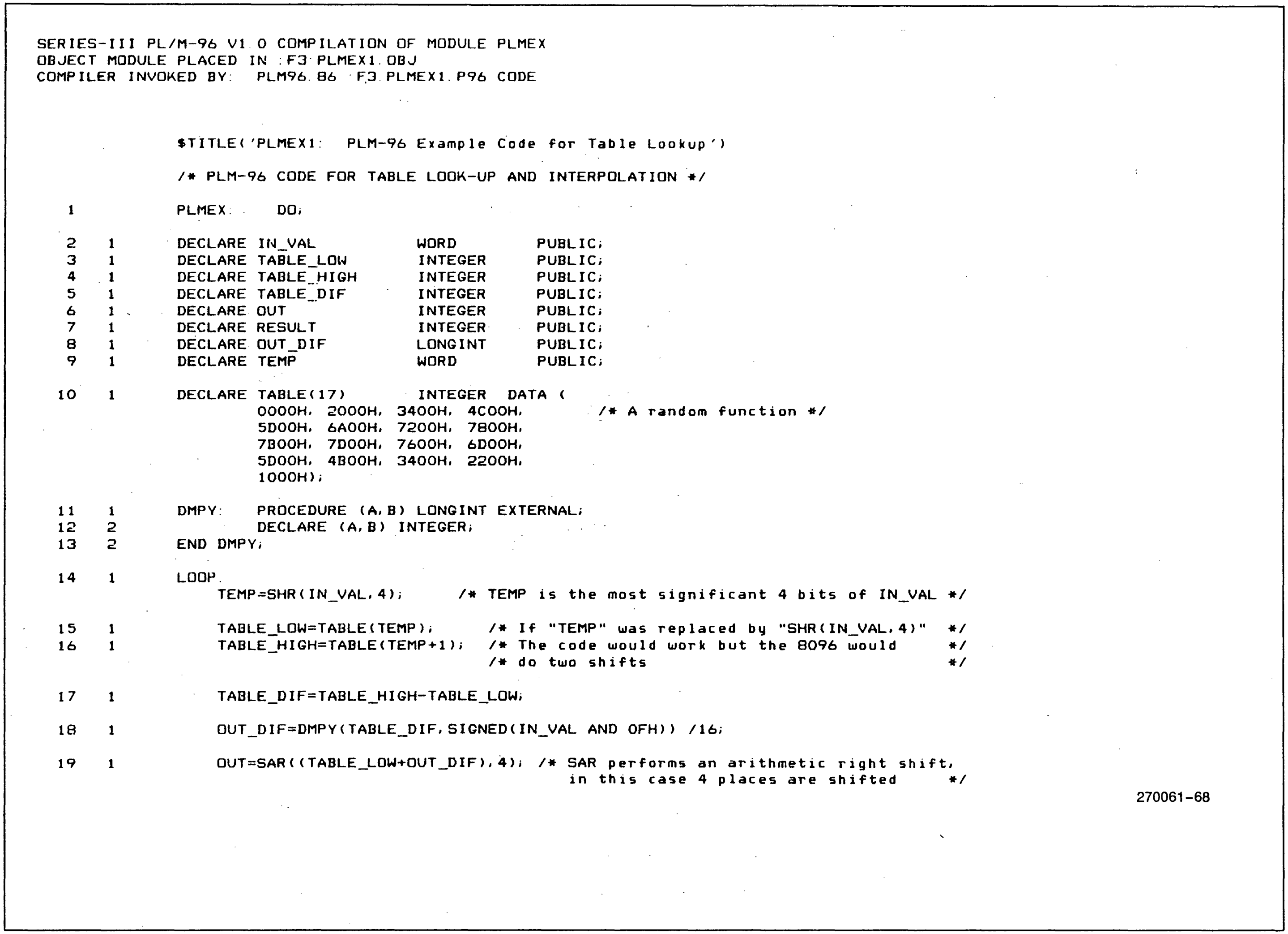
```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F3:INTER2.A96
OBJECT FILE: F3:INTER2.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                   LINE        SOURCE STATEMENT
                                    1   $TITLE('INTER2.A96  Interpolation routine 2')
                                    2
                                    3   ;;;;;;;    8096 Assembly code for table lookup and interpolation
                                    4   ;;;;;;;    Using tabled values in place of division
                                    5
                                    6   $INCLUDE(:F0:DEMO96.INC) ; Include demo definitions
                              =1    7   $nolist  ;  Turn listing off for include file
                              =1   55            ;  End of include file
                                   56
    0024                           57   RSEG  at 24H
                                   58
    0024                           59            IN_VAL:         dsb     1            ; Actual Input Value
    0026                           60            TABLE_LOW:      dsw     1            ; Table value for function
    0028                           61            TABLE_INC:      dsw     1            ; Incremental change in function
    002A                           62            IN_DIF:         dsw     1            ; Upper Input - Lower Input
      002A                         63            IN_DIFB         equ     IN_DIF  :byte
    002C                           64            OUT:            dsw     1
    002E                           65            RESULT:         dsw     1
    0030                           66            OUT_DIF:        dsl     1            ; Delta Out
                                   67
                                   68
    2080                           69   CSEG at 2080H
                                   70
    2080 A1000118                  71            LD      SP, #100H       ; Initialize SP to top of reg. file
                                   72
    2084 B0241C                    73   look:    LDB     AL, IN_VAL      ; Load temp with Actual Value
    2087 18031C                    74            SHRB    AL, #3          ; Divide the byte by 8
    208A 71FE1C                  - 75            ANDB    AL, #11111110B ; Insure AL is a word address
                                   76                                    ;  This effectively divides AL by 2
                                   77                                    ;  so AL = IN_VAL/16
    208D AC1C1C                    78            LDBZE   AX, AL          ; Load byte AL to word AX
                                   79
    2090 A31D002126                80            LD      TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value
                                   81                                             ;  in the value table at location AX
                                   82
    2095 A31D222128                83            LD      TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value
                                   84                                             ; in the increment table at
                                   85                                             ; location AX+2
                                   86
```

270061-66

A.2. Table Lookup 2 (Continued)

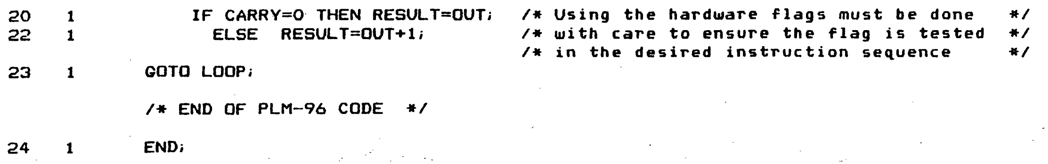
```
209A 510F242A             87          ANDB     IN_DIFB, IN_VAL, #0FH     ; IN_DIFB=least significant 4 bits
                          88                                             ;  of IN_VAL
209E AC2A2A               89          LDBZE    IN_DIF, IN_DIFB           ; Load byte IN_DIFB to word IN_DIF
                          90
20A1 FE4C282A30           91          MUL      OUT_DIF, IN_DIF, TABLE_INC
                          92                                             ; Output_difference =
                          93                                             ;  Input_difference*Incremental_change
                          94
20A6 4426302C             95          ADD      OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                          96                                             ;  generated with truncated IN_VAL
                          97                                             ;  as input
20AA 08042C               98          SHR      OUT, #4                   ; Round to 12-bit answer
20AD A4002C               99          ADDC     OUT, zero                 ; Round up if Carry = 1
                         100
20B0 C02E2C              101  no_inc: ST       OUT, RESULT               ; Store OUT to RESULT
20B3 27CF                102          BR       look                      ; Branch to "look:"
                         103
                         104
2100                     105  cseg    AT 2100H
                         106
2100                     107  val_table:
2100 000000200034004C    108          DCW      0000H, 2000H, 3400H, 4C00H  ; A random function
2108 005D006A00720078    109          DCW      5D00H, 6A00H, 7200H, 7800H
2110 007B007D0076006D    110          DCW      7B00H, 7D00H, 7600H, 6D00H
2118 005D004B00340022    111          DCW      5D00H, 4B00H, 3400H, 2200H
2120 0010                112          DCW      1000H
2122                     113  inc_table:
2122 0002400180011001    114          DCW       0200H,  0140H,  0180H,  0110H    ; Table of incremental
212A D000800060003000    115          DCW       00D0H,  0080H,  0060H,  0030H    ; differences
2132 200090FF70FF00FF    116          DCW      00020H, 0FF90H, 0FF70H, 0FF00H
213A E0FE90FEE0FEE0FE    117          DCW      0FEE0H, 0FE90H, 0FEE0H, 0FEE0H
                         118
2142                     119  END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-67



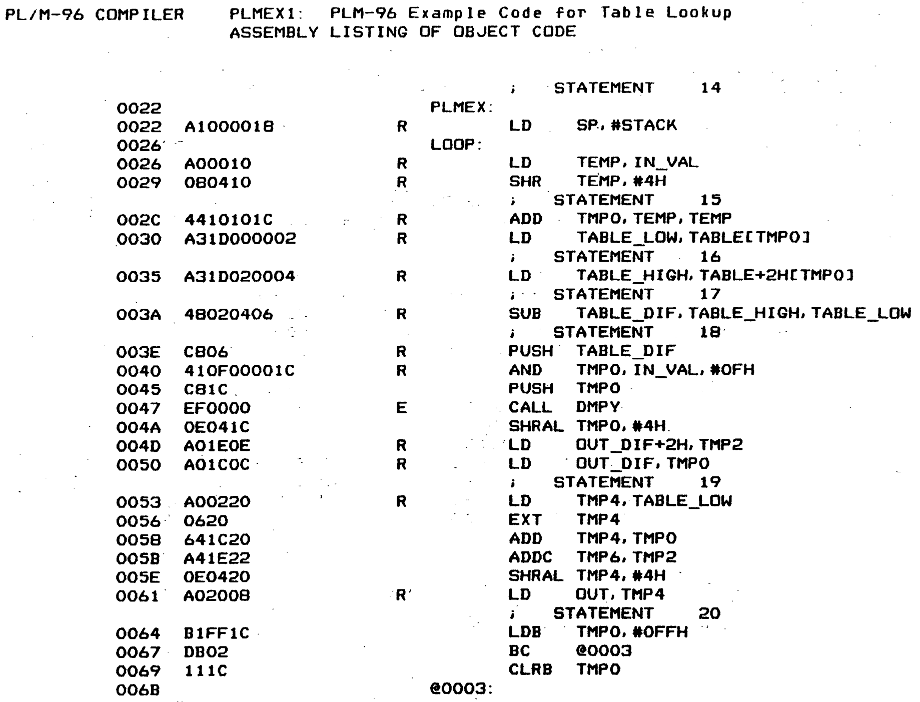
```
SERIES-III PL/M-96 V1.0 COMPILATION OF MODULE PLMEX
OBJECT MODULE PLACED IN :F3:PLMEX1.OBJ
COMPILER INVOKED BY:  PLM96.86 :F3:PLMEX1.P96 CODE


                $TITLE('PLMEX1:  PLM-96 Example Code for Table Lookup')

                /* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION */

   1            PLMEX:    DO;

   2    1       DECLARE IN_VAL          WORD        PUBLIC;
   3    1       DECLARE TABLE_LOW       INTEGER     PUBLIC;
   4    1       DECLARE TABLE_HIGH      INTEGER     PUBLIC;
   5    1       DECLARE TABLE_DIF       INTEGER     PUBLIC;
   6    1       DECLARE OUT             INTEGER     PUBLIC;
   7    1       DECLARE RESULT          INTEGER     PUBLIC;
   8    1       DECLARE OUT_DIF         LONGINT     PUBLIC;
   9    1       DECLARE TEMP            WORD        PUBLIC;

  10    1       DECLARE TABLE(17)       INTEGER   DATA (
                        0000H, 2000H, 3400H, 4C00H,       /* A random function */
                        5D00H, 6A00H, 7200H, 7800H,
                        7B00H, 7D00H, 7600H, 6D00H,
                        5D00H, 4B00H, 3400H, 2200H,
                        1000H);

  11    1       DMPY:   PROCEDURE (A,B) LONGINT EXTERNAL;
  12    2               DECLARE (A,B) INTEGER;
  13    2       END DMPY;

  14    1       LOOP:
                    TEMP=SHR(IN_VAL,4);      /* TEMP is the most significant 4 bits of IN_VAL */

  15    1           TABLE_LOW=TABLE(TEMP);       /* If "TEMP" was replaced by "SHR(IN_VAL,4)"  */
  16    1           TABLE_HIGH=TABLE(TEMP+1);    /* The code would work but the 8096 would     */
                                                 /* do two shifts                              */

  17    1           TABLE_DIF=TABLE_HIGH-TABLE_LOW;

  18    1           OUT_DIF=DMPY(TABLE_DIF,SIGNED(IN_VAL AND 0FH)) /16;

  19    1           OUT=SAR((TABLE_LOW+OUT_DIF),4); /* SAR performs an arithmetic right shift,
                                                        in this case 4 places are shifted    */
```

270061-68



```
20    1           IF CARRY=0 THEN RESULT=OUT;   /* Using the hardware flags must be done   */
22    1             ELSE  RESULT=OUT+1;         /* with care to ensure the flag is tested  */
                                                /* in the desired instruction sequence     */
23    1       GOTO LOOP;

              /* END OF PLM-96 CODE  */

24    1       END;
```

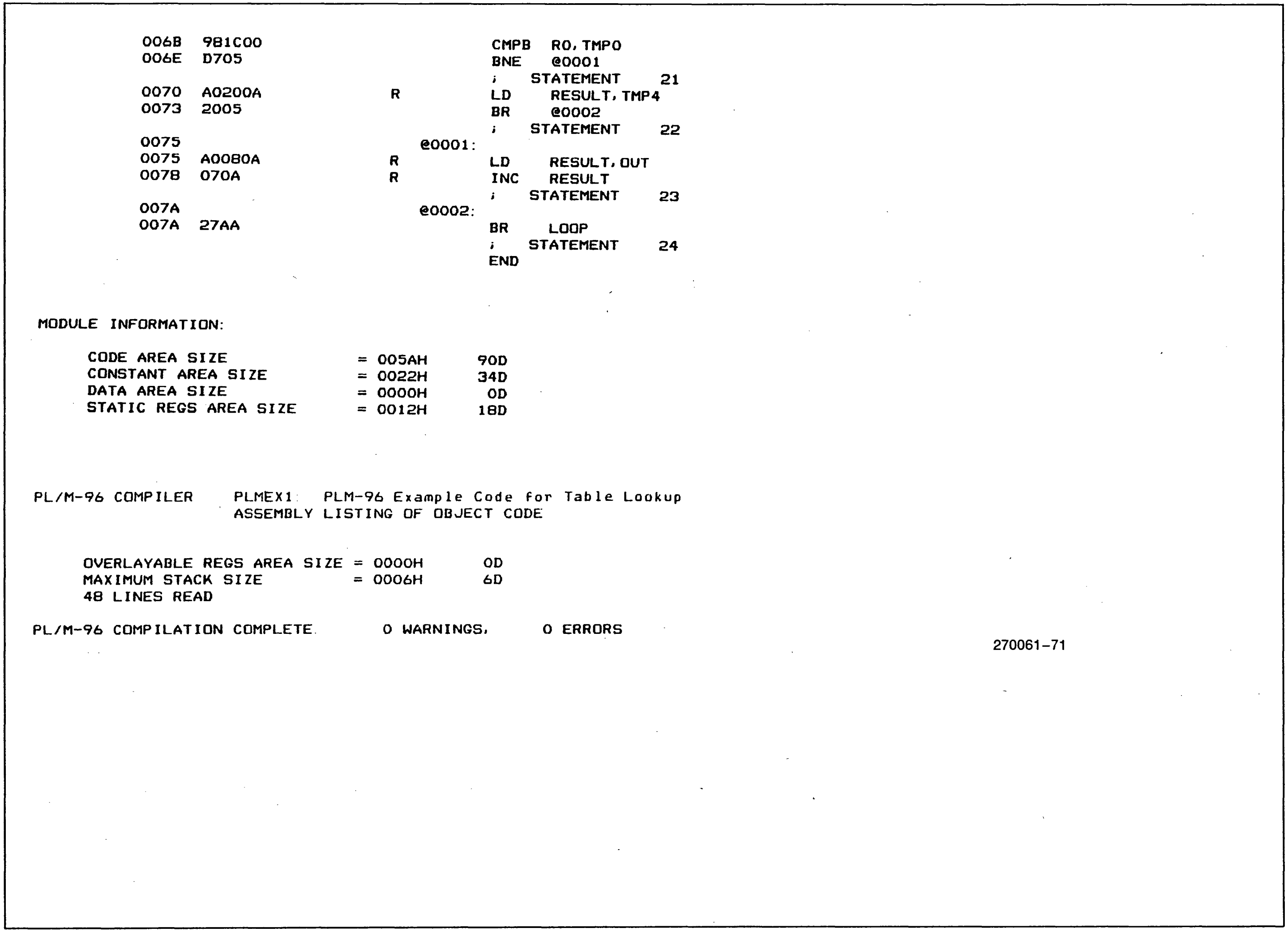
270061-69



```
PL/M-96 COMPILER     PLMEX1:  PLM-96 Example Code for Table Lookup
                     ASSEMBLY LISTING OF OBJECT CODE

                                                ;   STATEMENT    14
          0022                          PLMEX:
          0022   A1000018          R            LD     SP, #STACK
          0026                          LOOP:
          0026   A00010            R            LD     TEMP, IN_VAL
          0029   080410            R            SHR    TEMP, #4H
                                                ;   STATEMENT    15
          002C   4410101C          R            ADD    TMP0, TEMP, TEMP
          0030   A31D000002        R            LD     TABLE_LOW, TABLE[TMP0]
                                                ;   STATEMENT    16
          0035   A31D020004        R            LD     TABLE_HIGH, TABLE+2H[TMP0]
                                                ;   STATEMENT    17
          003A   48020406          R            SUB    TABLE_DIF, TABLE_HIGH, TABLE_LOW
                                                ;   STATEMENT    18
          003E   C806              R            PUSH   TABLE_DIF
          0040   410F00001C        R            AND    TMP0, IN_VAL, #0FH
          0045   C81C                           PUSH   TMP0
          0047   EF0000            E            CALL   DMPY
          004A   0E041C                         SHRAL  TMP0, #4H
          004D   A01E0E            R            LD     OUT_DIF+2H, TMP2
          0050   A01C0C            R            LD     OUT_DIF, TMP0
                                                ;   STATEMENT    19
          0053   A00220            R            LD     TMP4, TABLE_LOW
          0056   0620                           EXT    TMP4
          0058   641C20                         ADD    TMP4, TMP0
          005B   A41E22                         ADDC   TMP6, TMP2
          005E   0E0420                         SHRAL  TMP4, #4H
          0061   A02008            R            LD     OUT, TMP4
                                                ;   STATEMENT    20
          0064   B1FF1C                         LDB    TMP0, #0FFH
          0067   DB02                           BC     @0003
          0069   111C                           CLRB   TMP0
          006B                          @0003:
```

270061-70

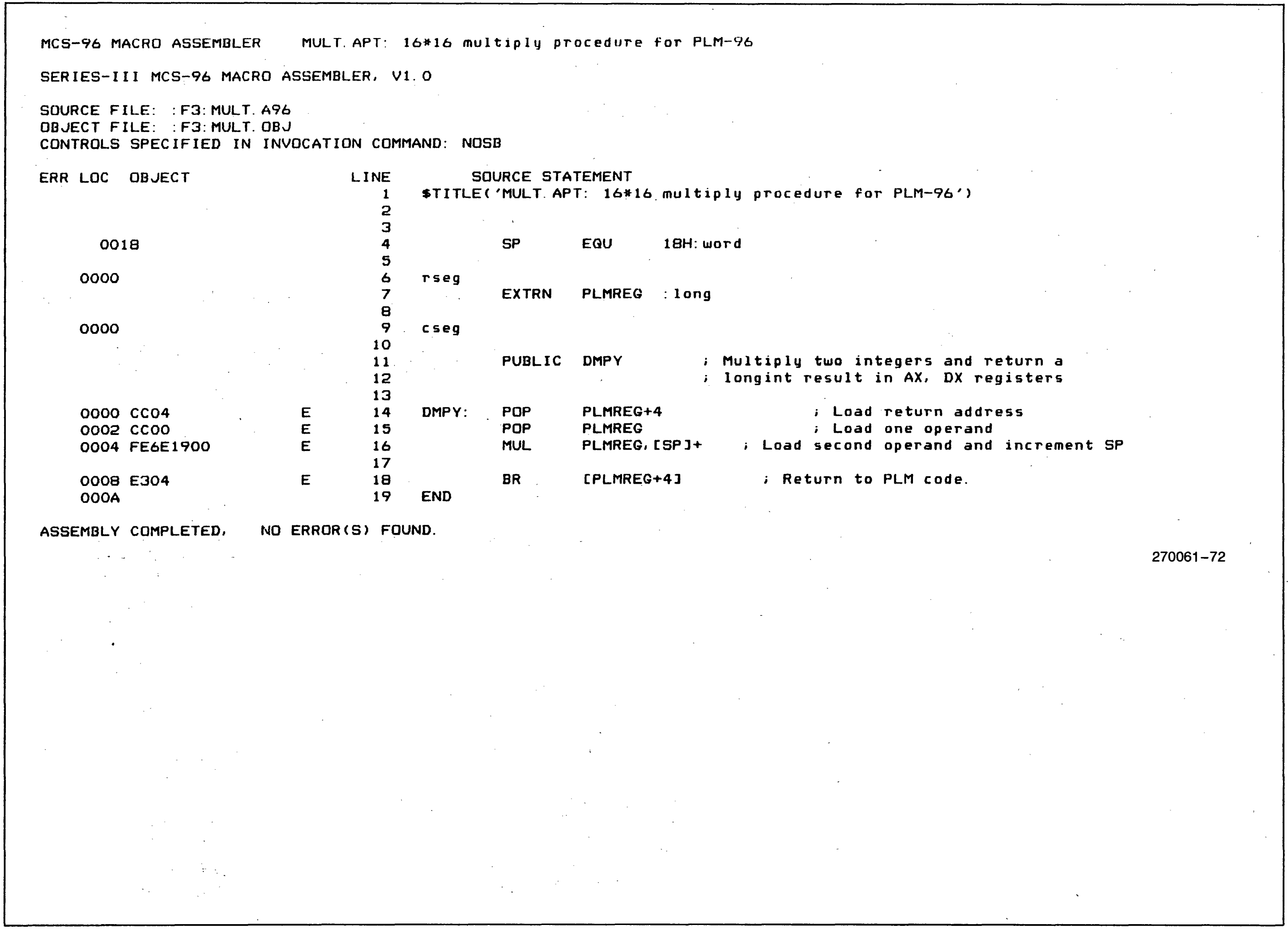
A.3. PLM-96 Code with Expansion (Continued)



```
        006B  981C00                       CMPB  R0,TMP0
        006E  D705                         BNE   @0001
                                           ;   STATEMENT     21
        0070  A0200A          R            LD    RESULT,TMP4
        0073  2005                         BR    @0002
                                           ;   STATEMENT     22
        0075                        @0001:
        0075  A0080A          R            LD    RESULT,OUT
        0078  070A            R            INC   RESULT
                                           ;   STATEMENT     23
        007A                        @0002:
        007A  27AA                         BR    LOOP
                                           ;   STATEMENT     24
                                           END


MODULE INFORMATION:

     CODE AREA SIZE              = 005AH      90D
     CONSTANT AREA SIZE          = 0022H      34D
     DATA AREA SIZE              = 0000H       0D
     STATIC REGS AREA SIZE       = 0012H      18D



PL/M-96 COMPILER     PLMEX1   PLM-96 Example Code for Table Lookup
                     ASSEMBLY LISTING OF OBJECT CODE


     OVERLAYABLE REGS AREA SIZE = 0000H       0D
     MAXIMUM STACK SIZE         = 0006H       6D
     48 LINES READ

PL/M-96 COMPILATION COMPLETE.        0 WARNINGS,      0 ERRORS
```

270061-71

A.3. PLM-96 Code with Expansion (Continued)

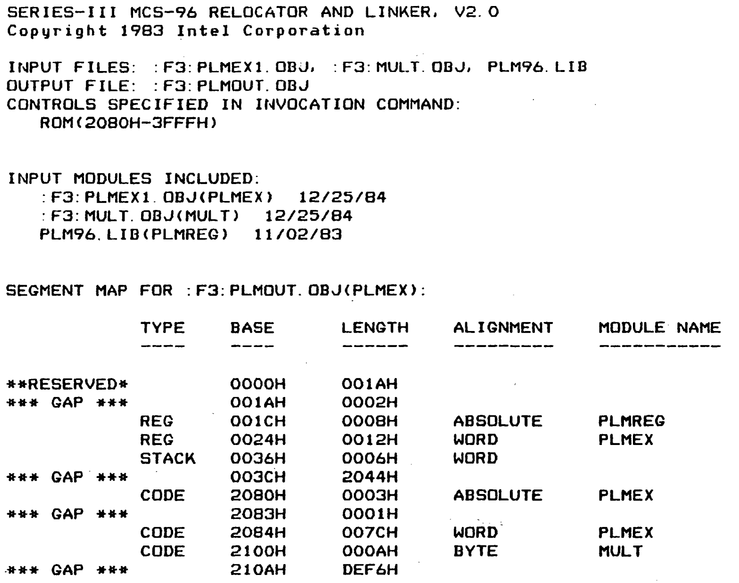


```
MCS-96 MACRO ASSEMBLER     MULT.APT: 16*16 multiply procedure for PLM-96

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:MULT.A96
OBJECT FILE: :F3:MULT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                   LINE        SOURCE STATEMENT
                                     1   $TITLE('MULT.APT: 16*16 multiply procedure for PLM-96')
                                     2
                                     3
      0018                           4           SP      EQU      18H:word
                                     5
    0000                             6   rseg
                                     7           EXTRN   PLMREG   :long
                                     8
    0000                             9   cseg
                                    10
                                    11           PUBLIC  DMPY          ; Multiply two integers and return a
                                    12                                 ; longint result in AX, DX registers
                                    13
    0000 CC04              E        14   DMPY:   POP     PLMREG+4                 ; Load return address
    0002 CC00              E        15           POP     PLMREG                   ; Load one operand
    0004 FE6E1900          E        16           MUL     PLMREG,[SP]+    ; Load second operand and increment SP
                                    17
    0008 E304              E        18           BR      [PLMREG+4]        ; Return to PLM code.
    000A                            19   END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-72

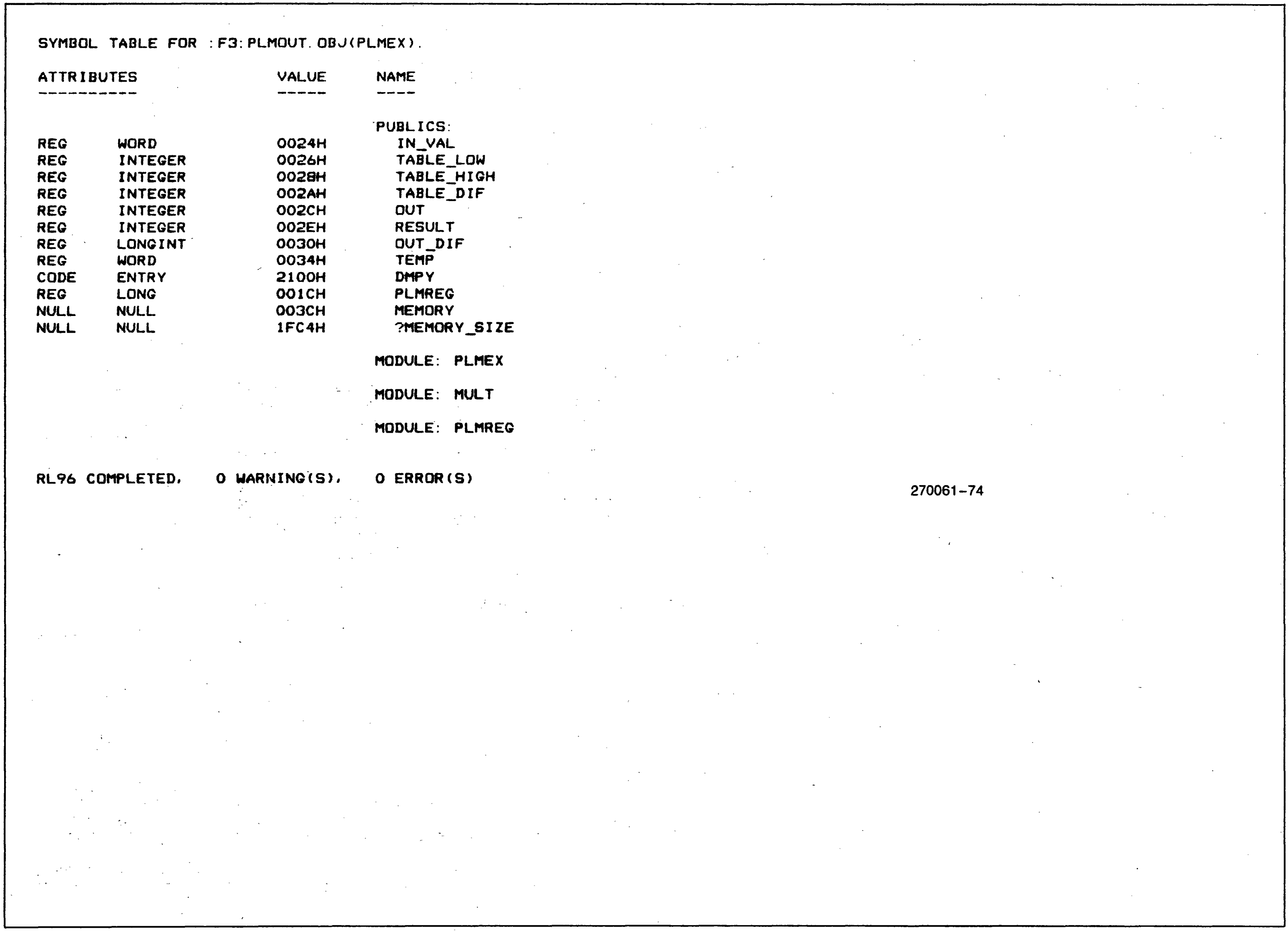
A.3. PLM-96 Code with Expansion (Continued)

A.3. PLM-96 Code with Expansion (Continued)



```
SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation

INPUT FILES: :F3:PLMEX1.OBJ, :F3:MULT.OBJ, PLM96.LIB
OUTPUT FILE: :F3:PLMOUT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:
   ROM(2080H-3FFFH)


INPUT MODULES INCLUDED:
   :F3:PLMEX1.OBJ(PLMEX)  12/25/84
   :F3:MULT.OBJ(MULT)  12/25/84
   PLM96.LIB(PLMREG)  11/02/83


SEGMENT MAP FOR :F3:PLMOUT.OBJ(PLMEX):

             TYPE     BASE      LENGTH     ALIGNMENT     MODULE NAME
             ----     ----      ------     ---------     -----------

**RESERVED*           0000H     001AH
*** GAP ***           001AH     0002H
             REG      001CH     0008H      ABSOLUTE      PLMREG
             REG      0024H     0012H      WORD          PLMEX
             STACK    0036H     0006H      WORD
*** GAP ***           003CH     2044H
             CODE     2080H     0003H      ABSOLUTE      PLMEX
*** GAP ***           2083H     0001H
             CODE     2084H     007CH      WORD          PLMEX
             CODE     2100H     000AH      BYTE          MULT
*** GAP ***           210AH     DEF6H
```

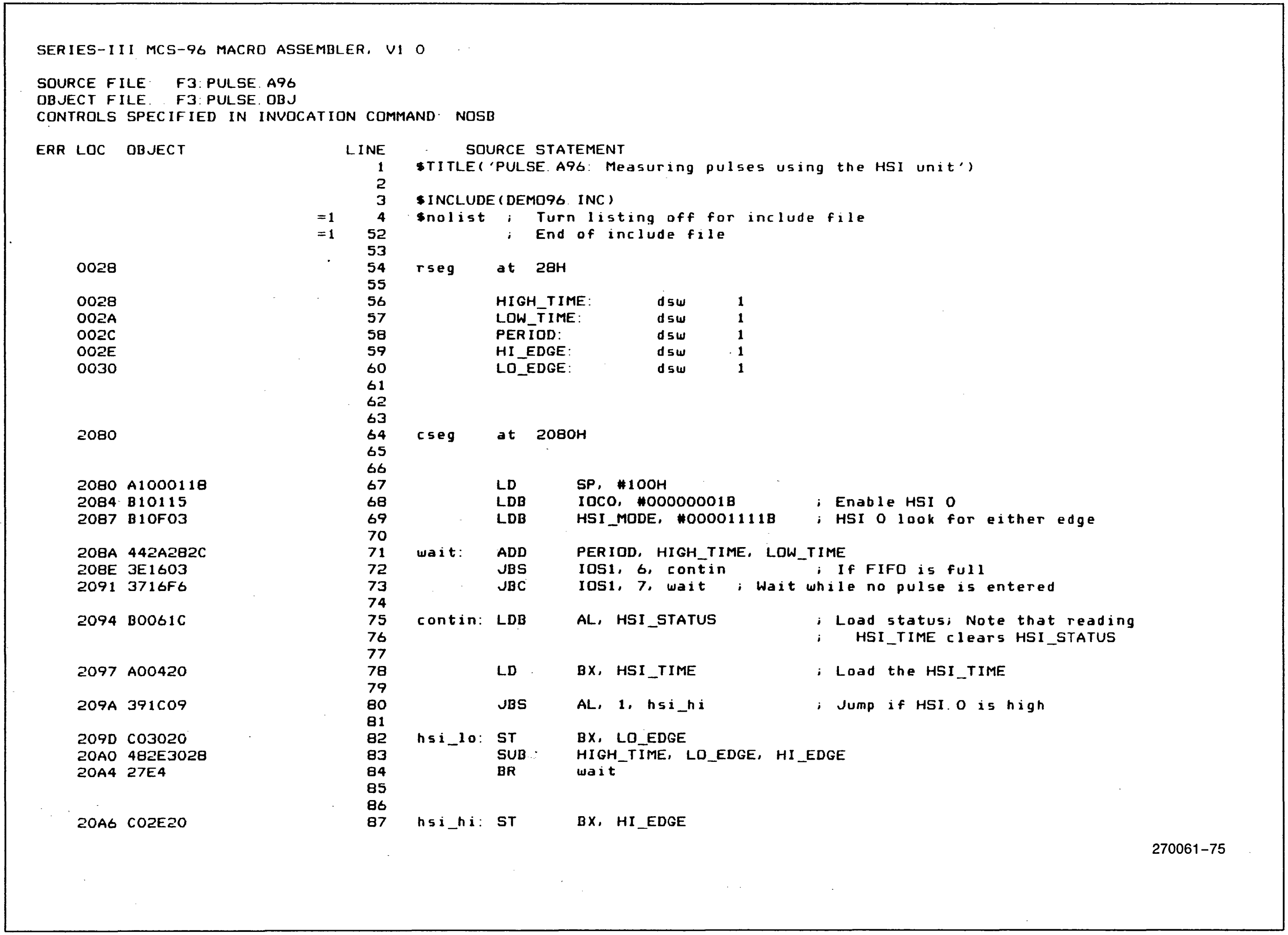
270061-73



```
SYMBOL TABLE FOR :F3:PLMOUT.OBJ(PLMEX).

ATTRIBUTES               VALUE      NAME
----------               -----      ----

                                    PUBLICS:
REG      WORD            0024H        IN_VAL
REG      INTEGER         0026H        TABLE_LOW
REG      INTEGER         0028H        TABLE_HIGH
REG      INTEGER         002AH        TABLE_DIF
REG      INTEGER         002CH        OUT
REG      INTEGER         002EH        RESULT
REG      LONGINT         0030H        OUT_DIF
REG      WORD            0034H        TEMP
CODE     ENTRY           2100H        DMPY
REG      LONG            001CH        PLMREG
NULL     NULL            003CH        MEMORY
NULL     NULL            1FC4H        ?MEMORY_SIZE

                                    MODULE:  PLMEX

                                    MODULE:  MULT

                                    MODULE:  PLMREG


RL96 COMPLETED,    0 WARNING(S),    0 ERROR(S)
```

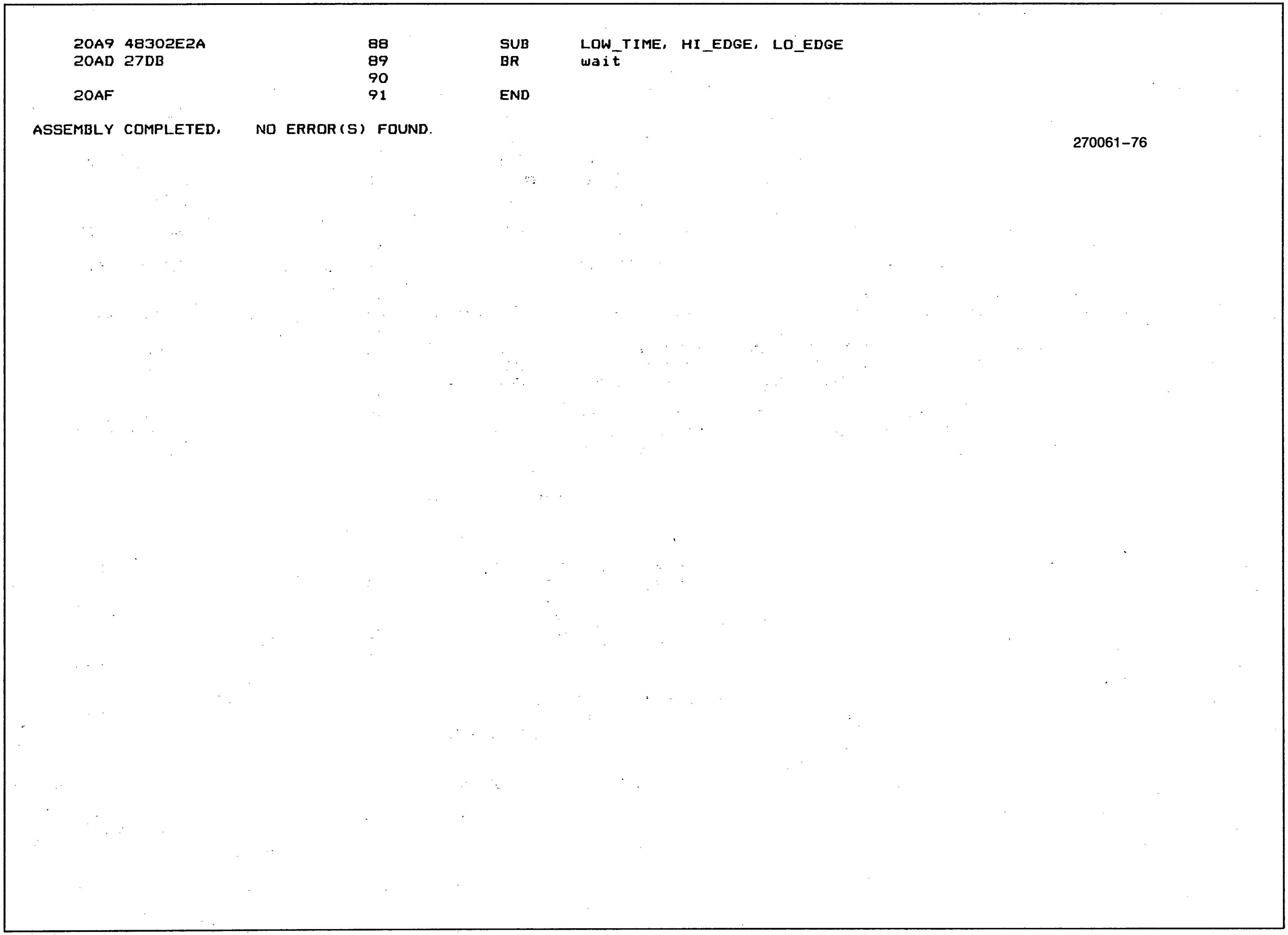
270061-74

A.3. PLM-96 Code with Expansion (Continued)



```
SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE   F3:PULSE.A96
OBJECT FILE   F3:PULSE.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND  NOSB

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                  1   $TITLE('PULSE.A96: Measuring pulses using the HSI unit')
                                  2
                                  3   $INCLUDE(DEMO96.INC)
                          =1      4   $nolist  ;  Turn listing off for include file
                          =1     52            ;  End of include file
                                 53
    0028                         54   rseg     at  28H
                                 55
    0028                         56            HIGH_TIME:      dsw     1
    002A                         57            LOW_TIME:       dsw     1
    002C                         58            PERIOD:         dsw     1
    002E                         59            HI_EDGE:        dsw     1
    0030                         60            LO_EDGE:        dsw     1
                                 61
                                 62
                                 63
    2080                         64   cseg     at  2080H
                                 65
                                 66
    2080 A1000118                67            LD      SP, #100H
    2084 B10115                  68            LDB     IOC0, #00000001B        ; Enable HSI 0
    2087 B10F03                  69            LDB     HSI_MODE, #00001111B    ; HSI 0 look for either edge
                                 70
    208A 442A282C                71   wait:    ADD     PERIOD, HIGH_TIME, LOW_TIME
    208E 3E1603                  72            JBS     IOS1, 6, contin         ; If FIFO is full
    2091 3716F6                  73            JBC     IOS1, 7, wait   ; Wait while no pulse is entered
                                 74
    2094 B0061C                  75   contin:  LDB     AL, HSI_STATUS          ; Load status; Note that reading
                                 76                                            ;   HSI_TIME clears HSI_STATUS
                                 77
    2097 A00420                  78            LD      BX, HSI_TIME            ; Load the HSI_TIME
                                 79
    209A 391C09                  80            JBS     AL, 1, hsi_hi           ; Jump if HSI.0 is high
                                 81
    209D C03020                  82   hsi_lo:  ST      BX, LO_EDGE
    20A0 482E3028                83            SUB     HIGH_TIME, LO_EDGE, HI_EDGE
    20A4 27E4                    84            BR      wait
                                 85
                                 86
    20A6 C02E20                  87   hsi_hi:  ST      BX, HI_EDGE
```

270061-75

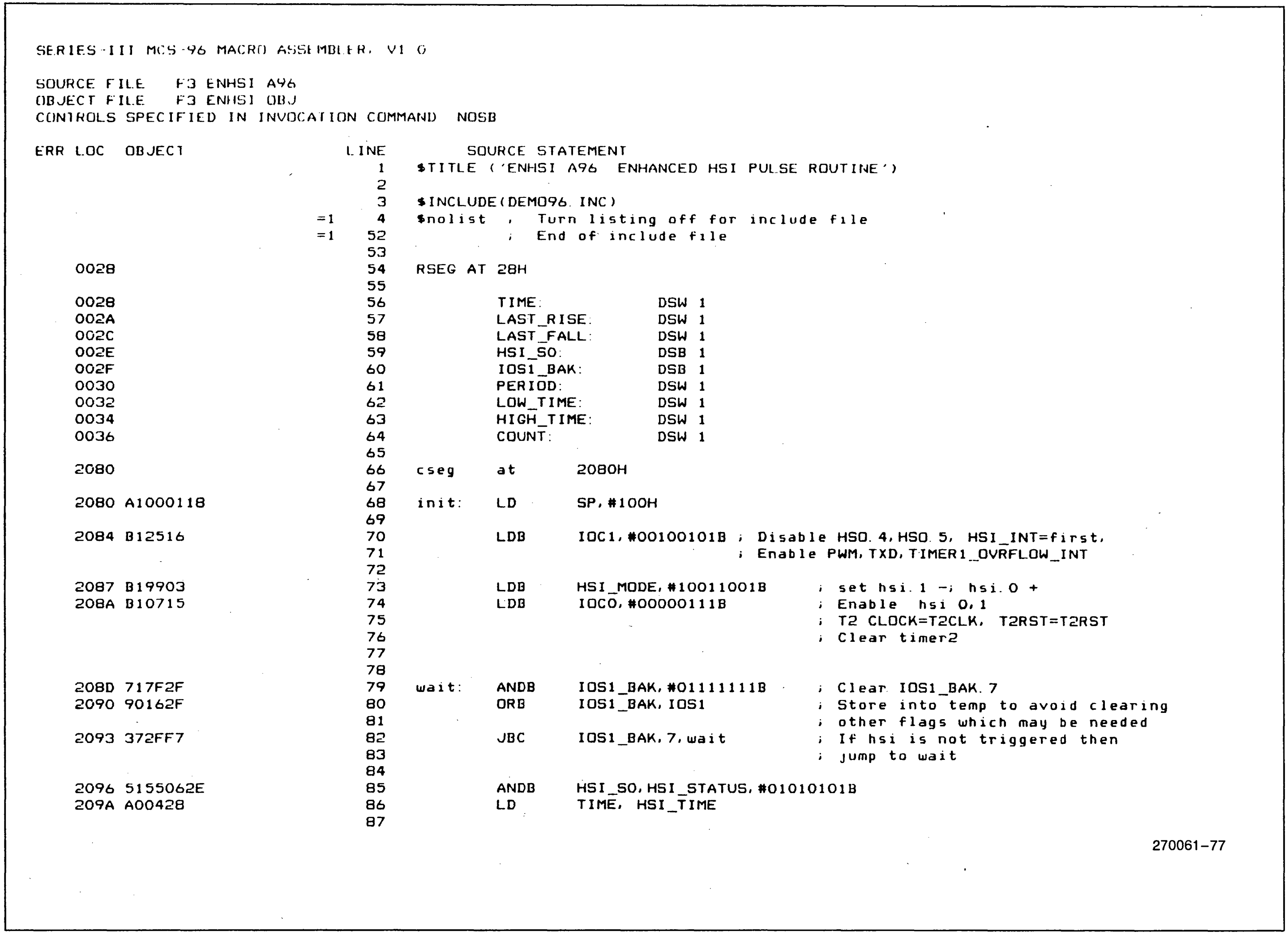


```
    20A9 48302E2A           88          SUB      LOW_TIME, HI_EDGE, LO_EDGE
    20AD 27DB               89          BR       wait
                            90
    20AF                    91          END

ASSEMBLY COMPLETED,    NO ERROR(S) FOUND.
```

270061-76

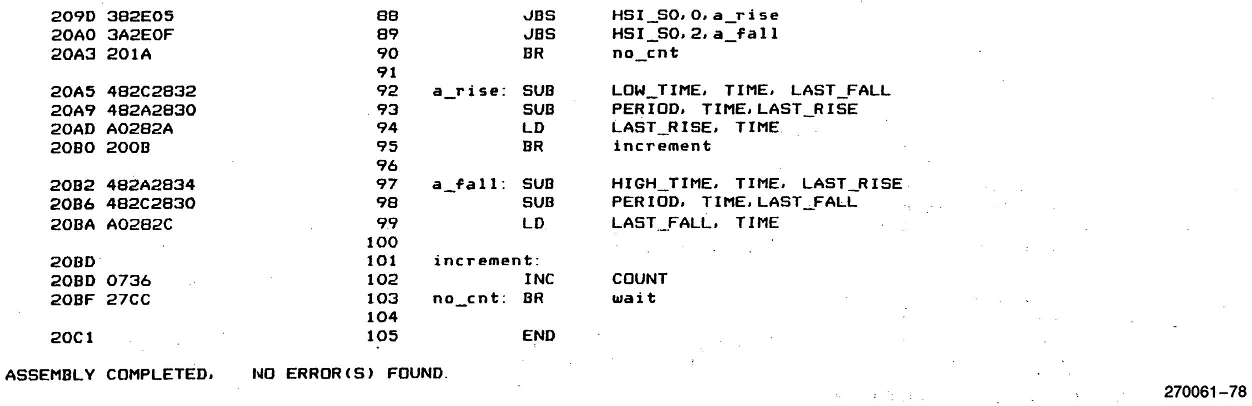
A.4. Pulse Measurement (Continued)

A.5. Enhanced Pulse Measurement



```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE   F3:ENHSI.A96
OBJECT FILE   F3:ENHSI.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND  NOSB

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                  1  $TITLE ('ENHSI.A96  ENHANCED HSI PULSE ROUTINE')
                                  2
                                  3  $INCLUDE(DEMO96.INC)
                            =1    4  $nolist  ;  Turn listing off for include file
                            =1   52          ;  End of include file
                                 53
    0028                         54  RSEG AT 28H
                                 55
    0028                         56          TIME:           DSW 1
    002A                         57          LAST_RISE:      DSW 1
    002C                         58          LAST_FALL:      DSW 1
    002E                         59          HSI_SO:         DSB 1
    002F                         60          IOS1_BAK:       DSB 1
    0030                         61          PERIOD:         DSW 1
    0032                         62          LOW_TIME:       DSW 1
    0034                         63          HIGH_TIME:      DSW 1
    0036                         64          COUNT:          DSW 1
                                 65
    2080                         66  cseg    at      2080H
                                 67
    2080 A1000118                68  init:   LD      SP,#100H
                                 69
    2084 B12516                  70          LDB     IOC1,#00100101B ; Disable HSO.4,HSO.5, HSI_INT=first,
                                 71                                  ; Enable PWM,TXD,TIMER1_OVRFLOW_INT
                                 72
    2087 B19903                  73          LDB     HSI_MODE,#10011001B      ; set hsi.1 -; hsi.0 +
    208A B10715                  74          LDB     IOC0,#00000111B          ; Enable  hsi 0,1
                                 75                                           ; T2 CLOCK=T2CLK, T2RST=T2RST
                                 76                                           ; Clear timer2
                                 77
                                 78
    208D 717F2F                  79  wait:   ANDB    IOS1_BAK,#01111111B      ; Clear IOS1_BAK.7
    2090 90162F                  80          ORB     IOS1_BAK,IOS1            ; Store into temp to avoid clearing
                                 81                                           ; other flags which may be needed
    2093 372FF7                  82          JBC     IOS1_BAK,7,wait          ; If hsi is not triggered then
                                 83                                           ; jump to wait
                                 84
    2096 5155062E                85          ANDB    HSI_SO,HSI_STATUS,#01010101B
    209A A00428                  86          LD      TIME, HSI_TIME
                                 87
```

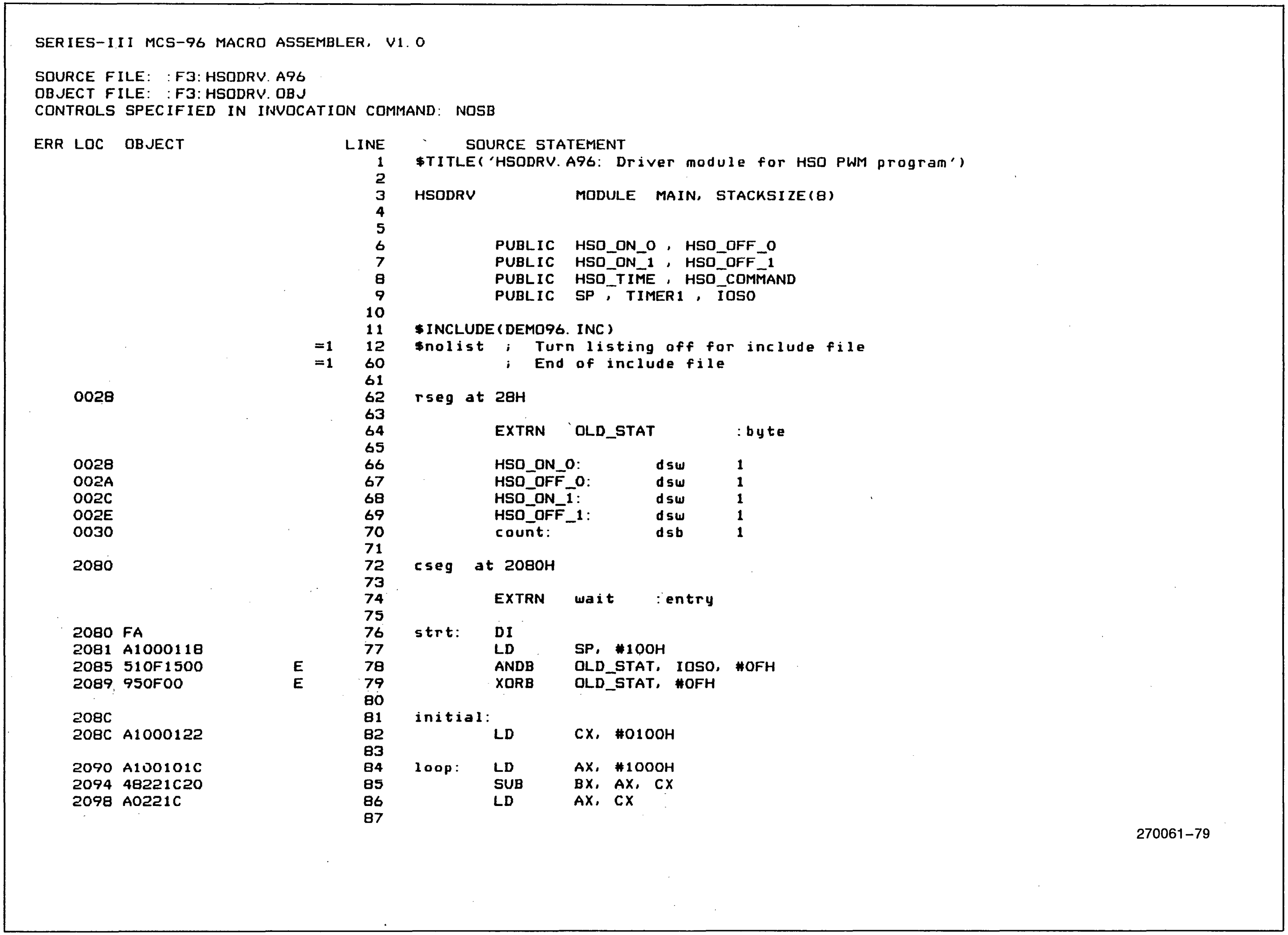
270061-77



```
209D 382E05                 JBS     HSI_SO,0,a_rise
20A0 3A2E0F                 JBS     HSI_SO,2,a_fall
20A3 201A                   BR      no_cnt

20A5 482C2832       a_rise: SUB     LOW_TIME, TIME, LAST_FALL
20A9 482A2830               SUB     PERIOD, TIME,LAST_RISE
20AD A0282A                 LD      LAST_RISE, TIME
20B0 200B                   BR      increment

20B2 482A2834       a_fall: SUB     HIGH_TIME, TIME, LAST_RISE
20B6 482C2830               SUB     PERIOD, TIME,LAST_FALL
20BA A0282C                 LD      LAST_FALL, TIME

20BD                increment:
20BD 0736                   INC     COUNT
20BF 27CC           no_cnt: BR      wait

20C1                        END

ASSEMBLY COMPLETED,    NO ERROR(S) FOUND.
```

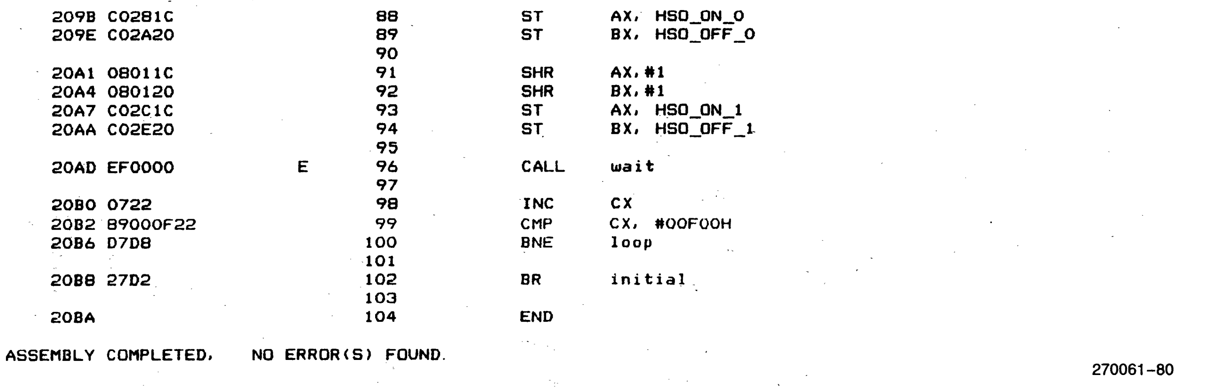
270061-78

A.5. Enhanced Pulse Measurement (Continued)



```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:HSODRV.A96
OBJECT FILE: :F3:HSODRV.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                 LINE      SOURCE STATEMENT
                                   1   $TITLE('HSODRV.A96: Driver module for HSO PWM program')
                                   2
                                   3   HSODRV          MODULE  MAIN, STACKSIZE(8)
                                   4
                                   5
                                   6           PUBLIC  HSO_ON_0 , HSO_OFF_0
                                   7           PUBLIC  HSO_ON_1 , HSO_OFF_1
                                   8           PUBLIC  HSO_TIME , HSO_COMMAND
                                   9           PUBLIC  SP , TIMER1 , IOS0
                                  10
                                  11   $INCLUDE(DEMO96.INC)
                           =1     12   $nolist  ;  Turn listing off for include file
                           =1     60            ;  End of include file
                                  61
    0028                          62   rseg at 28H
                                  63
                                  64           EXTRN   OLD_STAT        :byte
                                  65
    0028                          66           HSO_ON_0:       dsw     1
    002A                          67           HSO_OFF_0:      dsw     1
    002C                          68           HSO_ON_1:       dsw     1
    002E                          69           HSO_OFF_1:      dsw     1
    0030                          70           count:          dsb     1
                                  71
    2080                          72   cseg  at 2080H
                                  73
                                  74           EXTRN   wait    :entry
                                  75
    2080 FA                       76   strt:   DI
    2081 A1000118                 77           LD      SP, #100H
    2085 510F1500          E      78           ANDB    OLD_STAT, IOS0, #0FH
    2089 950F00            E      79           XORB    OLD_STAT, #0FH
                                  80
    208C                          81   initial:
    208C A1000122                 82           LD      CX, #0100H
                                  83
    2090 A100101C                 84   loop:   LD      AX, #1000H
    2094 48221C20                 85           SUB     BX, AX, CX
    2098 A0221C                   86           LD      AX, CX
                                  87
```

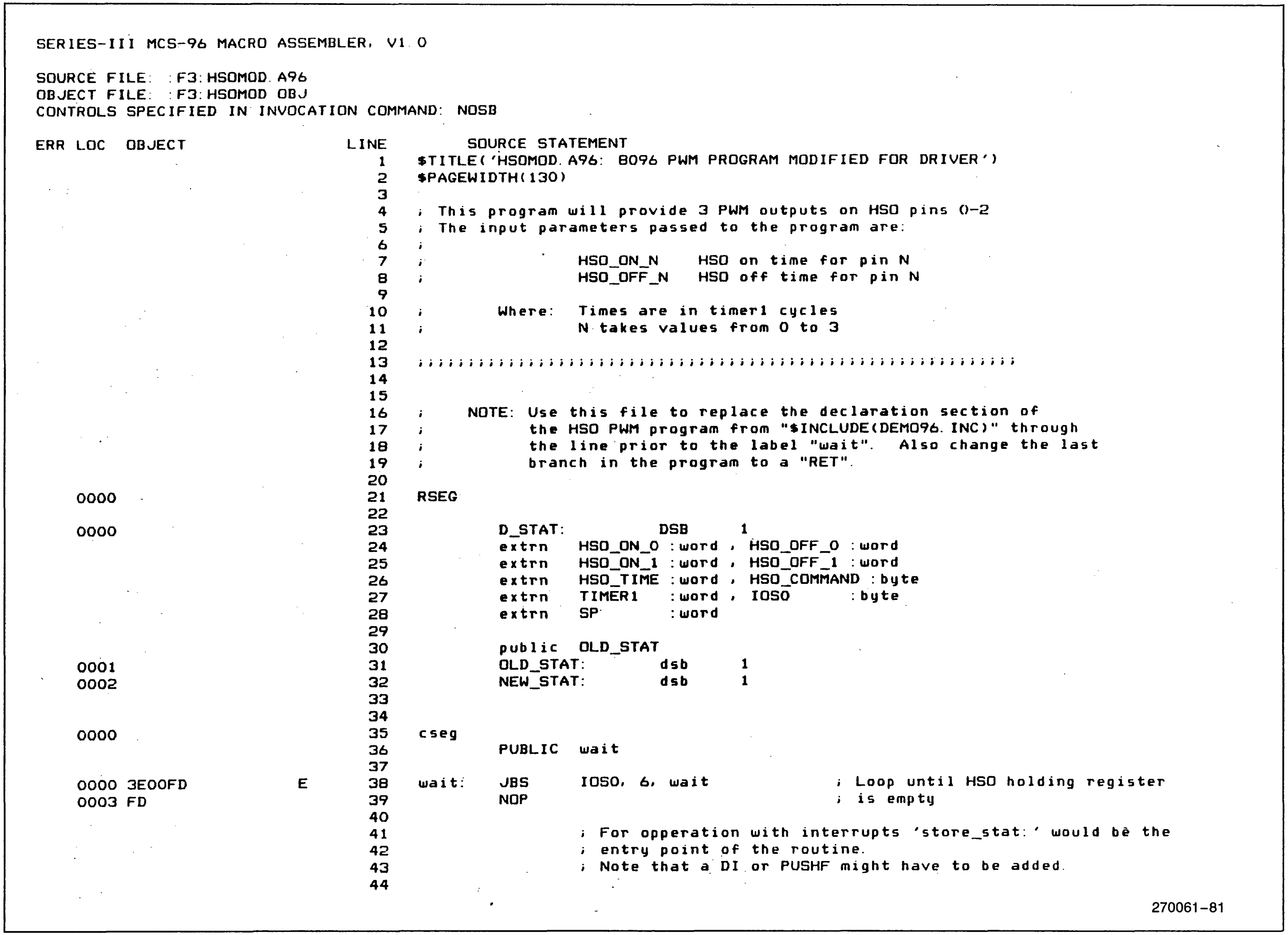
270061-79



```
 209B C0281C                 ST      AX, HSO_ON_0
 209E C02A20                 ST      BX, HSO_OFF_0

 20A1 08011C                 SHR     AX,#1
 20A4 080120                 SHR     BX,#1
 20A7 C02C1C                 ST      AX, HSO_ON_1
 20AA C02E20                 ST      BX, HSO_OFF_1

 20AD EF0000         E       CALL    wait

 20B0 0722                   INC     CX
 20B2 89000F22               CMP     CX, #00F00H
 20B6 D7D8                   BNE     loop

 20B8 27D2                   BR      initial

 20BA                        END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-80

A.6. PWM Using the HSO (Continued)



```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:HSOMOD.A96
OBJECT FILE: :F3:HSOMOD.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                  LINE       SOURCE STATEMENT
                                    1   $TITLE('HSOMOD.A96: 8096 PWM PROGRAM MODIFIED FOR DRIVER')
                                    2   $PAGEWIDTH(130)
                                    3
                                    4   ; This program will provide 3 PWM outputs on HSO pins 0-2
                                    5   ; The input parameters passed to the program are:
                                    6   ;
                                    7   ;               HSO_ON_N    HSO on time for pin N
                                    8   ;               HSO_OFF_N   HSO off time for pin N
                                    9
                                   10   ;       Where:  Times are in timer1 cycles
                                   11   ;               N takes values from 0 to 3
                                   12
                                   13   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                                   14
                                   15
                                   16   ;    NOTE: Use this file to replace the declaration section of
                                   17   ;          the HSO PWM program from "$INCLUDE(DEMO96.INC)" through
                                   18   ;          the line prior to the label "wait".  Also change the last
                                   19   ;          branch in the program to a "RET".
                                   20
    0000                           21   RSEG
                                   22
    0000                           23           D_STAT:         DSB     1
                                   24           extrn   HSO_ON_0 :word , HSO_OFF_0 :word
                                   25           extrn   HSO_ON_1 :word , HSO_OFF_1 :word
                                   26           extrn   HSO_TIME :word , HSO_COMMAND :byte
                                   27           extrn   TIMER1   :word , IOSO       :byte
                                   28           extrn   SP       :word
                                   29
                                   30           public  OLD_STAT
    0001                           31           OLD_STAT:       dsb     1
    0002                           32           NEW_STAT:       dsb     1
                                   33
                                   34
    0000                           35   cseg
                                   36           PUBLIC  wait
                                   37
    0000 3E00FD             E      38   wait:   JBS     IOSO, 6, wait           ; Loop until HSO holding register
    0003 FD                        39           NOP                             ; is empty
                                   40
                                   41                   ; For opperation with interrupts 'store_stat:' would be the
                                   42                   ; entry point of the routine.
                                   43                   ; Note that a DI or PUSHF might have to be added.
                                   44
```

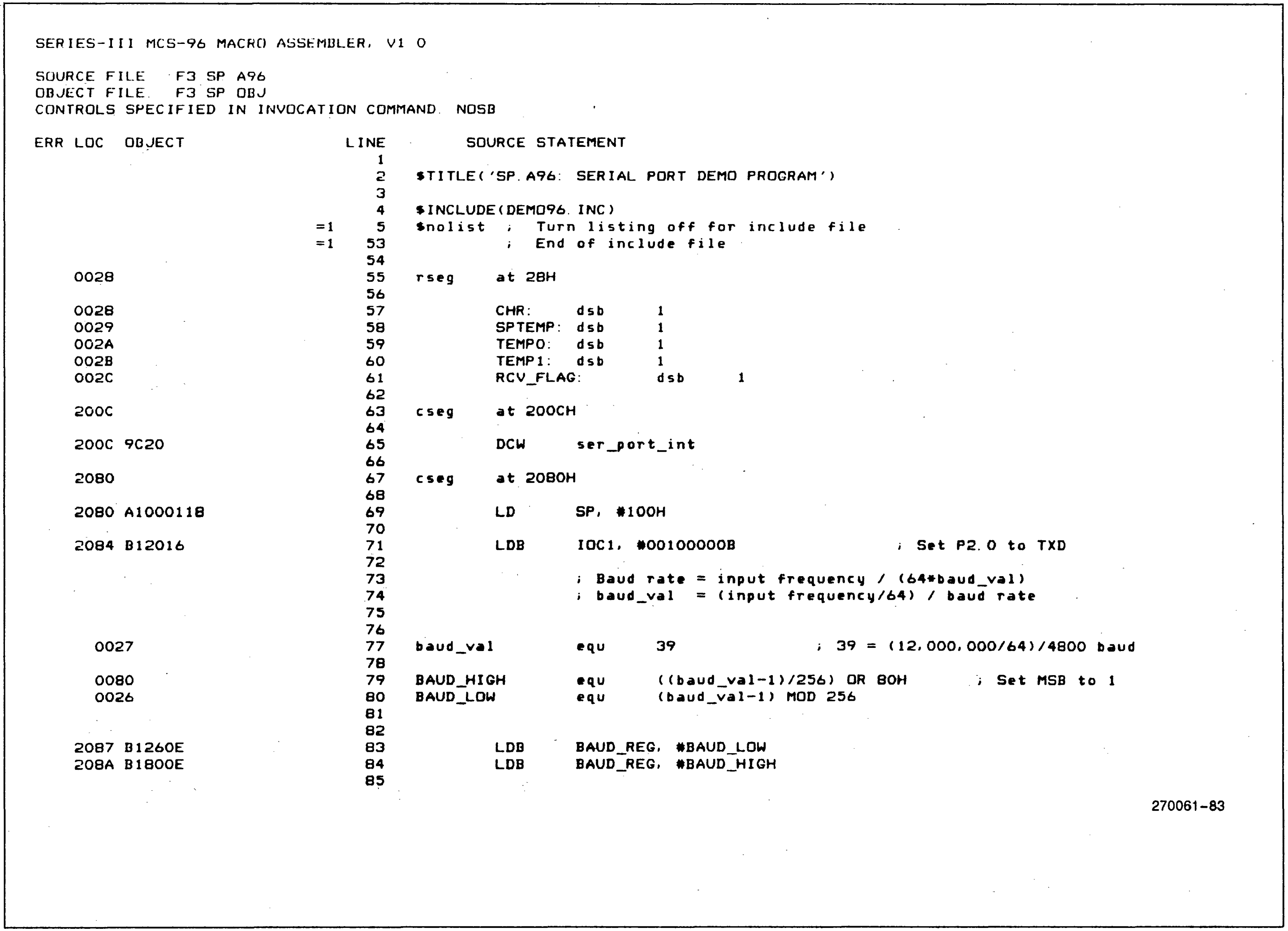
270061-81

A.6. PWM Using the HSO (Continued)

```
0004                        store_stat:
0004 510F0002      E                   ANDB NEW_STAT, IOSO, #0FH          ; Store new status of HSO
0008 980201        R                   CMPB OLD_STAT, NEW_STAT
000B DFF3                              JE   wait
000D 940201        R                   XORB OLD_STAT, NEW_STAT


0010                        check_0:
0010 300113        R                   JBC  OLD_STAT, 0, check_1         , Jump if OLD_STAT(0)=NEW_STAT(0)
0013 380209        R                   JBS  NEW_STAT, 0, set_off_0

0016                        set_on_0:
0016 B13000        E                   LDB  HSO_COMMAND, #00110000B      ; Set HSO for timer1, set pin 0
0019 44000000      E                   ADD  HSO_TIME, TIMER1, HSO_OFF_0 ; Time to set pin = Timer1 value
001D 2007                              BR   check_1                      ;  + Time for pin to be low

001F                        set_off_0:
001F B11000        E                   LDB  HSO_COMMAND, #00010000B      ; Set HSO for timer1, clear pin 0
0022 44000000      E                   ADD  HSO_TIME, TIMER1, HSO_ON_0  ; Time to clear pin = Timer1 value
                                                                         ;  + Time for pin to be high
0026                        check_1:
0026 310113        R                   JBC  OLD_STAT, 1, check_done      ; Jump if OLD_STAT(1)=NEW_STAT(1)
0029 390209        R                   JBS  NEW_STAT, 1, set_off_1

002C                        set_on_1:
002C B13100        E                   LDB  HSO_COMMAND, #00110001B      ; Set HSO for timer1, set pin 1
002F 44000000      E                   ADD  HSO_TIME, TIMER1, HSO_OFF_1 ; Time to set pin = Timer1 value
0033 2007                              BR   check_done

0035                        set_off_1:
0035 B11100        E                   LDB  HSO_COMMAND, #00010001B      ; Set HSO for timer1, clear pin 1
0038 44000000      E                   ADD  HSO_TIME, TIMER1, HSO_ON_1  ; Time to clear pin = Timer1 value
                                                                         ;  + Time for pin to be high
003C                        check_done:
003C B00201        R                   LDB  OLD_STAT, NEW_STAT           ; Store current status and
                                                                         ; wait for interrupt flag

003F F0                                RET
                                           use "BR  wait" if this routine is used with the driver

0040                                   END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-82

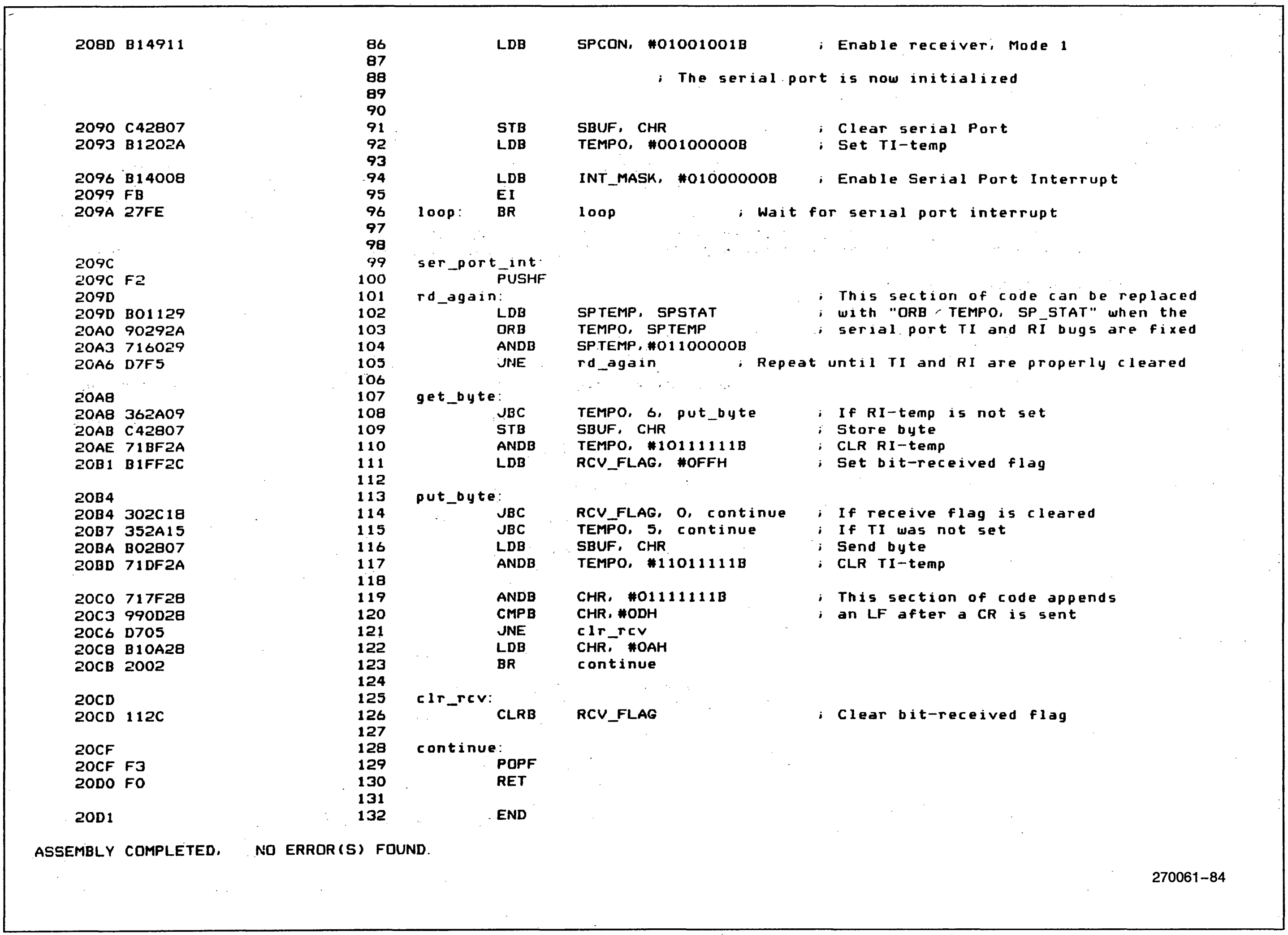
A.6. PWM Using the HSO (Continued)



```
SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE   F3 SP A96
OBJECT FILE   F3 SP OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND  NOSB

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                  1
                                  2   $TITLE('SP.A96: SERIAL PORT DEMO PROGRAM')
                                  3
                                  4   $INCLUDE(DEMO96.INC)
                           =1     5   $nolist  ;  Turn listing off for include file
                           =1    53            ;  End of include file
                                 54
    0028                         55   rseg     at 28H
                                 56
    0028                         57            CHR:     dsb     1
    0029                         58            SPTEMP:  dsb     1
    002A                         59            TEMPO:   dsb     1
    002B                         60            TEMP1:   dsb     1
    002C                         61            RCV_FLAG:        dsb     1
                                 62
    200C                         63   cseg     at 200CH
                                 64
    200C 9C20                    65            DCW      ser_port_int
                                 66
    2080                         67   cseg     at 2080H
                                 68
    2080 A1000118                69            LD       SP, #100H
                                 70
    2084 B12016                  71            LDB      IOC1, #00100000B              ; Set P2.0 to TXD
                                 72
                                 73                     ; Baud rate = input frequency / (64*baud_val)
                                 74                     ; baud_val  = (input frequency/64) / baud rate
                                 75
                                 76
      0027                       77   baud_val          equ     39             ; 39 = (12,000,000/64)/4800 baud
                                 78
      0080                       79   BAUD_HIGH         equ     ((baud_val-1)/256) OR 80H       ; Set MSB to 1
      0026                       80   BAUD_LOW          equ     (baud_val-1) MOD 256
                                 81
                                 82
    2087 B1260E                  83            LDB      BAUD_REG, #BAUD_LOW
    208A B1800E                  84            LDB      BAUD_REG, #BAUD_HIGH
                                 85
```

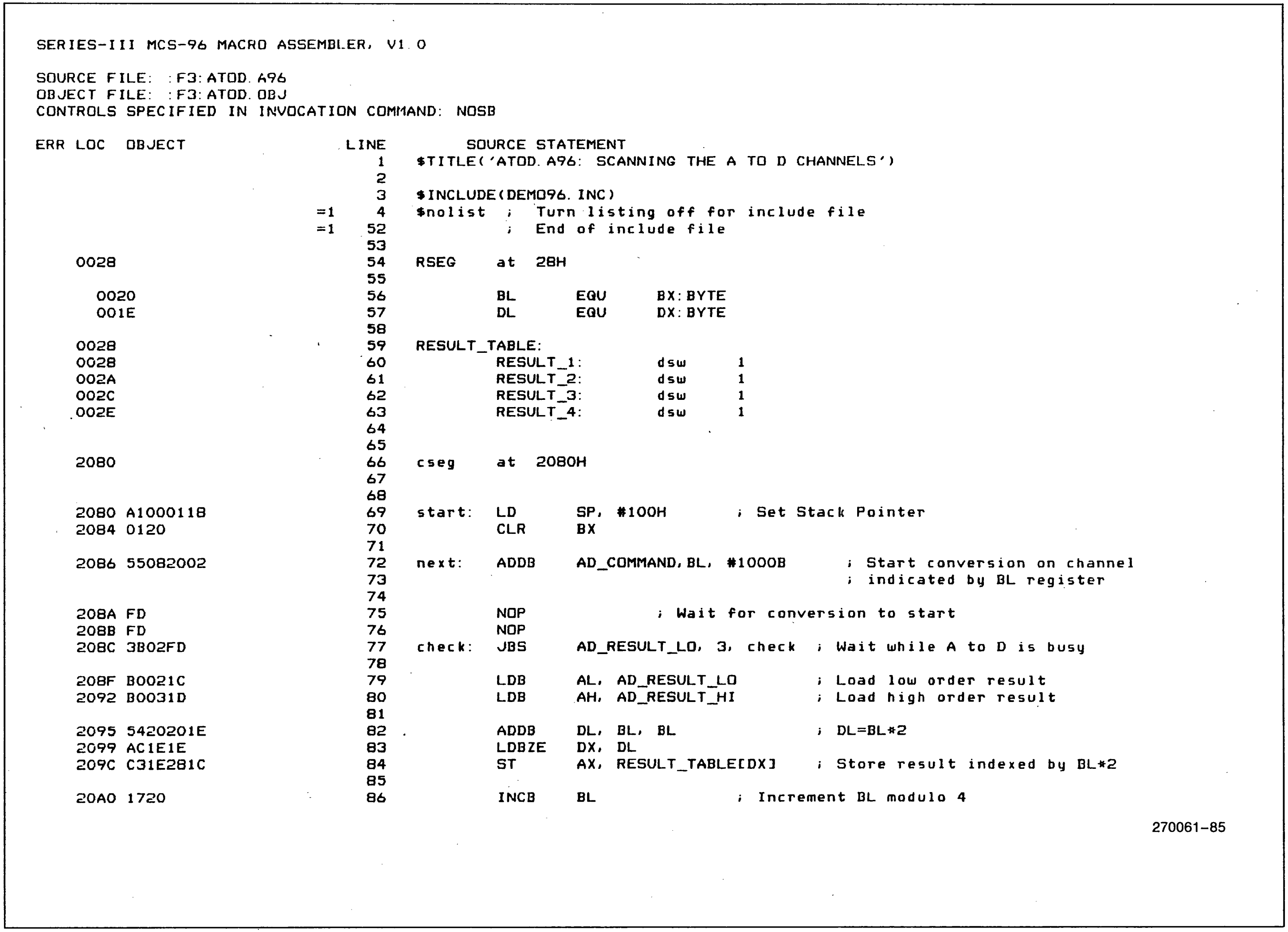
270061-83

A.7. Serial Port (Continued)



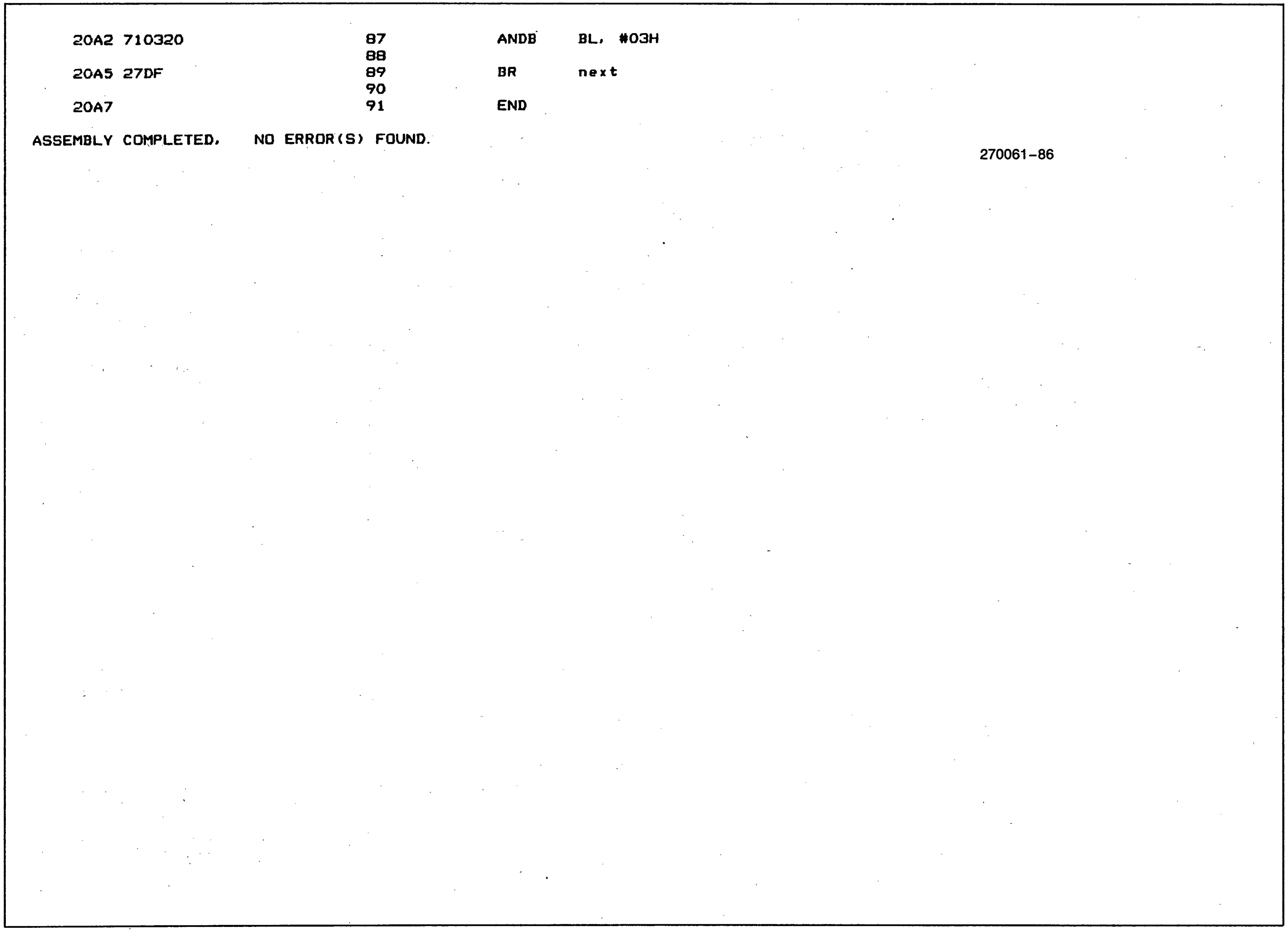
```
208D B14911          LDB     SPCON, #01001001B        ; Enable receiver, Mode 1

                                     ; The serial port is now initialized


2090 C42807          STB     SBUF, CHR                ; Clear serial Port
2093 B1202A          LDB     TEMPO, #00100000B        ; Set TI-temp

2096 B14008          LDB     INT_MASK, #01000000B     ; Enable Serial Port Interrupt
2099 FB              EI
209A 27FE    loop:   BR      loop           ; Wait for serial port interrupt


209C         ser_port_int:
209C F2              PUSHF
209D         rd_again:                                ; This section of code can be replaced
209D B01129          LDB     SPTEMP, SPSTAT           ; with "ORB TEMPO, SP_STAT" when the
20A0 90292A          ORB     TEMPO, SPTEMP            ; serial port TI and RI bugs are fixed
20A3 716029          ANDB    SPTEMP,#01100000B
20A6 D7F5            JNE     rd_again       ; Repeat until TI and RI are properly cleared

20A8         get_byte:
20A8 362A09          JBC     TEMPO, 6, put_byte       ; If RI-temp is not set
20AB C42807          STB     SBUF, CHR                ; Store byte
20AE 71BF2A          ANDB    TEMPO, #10111111B        ; CLR RI-temp
20B1 B1FF2C          LDB     RCV_FLAG, #0FFH          ; Set bit-received flag

20B4         put_byte:
20B4 302C18          JBC     RCV_FLAG, 0, continue    ; If receive flag is cleared
20B7 352A15          JBC     TEMPO, 5, continue       ; If TI was not set
20BA B02807          LDB     SBUF, CHR                ; Send byte
20BD 71DF2A          ANDB    TEMPO, #11011111B        ; CLR TI-temp

20C0 717F28          ANDB    CHR, #01111111B          ; This section of code appends
20C3 990D28          CMPB    CHR,#0DH                 ; an LF after a CR is sent
20C6 D705            JNE     clr_rcv
20C8 B10A28          LDB     CHR, #0AH
20CB 2002            BR      continue

20CD         clr_rcv:
20CD 112C            CLRB    RCV_FLAG                 ; Clear bit-received flag

20CF         continue:
20CF F3              POPF
20D0 F0              RET

20D1                 END

ASSEMBLY COMPLETED,    NO ERROR(S) FOUND.
```

270061-84



```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:ATOD.A96
OBJECT FILE:  :F3:ATOD.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                  1    $TITLE('ATOD.A96:  SCANNING THE A TO D CHANNELS')
                                  2
                                  3    $INCLUDE(DEMO96.INC)
                           =1     4    $nolist  ;  Turn listing off for include file
                           =1    52             ;  End of include file
                                 53
    0028                         54    RSEG     at  28H
                                 55
      0020                       56             BL       EQU      BX:BYTE
      001E                       57             DL       EQU      DX:BYTE
                                 58
    0028                         59    RESULT_TABLE:
    0028                         60             RESULT_1:         dsw      1
    002A                         61             RESULT_2:         dsw      1
    002C                         62             RESULT_3:         dsw      1
    002E                         63             RESULT_4:         dsw      1
                                 64
                                 65
    2080                         66    cseg     at  2080H
                                 67
                                 68
    2080 A100011B                69    start:   LD       SP, #100H        ; Set Stack Pointer
    2084 0120                    70             CLR      BX
                                 71
    2086 55082002                72    next:    ADDB     AD_COMMAND, BL, #1000B       ; Start conversion on channel
                                 73                                                   ; indicated by BL register
                                 74
    208A FD                      75             NOP               ; Wait for conversion to start
    208B FD                      76             NOP
    208C 3B02FD                  77    check:   JBS      AD_RESULT_LO, 3, check  ; Wait while A to D is busy
                                 78
    208F B0021C                  79             LDB      AL, AD_RESULT_LO        ; Load low order result
    2092 B0031D                  80             LDB      AH, AD_RESULT_HI        ; Load high order result
                                 81
    2095 5420201E                82             ADDB     DL, BL, BL              ; DL=BL*2
    2099 AC1E1E                  83             LDBZE    DX, DL
    209C C31E281C                84             ST       AX, RESULT_TABLE[DX]    ; Store result indexed by BL*2
                                 85
    20A0 1720                    86             INCB     BL               ; Increment BL modulo 4
```

270061-85

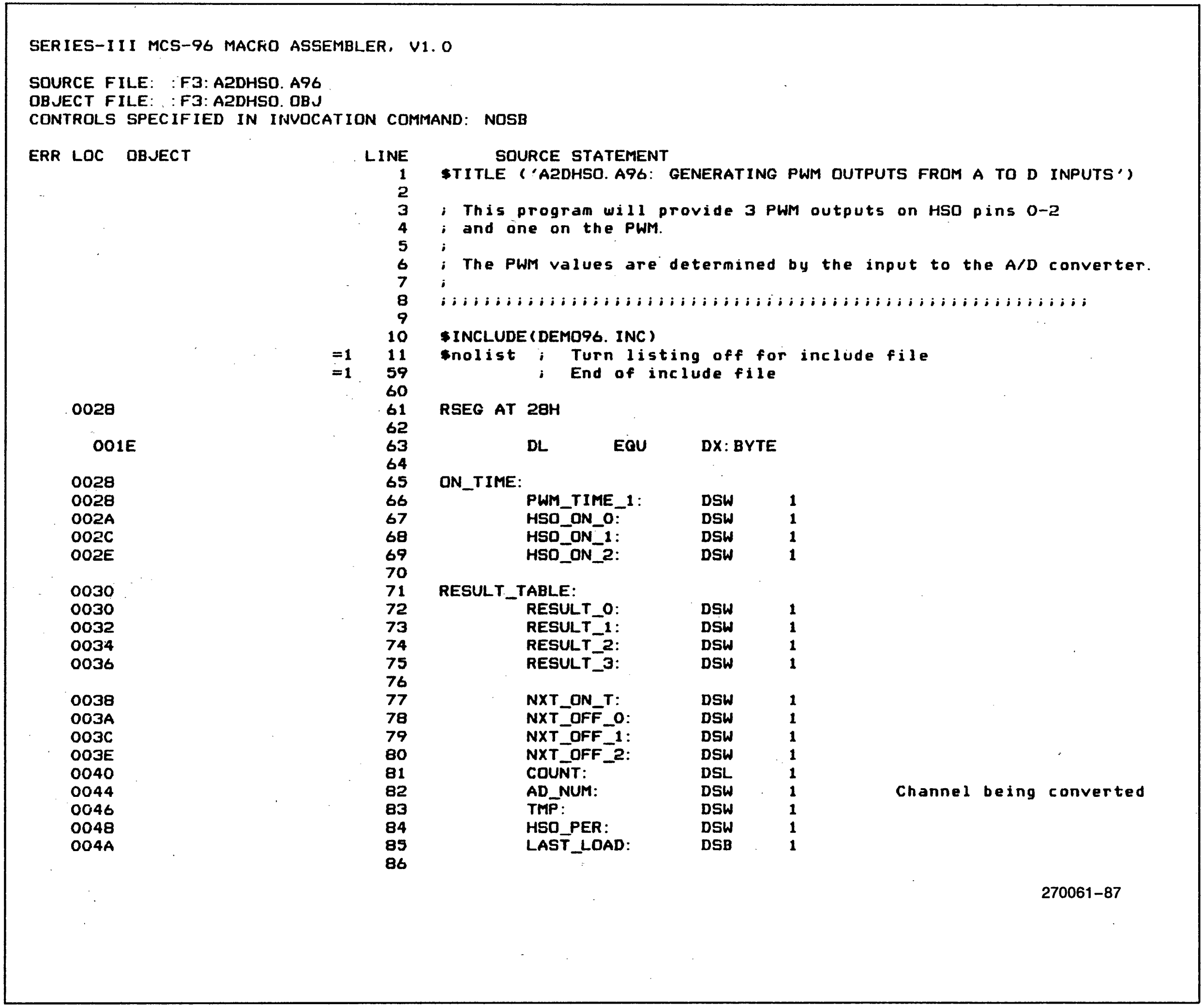


```
    20A2 710320                87          ANDB     BL, #03H
                               88
    20A5 27DF                  89          BR       next
                               90
    20A7                       91          END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-86

A.8. A to D Converter (Continued)

APPENDIX B
HSO AND A TO D UNDER INTERRUPT CONTROL



```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:A2DHSO.A96
OBJECT FILE: :F3:A2DHSO.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                   LINE       SOURCE STATEMENT
                                     1    $TITLE ('A2DHSO.A96: GENERATING PWM OUTPUTS FROM A TO D INPUTS')
                                     2
                                     3    ; This program will provide 3 PWM outputs on HSO pins 0-2
                                     4    ; and one on the PWM.
                                     5    ;
                                     6    ; The PWM values are determined by the input to the A/D converter.
                                     7    ;
                                     8    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                                     9
                                    10    $INCLUDE(DEMO96.INC)
                               =1   11    $nolist  ;  Turn listing off for include file
                               =1   59             ;  End of include file
                                    60
    0028                            61    RSEG AT 28H
                                    62
      001E                          63            DL       EQU      DX:BYTE
                                    64
    0028                            65    ON_TIME:
    0028                            66            PWM_TIME_1:      DSW      1
    002A                            67            HSO_ON_0:        DSW      1
    002C                            68            HSO_ON_1:        DSW      1
    002E                            69            HSO_ON_2:        DSW      1
                                    70
    0030                            71    RESULT_TABLE:
    0030                            72            RESULT_0:        DSW      1
    0032                            73            RESULT_1:        DSW      1
    0034                            74            RESULT_2:        DSW      1
    0036                            75            RESULT_3:        DSW      1
                                    76
    0038                            77            NXT_ON_T:        DSW      1
    003A                            78            NXT_OFF_0:       DSW      1
    003C                            79            NXT_OFF_1:       DSW      1
    003E                            80            NXT_OFF_2:       DSW      1
    0040                            81            COUNT:           DSL      1
    0044                            82            AD_NUM:          DSW      1         Channel being converted
    0046                            83            TMP:             DSW      1
    0048                            84            HSO_PER:         DSW      1
    004A                            85            LAST_LOAD:       DSB      1
                                    86
```

270061-87

```
2000                     cseg    AT 2000H

2000 8020                        DCW     start           ; Timer_ovf_int
2002 1D21                        DCW     Atod_done_int
2004 8020                        DCW     start           ; HSI_data_int
2006 CC20                        DCW     HSO_exec_int

                         $EJECT

2080                     cseg    AT 2080H

2080 A1000118            start:  LD      SP, #100H           ; Set Stack Pointer
2084 011C                        CLR     AX
2086 051C                wait:   DEC     AX                  ; wait approx. 0.2 seconds for
2088 D7FC                        JNE     wait                ; SBE to finish communications

208A 1144                        CLRB    AD_NUM

208C A1800028                    LD      PWM_TIME_1, #080H
2090 A1000148                    LD      HSO_PER, #100H
2094 A140002A                    LD      HSO_ON_0, #040H
2098 A180002C                    LD      HSO_ON_1, #080H
209C A1C0002E                    LD      HSO_ON_2, #0C0H

20A0 4500010A38                  ADD     NXT_ON_T,  Timer1, #100H

20A5 B13606                      LDB     HSO_COMMAND, #00110110B     ; Set HSO for timer1, set pin 0,1
20A8 A03804                      LD      HSO_TIME, NXT_ON_T          ; with interrupt
20AB FD                          NOP
20AC FD                          NOP
20AD B12206                      LDB     HSO_COMMAND, #00100010B     ; Set HSO for timer1, set pin 2
20B0 643804                      ADD     HSO_TIME, NXT_ON_T          ; without interrupt

20B3 91074A                      ORB     LAST_LOAD, #00000111B   ; Last loaded value was set all pins
20B6 B10A08                      LDB     INT_MASK, #00001010B    ; Enable HSO and A/D interrupts
20B9 B10A09                      LDB     INT_PENDING, #00001010B ; Fake an A/D and HSO interrupt
20BC FB                          EI

20BD 91010F              loop:   ORB     Port1, #00000001B       ; set P1.0
20C0 65010040                    ADD     COUNT, #01
20C4 A40042                      ADDC    COUNT+2, zero
20C7 71FE0F                      ANDB    Port1, #11111110B       ; clear P1.0
20CA 27F1                        BR      loop

                         $EJECT
```

270061-88

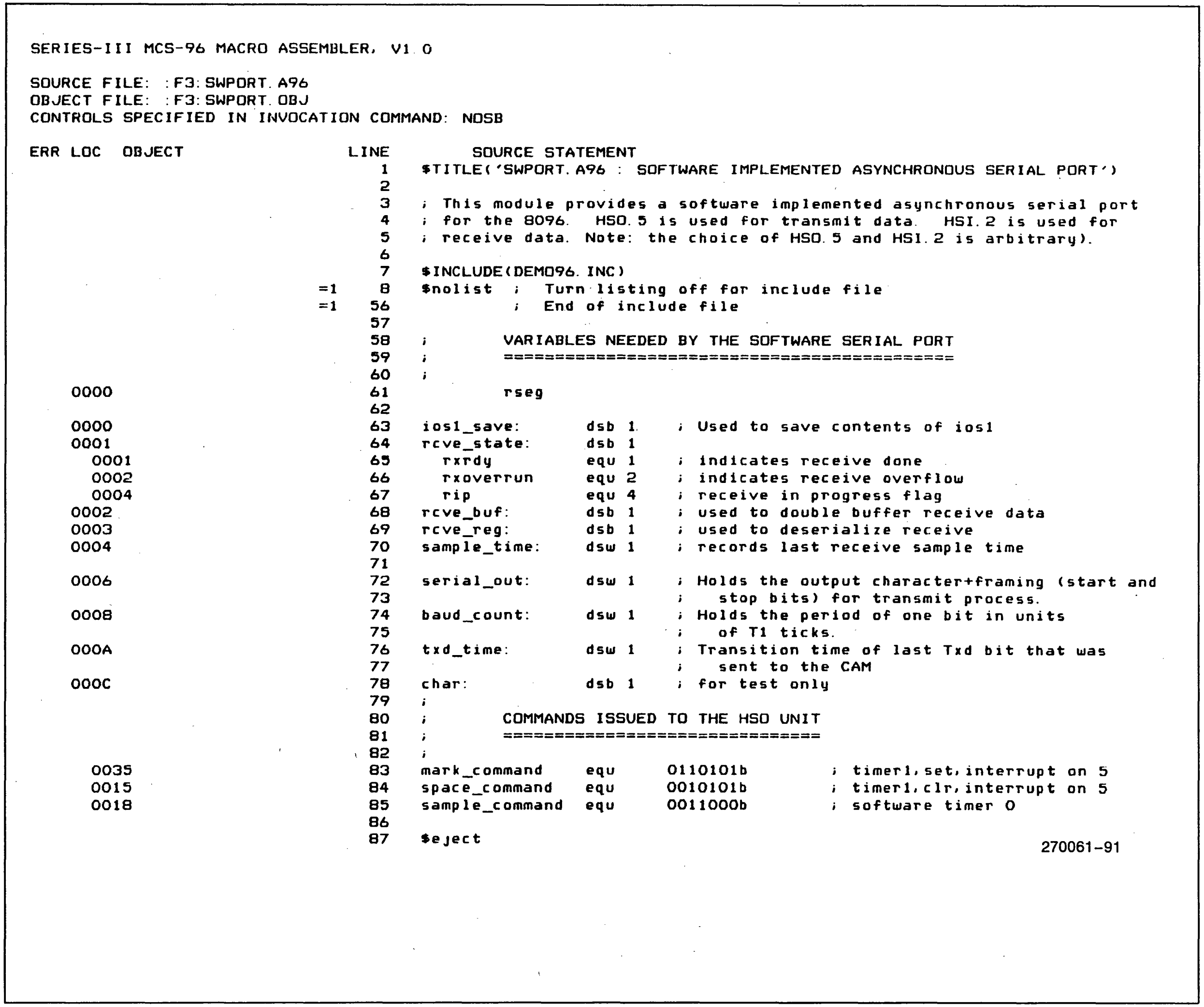
```

                       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                       ;;;;;;;;;;;;;;;;;;;;;          HSO  EXECUTED  INTERRUPT          ;;;;;;;;;;;;;;;;;;;;;;
                       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

20CC                   HSO_exec_int:
20CC F2                        PUSHF
20CD 91020F                    ORB      Port1, #00000010B        ; Set p1.1

20D0 48380A46                  SUB      TMP,TIMER1, NXT_ON_T
20D4 880046                    CMP      TMP,ZERO
20D7 DE19                      JLT      set_off_times

20D9                   set_on_times:
20D9 644838                    ADD      NXT_ON_T, HSO_PER
20DC B13606                    LDB      HSO_COMMAND, #00110110B   ; Set HSO for timer1, set pin 0,1
20DF A03804                    LD       HSO_TIME, NXT_ON_T
20E2 FD                        NOP
20E3 FD                        NOP
20E4 B12206                    LDB      HSO_COMMAND, #00100010B   ; Set HSO for timer1, set pin 2
20E7 A03804                    LD       HSO_TIME, NXT_ON_T

20EA 91074A                    ORB      LAST_LOAD, #00000111B     ; Last loaded value was all ones

20ED B02817                    LDB      PWM_CONTROL, PWM_TIME_1        ; Now is as good a time as any
                                                                       ; to update the PWM reg
20F0 2026                      BR       check_done


20F2                   set_off_times:
20F2 304A23                    JBC      LAST_LOAD, 0, check_done

20F5 442A383A                  ADD      NXT_OFF_0, NXT_ON_T, HSO_ON_0
20F9 B11006                    LDB      HSO_COMMAND, #00010000B        ; Set HSO for timer1, clear pin 0
20FC A03A04                    LD       HSO_TIME, NXT_OFF_0

20FF FD                        NOP
2100 442C383C                  ADD      NXT_OFF_1, NXT_ON_T, HSO_ON_1
2104 B11106                    LDB      HSO_COMMAND, #00010001B        ; Set HSO for timer1, clear pin 1
2107 A03C04                    LD       HSO_TIME, NXT_OFF_1

210A FD                        NOP
210B 442E383E                  ADD      NXT_OFF_2, NXT_ON_T, HSO_ON_2
210F B11206                    LDB      HSO_COMMAND, #00010010B         ; Set HSO for timer1, clear pin 2
2112 A03E04                    LD       HSO_TIME, NXT_OFF_2

2115 71F84A                    ANDB     LAST_LOAD, #11111000B   ; Last loaded value was all 0s

2118                   check_done:
2118 71FD0F                    ANDB     Port1, #11111101B        ; Clear P1.1
```

270061-89

```
211B F3                          POPF
211C F0                          RET

                         $EJECT

                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         ;;;;;;;;;;;;;;;;        A TO D  COMPLETE  INTERRUPT        ;;;;;;;;;;;;;;;;;;;;
                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

211D                     ATOD_done_int:
211D F2                          PUSHF
211E 91040F                      ORB     Port1, #00000100B       ; Set P1.2

2121 51C0021C                    ANDB    AL, AD_RESULT_LO,#11000000B     ; Load low order result
2125 B0031D                      LDB     AH, AD_RESULT_HI                ; Load high order result
2128 5444441E                    ADDB    DL, AD_NUM, AD_NUM              ; DL= AD_NUM *2
212C AC1E1E                      LDBZE   DX, DL
212F C31E301C                    ST      AX, RESULT_TABLE[DX]    ; Store result indexed by DX

2133 99401C                      CMPB    AL, #01000000B
2136 D107                        JNH     no_rnd                  ; Round up if needed
2138 99FF1D                      CMPB    AH,#0FFH        ; Don't increment if AH=0FFH
213B DF02                        JE      no_rnd
213D 171D                        INCB    AH

213F B01D1C              no_rnd: LDB     AL, AH          ; Align byte and change to word
2142 111D                        CLRB    AH
2144 C31E281C                    ST      AX, ON_TIME[DX]

2148 1744                        INCB    AD_NUM
214A 710344                      ANDB    AD_NUM, #03H            ; Keep AD_NUM between 0 and 3

214D 55084402            next:   ADDB    AD_COMMAND, AD_NUM, #1000B     ; Start conversion on channel
                                                                        ; indicated by AD_NUM register
2151 71FB0F                      ANDB    Port1, #11111011B       ; Clear P1.2
2154 F3                          POPF
2155 F0                          RET


2156                             END

ASSEMBLY COMPLETED,    NO ERROR(S) FOUND.
```

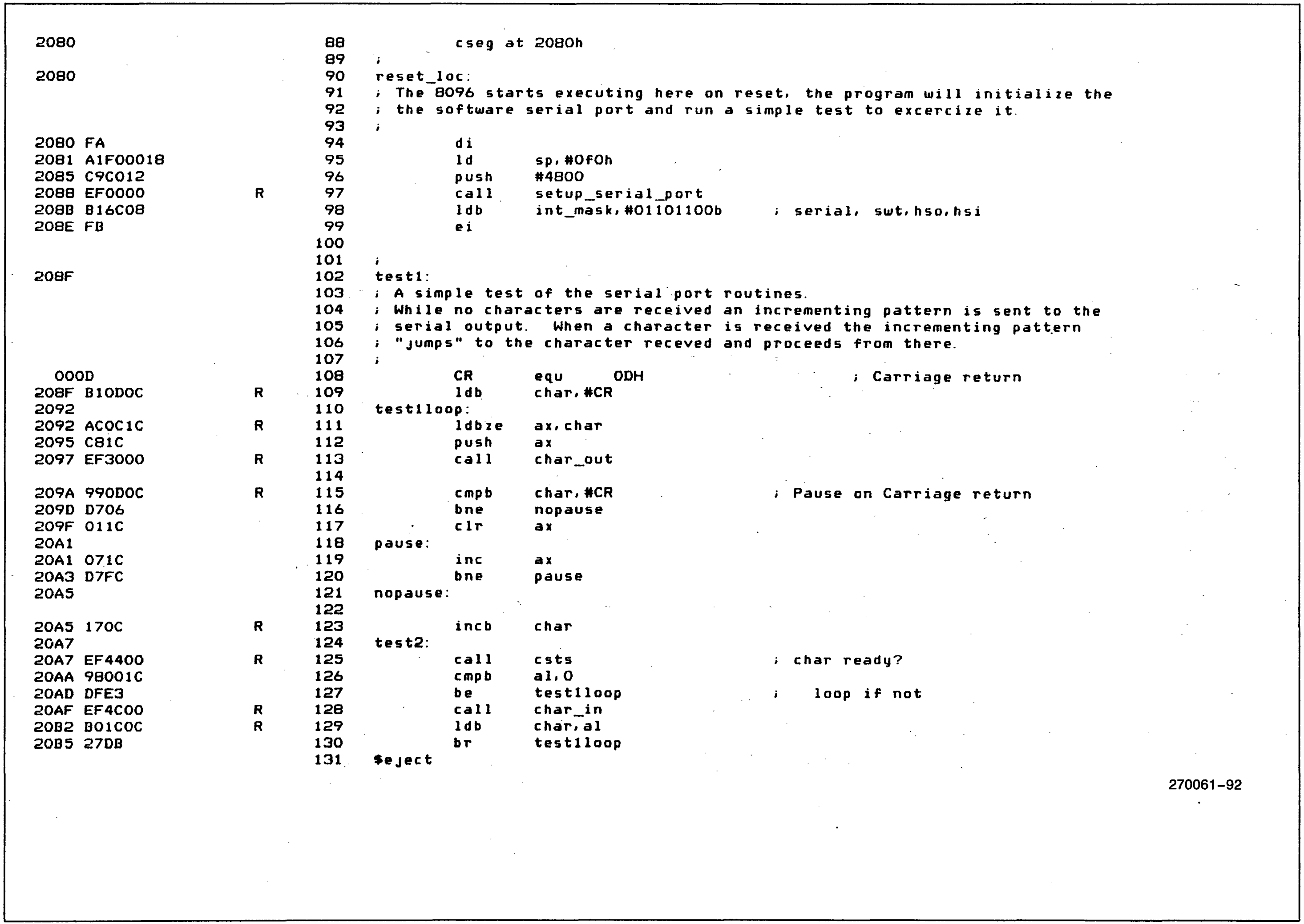
270061-90

APPENDIX C SOFTWARE SERIAL PORT



```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:SWPORT.A96
OBJECT FILE: :F3:SWPORT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                 LINE       SOURCE STATEMENT
                                   1  $TITLE('SWPORT.A96 : SOFTWARE IMPLEMENTED ASYNCHRONOUS SERIAL PORT')
                                   2
                                   3  ; This module provides a software implemented asynchronous serial port
                                   4  ; for the 8096.  HSO.5 is used for transmit data.  HSI.2 is used for
                                   5  ; receive data. Note: the choice of HSO.5 and HSI.2 is arbitrary).
                                   6
                                   7  $INCLUDE(DEMO96.INC)
                              =1   8  $nolist  ;  Turn listing off for include file
                              =1  56           ;  End of include file
                                  57
                                  58  ;        VARIABLES NEEDED BY THE SOFTWARE SERIAL PORT
                                  59  ;        ==========================================
                                  60  ;
    0000                          61           rseg
                                  62
    0000                          63  iosl_save:        dsb 1     ; Used to save contents of iosl
    0001                          64  rcve_state:       dsb 1
      0001                        65    rxrdy           equ 1     ; indicates receive done
      0002                        66    rxoverrun       equ 2     ; indicates receive overflow
      0004                        67    rip             equ 4     ; receive in progress flag
    0002                          68  rcve_buf:         dsb 1     ; used to double buffer receive data
    0003                          69  rcve_reg:         dsb 1     ; used to deserialize receive
    0004                          70  sample_time:      dsw 1     ; records last receive sample time
                                  71
    0006                          72  serial_out:       dsw 1     ; Holds the output character+framing (start and
                                  73                              ;   stop bits) for transmit process.
    0008                          74  baud_count:       dsw 1     ; Holds the period of one bit in units
                                  75                              ;   of T1 ticks.
    000A                          76  txd_time:         dsw 1     ; Transition time of last Txd bit that was
                                  77                              ;   sent to the CAM
    000C                          78  char:             dsb 1     ; for test only
                                  79  ;
                                  80  ;        COMMANDS ISSUED TO THE HSO UNIT
                                  81  ;        ==========================
                                  82  ;
      0035                        83  mark_command     equ      0110101b        ; timer1,set,interrupt on 5
      0015                        84  space_command    equ      0010101b        ; timer1,clr,interrupt on 5
      0018                        85  sample_command   equ      0011000b        ; software timer 0
                                  86
                                  87  $eject
```

270061-91



```
2080                            cseg at 2080h
                        ;
2080                    reset_loc:
                        ; The 8096 starts executing here on reset, the program will initialize the
                        ; the software serial port and run a simple test to excercize it.
                        ;
2080 FA                         di
2081 A1F00018                   ld      sp,#0F0h
2085 C9C012                     push    #4800
2088 EF0000          R          call    setup_serial_port
208B B16C08                     ldb     int_mask,#01101100b     ; serial, swt,hso,hsi
208E FB                         ei

                        ;
208F                    test1:
                        ; A simple test of the serial port routines.
                        ; While no characters are received an incrementing pattern is sent to the
                        ; serial output.  When a character is received the incrementing pattern
                        ; "jumps" to the character receved and proceeds from there.
                        ;
  000D                          CR      equ     0DH                     ; Carriage return
208F B10D0C          R          ldb     char,#CR
2092                    test1loop:
2092 AC0C1C          R          ldbze   ax,char
2095 C81C                       push    ax
2097 EF3000          R          call    char_out

209A 990D0C          R          cmpb    char,#CR                ; Pause on Carriage return
209D D706                       bne     nopause
209F 011C                       clr     ax
20A1                    pause:
20A1 071C                       inc     ax
20A3 D7FC                       bne     pause
20A5                    nopause:

20A5 170C            R          incb    char
20A7                    test2:
20A7 EF4400          R          call    csts                    ; char ready?
20AA 98001C                     cmpb    al,0
20AD DFE3                       be      test1loop               ;   loop if not
20AF EF4C00          R          call    char_in
20B2 B01C0C          R          ldb     char,al
20B5 27DB                       br      test1loop
                        $eject
```

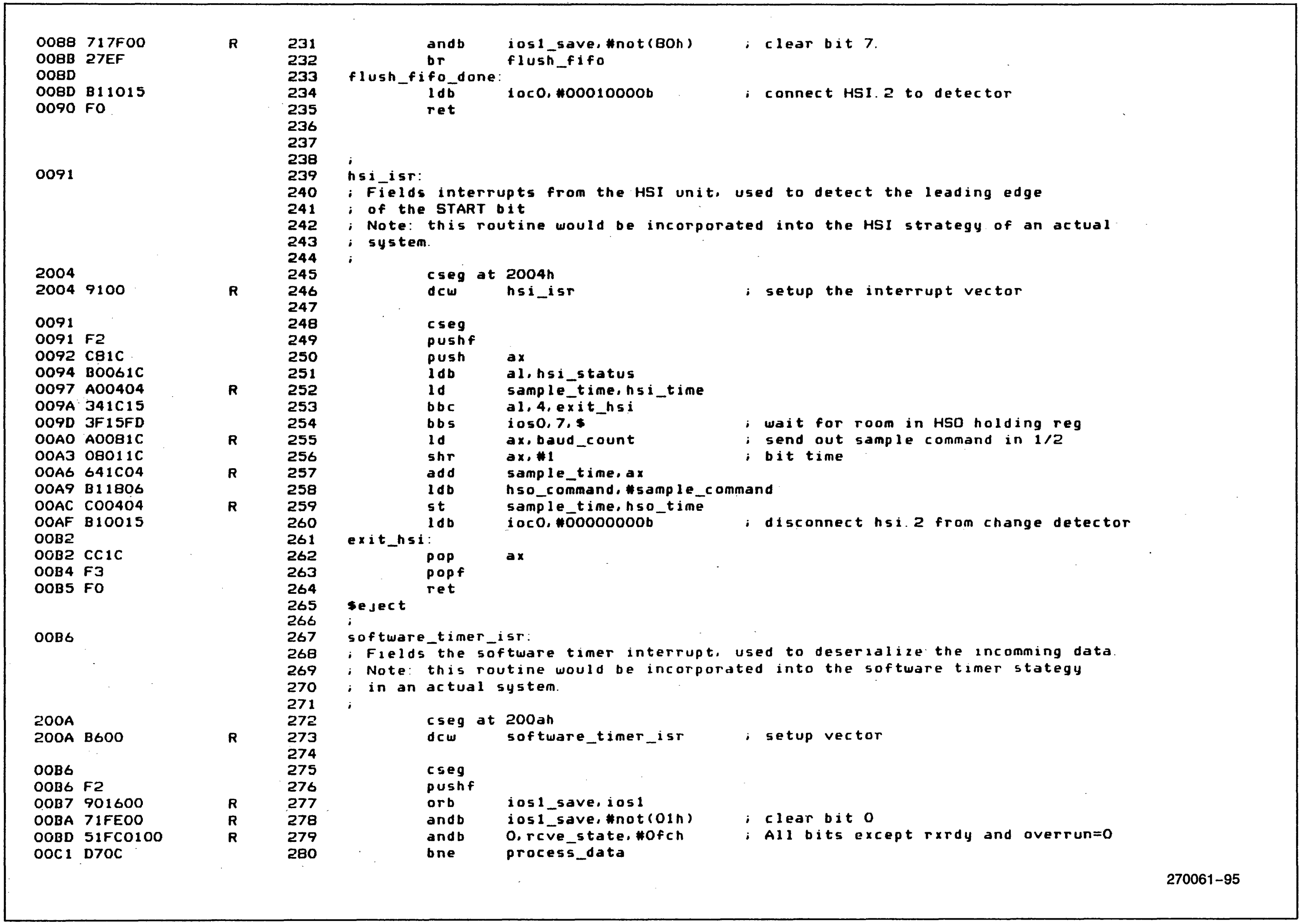
270061-92

```
0000                           cseg
                       ;
0000                   setup_serial_port:
                       ; Called on system reset to intiate the software serial port.
                       ;
0000 CC22                      pop     cx                 ; the return address
0002 CC20                      pop     bx                 ; the baud rate (in decimal)
0004 A107001E                  ld      dx,#0007h          ; dx:ax:=500,000 (assumes 12 Mhz crystal)
0008 A120A11C                  ld      ax,#0A120h
000C 8C201C                    divu    ax,bx              ; calculate the baud count (500,000/baudrate)
000F C0081C          R         st      ax,baud_count
0012 C00600          R         st      0,serial_out       ; clear serial out
0015 B16016                    ldb     ioc1,#01100000b    ; Enable HSO.5 and Txd
0018 3E15FD                    bbs     ios0,6,$           ; Wait for room in the HSO CAM
                                                          ; and issue a MARK command.
001B 44140A0A        R         add     txd_time,timer1,20
001F B13506                    ldb     hso_command,#mark_command
0022 A00A04          R         ld      hso_time,txd_time
0025 1102            R         clrb    rcve_buf           ; clear out the receive variables
0027 1103            R         clrb    rcve_reg
0029 1101            R         clrb    rcve_state
002B EF4800                    call    init_receive       ; setup to detect a start bit
002E E322                      br      [cx]               ; return
                       $eject
                       ;
0030                   char_out:
                       ; Output character to the software serial port
                       ;
0030 CC22                      pop     cx                 ; the return address
0032 CC20                      pop     bx                 ; the character for output
0034 B10121                    ldb     (bx+1),#01h        ; add the start and stop bits
0037 642020                    add     bx,bx              ;   to the char and leave as 16 bit
003A                   wait_for_xmit:
003A 880006          R         cmp     serial_out,0       ; wait for serial_out=0 (it will be cleared by
003D D7FB                      bne     wait_for_xmit      ;   the hso interrupt process)
003F C00620          R         st      bx,serial_out      ; put the formatted character in serial_out
0042 E322                      br      [cx]               ; return to caller
                       ;
0044                   csts:
                       ; Returns "true" (ax<>0) if char_in has a character.
                       ;
0044 011C                      clr     ax
0046 300102          R         bbc     rcve_state,0,csts_exit
0049 071C                      inc     ax
004B                   csts_exit:
004B F0                        ret
                       ;
004C                   char_in:
```

270061-93

```
                       ; Get a character from the software serial port
                       ;
                                                         ; wait for character ready
004C 3001FD       R            bbc     rcve_state,0,char_in
004F F2                        pushf                     ; set up a critical region
0050 71FE01       R            andb    rcve_state,#not(rxrdy)
0053 AC021C       R            ldbze   al,rcve_buf
0056 F3                        popf                      ; leave the critical region
0057 F0                        ret
                       $eject
                       ;
0058                   hso_isr:
                       ; Fields the hso interrupts and performs the serialization of the data.
                       ; Note: this routine would be incorporated into the hso service strategy
                       ;       for an actual system.

2006                           cseg    at 2006h
2006 5800         R            dcw     hso_isr           ; Set up vector

0058                           cseg
0058 F2                        pushf
0059 64080A       R            add     txd_time,baud_count
005C 880006       R            cmp     serial_out,0      ; if character is done send a mark
005F DF0D                      be      send_mark
0061 080106       R            shr     serial_out,#1     ; else send bit 0 of serial_out and shift
0064 DB08                      bc      send_mark         ;    serial_out left one place.
0066                   send_space:
0066 B11506                    ldb     hso_command,#space_command
0069 A00A04       R            ld      hso_time,txd_time
006C 2006                      br      hso_isr_exit
006E                   send_mark:
006E B13506                    ldb     hso_command,#mark_command
0071 A00A04       R            ld      hso_time,txd_time

0074                   hso_isr_exit:
0074 F3                        popf
0075 F0                        ret
                       $eject
                       ;
0076                   init_receive:
                       ; Called to prepare the serial input process to find the leading edge of
                       ; a start bit.
                       ;
0076 B10015                    ldb     ioc0,#00000000b           ; disconnect change detector
0079 B12003                    ldb     hsi_mode,#00100000b       ; negative edges on HSI.2
007C                   flush_fifo:
007C 901600       R            orb     ios1_save,ios1
007F 37000B       R            bbc     ios1_save,7,flush_fifo_done
0082 B0061C                    ldb     al,hsi_status
0085 A0041C                    ld      ax,hsi_time               ; trash the fifo entry
```

270061-94



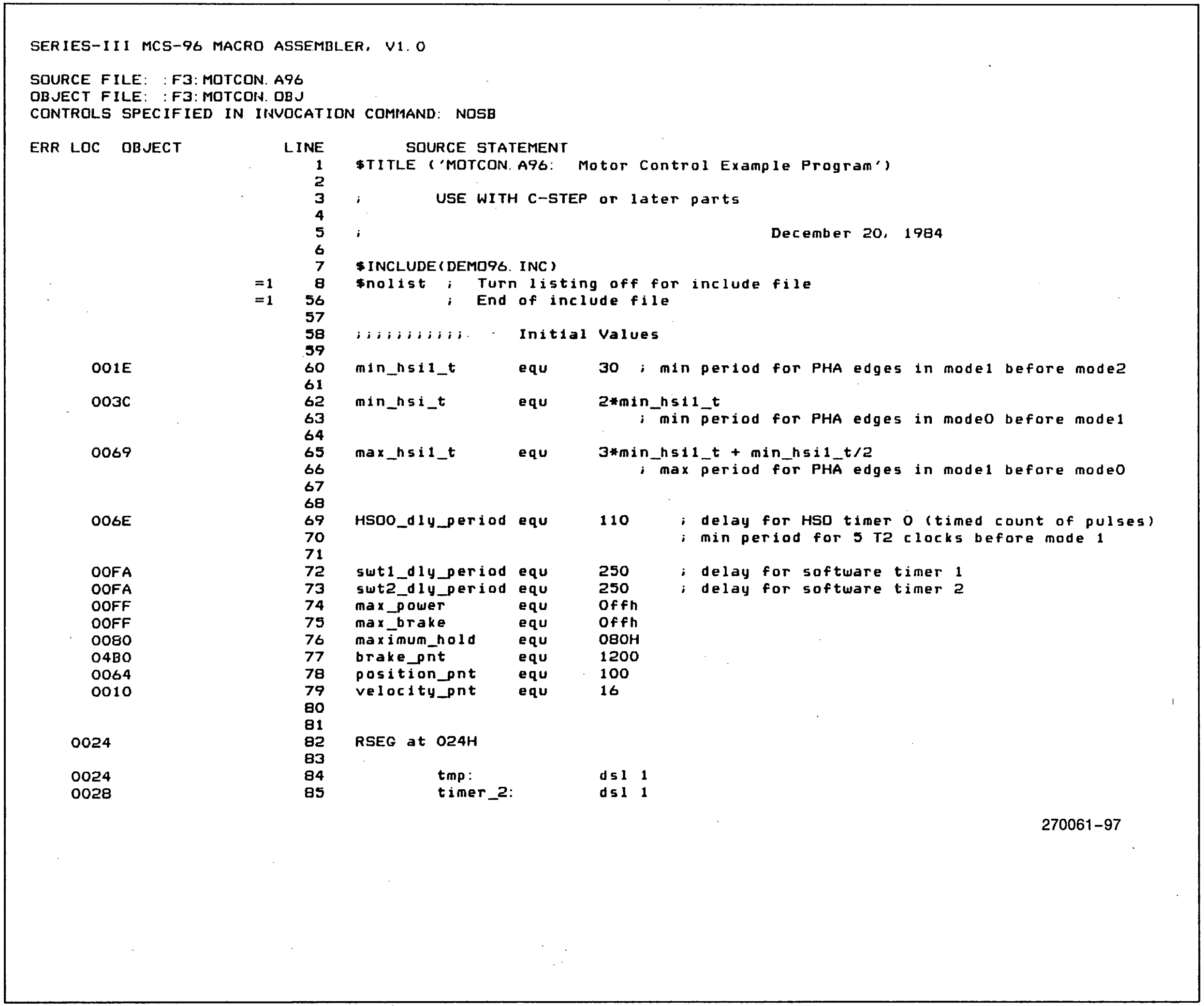
```
0088 717F00       R        andb    ios1_save,#not(80h)      ; clear bit 7.
008B 27EF                  br      flush_fifo
008D                 flush_fifo_done:
008D B11015                ldb     ioc0,#00010000b          ; connect HSI.2 to detector
0090 F0                    ret



                     ;
0091                 hsi_isr:
                     ; Fields interrupts from the HSI unit, used to detect the leading edge
                     ; of the START bit
                     ; Note: this routine would be incorporated into the HSI strategy of an actual
                     ; system.
                     ;
2004                       cseg at 2004h
2004 9100         R        dcw     hsi_isr                  ; setup the interrupt vector

0091                       cseg
0091 F2                    pushf
0092 C81C                  push    ax
0094 B0061C                ldb     al,hsi_status
0097 A00404       R        ld      sample_time,hsi_time
009A 341C15                bbc     al,4,exit_hsi
009D 3F15FD                bbs     ios0,7,$                 ; wait for room in HSO holding reg
00A0 A0081C       R        ld      ax,baud_count            ; send out sample command in 1/2
00A3 08011C                shr     ax,#1                    ; bit time
00A6 641C04       R        add     sample_time,ax
00A9 B11806                ldb     hso_command,#sample_command
00AC C00404       R        st      sample_time,hso_time
00AF B10015                ldb     ioc0,#00000000b          ; disconnect hsi.2 from change detector
00B2                 exit_hsi:
00B2 CC1C                  pop     ax
00B4 F3                    popf
00B5 F0                    ret
                     $eject
                     ;
00B6                 software_timer_isr:
                     ; Fields the software timer interrupt, used to deserialize the incomming data.
                     ; Note: this routine would be incorporated into the software timer stategy
                     ; in an actual system.
                     ;
200A                       cseg at 200ah
200A B600         R        dcw     software_timer_isr       ; setup vector

00B6                       cseg
00B6 F2                    pushf
00B7 901600       R        orb     ios1_save,ios1
00BA 71FE00       R        andb    ios1_save,#not(01h)      ; clear bit 0
00BD 51FC0100     R        andb    0,rcve_state,#0fch       ; All bits except rxrdy and overrun=0
00C1 D70C                  bne     process_data
```

270061-95

```
  00C3                         281  process_start_bit:
  00C3 350604                  282          bbc     hsi_status,5,start_ok
  00C6 2FAE                    283          call    init_receive
  00C8 2032                    284          br      software_timer_exit
  00CA                         285  start_ok:
  00CA 910401           R      286          orb     rcve_state,#rip ; set receive in progress flag
  00CD 2021                    287          br      schedule_sample
                               288
  00CF                         289  process_data:
  00CF 3F010E           R      290          bbs     rcve_state,7,check_stopbit
  00D2 180103           R      291          shrb    rcve_reg,#1
  00D5 350603                  292          bbc     hsi_status,5,datazero
  00D8 918003           R      293          orb     rcve_reg,#80h    ; set the new data bit
  00DB                         294  datazero:
  00DB 751001           R      295          addb    rcve_state,#10h ; increment bit count
  00DE 2010                    296          br      schedule_sample
                               297
  00E0                         298  check_stopbit:
  00E0 3506FD                  299          bbc     hsi_status,5,$   ; DEBUG ONLY
  00E3 B00302           R      300          ldb     rcve_buf,rcve_reg
  00E6 910101           R      301          orb     rcve_state,#rxrdy
  00E9 710301           R      302          andb    rcve_state,#03h ; Clear all but ready and overrun bits
  00EC 2F88                    303          call    init_receive
  00EE 200C                    304          br      software_timer_exit
                               305
  00F0                         306  schedule_sample:
  00F0 3F15FD                  307          bbs     ios0,7,$         ; wait for holding reg empty
  00F3 B11806                  308          ldb     hso_command,#sample_command
  00F6 640804           R      309          add     sample_time,baud_count
  00F9 C00404           R      310          st      sample_time,hso_time
                               311
  00FC                         312  software_timer_exit:
  00FC F3                      313          popf
  00FD F0                      314          ret
                               315
                               316
  00FE                         317          end

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-96

APPENDIX D
MOTOR CONTROL PROGRAM



```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:MOTCON.A96
OBJECT FILE: :F3:MOTCON.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT          LINE        SOURCE STATEMENT
                            1   $TITLE ('MOTCON.A96:  Motor Control Example Program')
                            2
                            3   ;       USE WITH C-STEP or later parts
                            4
                            5   ;                                           December 20, 1984
                            6
                            7   $INCLUDE(DEMO96.INC)
                       =1   8   $nolist  ;  Turn listing off for include file
                       =1  56            ;  End of include file
                           57
                           58   ;;;;;;;;;;;     Initial Values
                           59
      001E                 60   min_hsi1_t       equ     30  ; min period for PHA edges in mode1 before mode2
                           61
      003C                 62   min_hsi_t        equ     2*min_hsi1_t
                           63                                ; min period for PHA edges in mode0 before mode1
                           64
      0069                 65   max_hsi1_t       equ     3*min_hsi1_t + min_hsi1_t/2
                           66                                ; max period for PHA edges in mode1 before mode0
                           67
                           68
      006E                 69   HSO0_dly_period equu     110      ; delay for HSO timer 0 (timed count of pulses)
                           70                                     ; min period for 5 T2 clocks before mode 1
                           71
      00FA                 72   swt1_dly_period equ      250      ; delay for software timer 1
      00FA                 73   swt2_dly_period equ      250      ; delay for software timer 2
      00FF                 74   max_power        equ     0ffh
      00FF                 75   max_brake        equ     0ffh
      0080                 76   maximum_hold     equ     080H
      04B0                 77   brake_pnt        equ     1200
      0064                 78   position_pnt     equ     100
      0010                 79   velocity_pnt     equ     16
                           80
                           81
    0024                   82   RSEG at 024H
                           83
    0024                   84           tmp:            dsl 1
    0028                   85           timer_2:        dsl 1
```

270061-97

```
002C                    tmr2_old:       dsl 1
0030                    position:       dsl 1
0034                    des_pos:        dsl 1
0038                    pos_err:        dsl 1
003C                    delta_p:        dsl 1
0040                    time:           dsl 1
0044                    des_time:       dsl 1
0048                    time_err:       dsl 1

               $EJECT

004C                    last_time_err:  dsw 1
004E                    last_pos_err:   dsw 1
0050                    pos_delta:      dsw 1
0052                    time_delta:     dsw 1
0054                    last_pos:       dsw 1
0056                    last1_time:     dsw 1
0058                    last2_time:     dsw 1
005A                    boost:          dsw 1
005C                    tmp1:           dsw 1
005E                    out_ptr:        dsw 1
0060                    offset:         dsw 1
0062                    nxt_pos:        dsw 1
0064                    rpwr:           dsw 1
0066                    old_t2:         dsw 1

0068                    direct:         dsb 1    ;   1=forward, 0=reverse
0069                    pwm_dir:        dsb 1
006A                    hsi_s0:         dsb 1
006B                    last_stat:      dsb 1
006C                    pwm_pwr:        dsb 1
006D                    iosl_bak:       dsb 1
006E                    TR_COL:         DSB 1    ;  COLLECT TRACE IF TR_COL=00
006F                    main_dly:       dsb 1

0070                    max_pwr:        dsw 1
0072                    max_brk:        dsw 1
0074                    max_hold:       dsw 1
0076                    vel_pnt:        dsw 1
0078                    brk_pnt:        dsw 1
007A                    pos_pnt:        dsw 1
007C                    HSO0_dly:       dsw 1
007E                    swt1_dly:       dsw 1
0080                    swt2_dly:       dsw 1
0082                    min_hsi:        dsw 1
0084                    min_hsil:       dsw 1
0086                    max_hsil:       dsw 1


0100           dseg at 100H
```

270061-98

```
0100                mode_view:    dsb      1
0102                count_out:    dsw      1
0104                err_view:     dsw      1


                    $eject

                    ;             PIN#     PORT     FLAG USAGE

                    ;             22       P1.0     mode0 0   mode1 1   mode2 1  or 0
                    ,             23       P1.1           0         0         1     1
                    ;             24       P1.2     software timer 2 routine enter/leave
                    ;             25       P1.3     Main program toggle
                    ;             26       P1.4     HSI overflow toggle
                    ;             37       P1.5     software timer 0 routine enter/leave
                    ;             38       P1.6     hsi_int enter/leave
                    ;             39       P1.7     software timer 1 routine enter/leave
                    ;             40       P2.6     Input direction (0=reverse, 1=forward)
                    ;             45       P2.7     direction 0=rev, 1=fwd

2000                cseg          at       2000H
2000 0022                         dcw      timer_ovf_int
2002 1020                         dcw      atod_done_int
2004 0424                         dcw      hsi_data_int
2006 8022                         dcw      hso_exec_int
2008 1020                         dcw      hsi_0_int
200A 2022                         dcw      soft_tmr_int
200C 1020                         dcw      ser_port_int
200E 1020                         dcw      external_int

2010                atod_done_int:
2010                hsi_0_int:
2010                ser_port_int:
2010                external_int:

2080                cseg          at       2080H

2080 A1F00018       init:         ld       sp,#0F0H
2084 B1FF17                       ldb      pwm_control,#0FFH

2087 1168                         clrb     direct
2089 A170175C                     ld       tmp1,#6000             ; wait about 3 seconds for motor
208D 055C           delay:        dec      tmp1          ;    to come to a stop
208F E068FD                       djnz     direct,$               ; wait 0.512 milliseconds
2092 88005C                       cmp      tmp1,zero
2095 D2F6                         jgt      delay

2097 B1FF0F                       ldb      port1,#0FFH
209A B1FF10                       ldb      port2,#0ffH
```

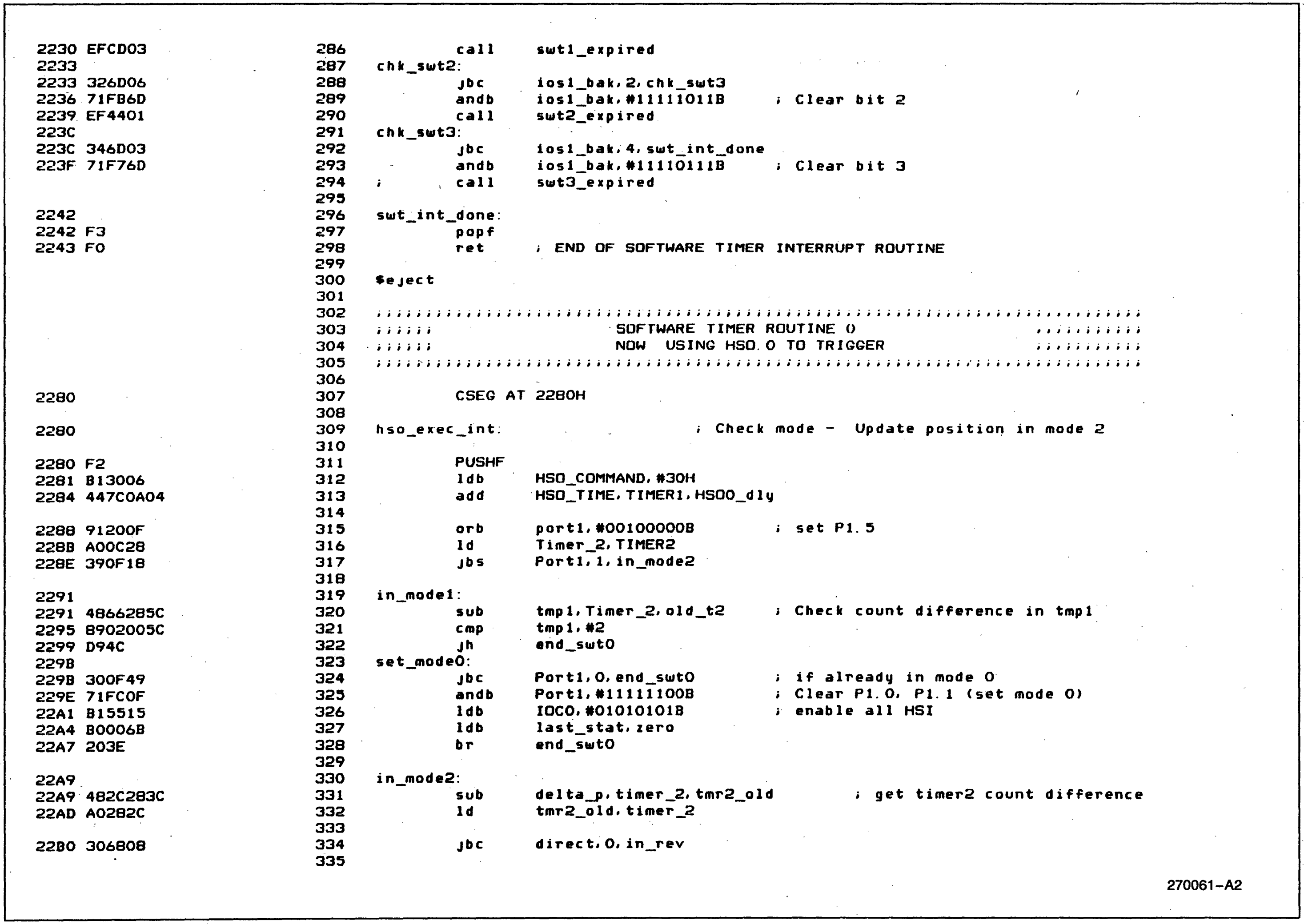
270061-99

```
209D B12516                 ldb    IOC1,#00100101B ; Disable HSO.4,HSO.5, HSI_INT=first,
                                                   ; Enable PWM,TXD,TIMER1_OVRFLOW_INT

20A0 71FC0F                 andb   Port1,#11111100B        ; clear P1.0,1 (set mode 0)
20A3 B19903                 ldb    HSI_mode,#10011001B     ; set hsi.1,3 -; hsi.0,2 +
20A6 B15715                 ldb    IOCO,#01010111B         ; Enable all hsi
                                                           ; T2 CLOCK=T2CLK, T2RST=T2RST
                                                           ; Clear timer2
                    $eject

20A9 A00400                 ld     zero,hsi_time
20AC 0140                   clr    time
20AE 0142                   clr    time+2
20B0 0128                   clr    timer_2
20B2 012A                   clr    timer_2+2
20B4 0130                   clr    position
20B6 0132                   clr    position+2
20B8 0154                   clr    last_pos
20BA 0134                   clr    des_pos
20BC 0136                   clr    des_pos+2
20BE 0144                   clr    des_time
20C0 0146                   clr    des_time+2
20C2 A00A56                 ld     last1_time,Timer1
20C5 4900085658             sub    last2_time,last1_time,#800H
20CA 116D                   clrb   ios1_bak
20CC 1109                   clrb   int_pending
20CE A1F0015E               ld     out_ptr,#1F0H
20D2 A13C0082               ld     min_hsi,#min_hsi_t
20D6 A11E0084               ld     min_hsi1,#min_hsi1_t
20DA A1690086               ld     max_hsi1,#max_hsi1_t
20DE A16E007C               ld     HSO0_dly,#HSO0_dly_period
20E2 A1FA007E               ld     swt1_dly,#swt1_dly_period
20E6 A1FA0080               ld     swt2_dly,#(swt2_dly_period)
20EA A1FF0070               ld     max_pwr,#max_power
20EE A1FF0072               ld     max_brk,#max_brake
20F2 A1800074               ld     max_hold,#maximum_hold
20F6 A1B00478               ld     brk_pnt,#brake_pnt
20FA A164007A               ld     pos_pnt,#position_pnt
20FE A1100076               ld     vel_pnt,#velocity_pnt
2102 A1002962               ld     nxt_pos,#pos_table
2106 B0006C                 ldb    pwm_pwr,zero
2109 B10169                 ldb    pwm_dir,#01h            ; FORWARD

210C B12D08                 ldb    int_mask,#00101101B     ; Enable tmr_ovf, hsi, swt, HSO,interrupts
210F B13006                 ldb    hso_command,#30H        ; set HSO_0
2112 447C0A04               add    hso_time,timer1,HSO0_dly
2116 FD                     nop
2117 FD                     NOP
2118 B13906                 ldb    hso_command,#39H        ; set swt_1
211B 447E0A04               add    hso_time,timer1,swt1_dly
```

270061-A0

```
211F FD                          nop
2120 FD                          nop
2121 B13A06                      ldb     hso_command,#3AH         ; set swt_2
2124 44800A04                    add     hso_time,timer1,swt2_dly

2128 A00A40                      ld      time,TIMER1
212B A00C2C                      ld      tmr2_old,timer2
212E FB                          ei

212F E7CE06                      br      main_prog

                       $eject


                       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                       ;;;;;;                   TIMER  INTERRUPT  SERVICE                 ;;;;;;;;;;;
                       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

2200                             CSEG AT 2200H

2200                   timer_ovf_int:
2200 F2                          pushf

2201 90166D                      orb     ios1_bak,IOS1
2204 356D05            chk_t1:   jbc     ios1_bak,5,tmr_int_done
2207 0742                        inc     time+2
2209 71DF6D                      andb    ios1_bak,#11011111B      ; clear bit 5
220C                   tmr_int_done:
220C F3                          popf
220D F0                          ret              ; End of timer interrupt routine



                       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                       ;;;;;        SOFTWARE TIMER INTERRUPT SERVICE ROUTINE           ;;;;;;;;;
                       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

2220                             CSEG AT 2220H


2220                   soft_tmr_int:
2220 F2                          pushf
2221 90166D                      orb     ios1_bak,IOS1
2224                   chk_swt0:
2224 306D03                      jbc     ios1_bak,0,chk_swt1
2227 71FE6D                      andb    ios1_bak,#11111110B      ; Clear bit 0 - end swt0
                       ;         call    swt0_expired
222A                   chk_swt1:
222A 316D06                      jbc     ios1_bak,1,chk_swt2
222D 71FD6D                      andb    ios1_bak,#11111101B      ; Clear bit 1
```

270061-A1



```
2230 EFCD03                     call    swt1_expired
2233                    chk_swt2:
2233 326D06                     jbc     ios1_bak,2,chk_swt3
2236 71FB6D                     andb    ios1_bak,#11111011B     ; Clear bit 2
2239 EF4401                     call    swt2_expired
223C                    chk_swt3:
223C 346D03                     jbc     ios1_bak,4,swt_int_done
223F 71F76D                     andb    ios1_bak,#11110111B     ; Clear bit 3
                        ;       call    swt3_expired

2242                    swt_int_done:
2242 F3                         popf
2243 F0                         ret     ; END OF SOFTWARE TIMER INTERRUPT ROUTINE

                        $eject

                        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        ;;;;;;                  SOFTWARE TIMER ROUTINE 0            ;;;;;;;;;;;
                        ;;;;;;                  NOW  USING HSO.0 TO TRIGGER         ;;;;;;;;;;;
                        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

2280                            CSEG AT 2280H

2280                    hso_exec_int:                   ; Check mode -  Update position in mode 2

2280 F2                         PUSHF
2281 B13006                     ldb     HSO_COMMAND,#30H
2284 447C0A04                   add     HSO_TIME,TIMER1,HSO0_dly

2288 91200F                     orb     port1,#00100000B        ; set P1.5
228B A00C28                     ld      Timer_2,TIMER2
228E 390F18                     jbs     Port1,1,in_mode2

2291                    in_mode1:
2291 4866285C                   sub     tmp1,Timer_2,old_t2     ; Check count difference in tmp1
2295 8902005C                   cmp     tmp1,#2
2299 D94C                       jh      end_swt0
229B                    set_mode0:
229B 300F49                     jbc     Port1,0,end_swt0        ; if already in mode 0
229E 71FC0F                     andb    Port1,#11111100B        ; Clear P1.0, P1.1 (set mode 0)
22A1 B15515                     ldb     IOC0,#01010101B         ; enable all HSI
22A4 B0006B                     ldb     last_stat,zero
22A7 203E                       br      end_swt0

22A9                    in_mode2:
22A9 482C283C                   sub     delta_p,timer_2,tmr2_old        ; get timer2 count difference
22AD A0282C                     ld      tmr2_old,timer_2

22B0 306808                     jbc     direct,0,in_rev

```

270061-A2

```
22B3 643C30      in_fwd: add     position,delta_p
22B6 A40032              addc    position+2,zero
22B9 2006                br      chk_mode

22BB 683C30      in_rev: sub     position,delta_p
22BE A80032              subc    position+2,zero

22C1             chk_mode:
22C1 4866285C            sub     tmp1,Timer_2,old_t2      ; Check count difference in tmp1
22C5 8905005C            cmp     tmp1,#5                  ; set model if count is too low
22C9 D21C                jgt     end_swt0                 ; count <= 5

22CB             set_model:
22CB 71FD0F              andb    Port1,#11111101B         ; Clear P1.1, set P1.0 (set mode 1)
22CE 91010F              orb     Port1,#00000001B
22D1 B10515              ldb     IOC0,#00000101B          ; enable HSI 0 and 1
22D4 A00400              ld      zero, HSI_TIME
22D7 48840A56            sub     last1_time,Timer1,min_hsi1
                                         ; set up so (time-last2_time)>min_hsi1 on next HSI
                 $EJECT

22DB             clr_hsi:
22DB A00400              ld      ZERO, HSI_TIME
22DE 717F6D              andb    ios1_bak,#01111111B              ; clear bit 7
22E1 90166D              orb     ios1_bak,ios1
22E4 3F6DF4              jbs     ios1_bak,7,clr_hsi       ; If hsi is triggered then clear hsi

22E7             end_swt0:
22E7 A02866              ld      old_t2,TIMER_2
22EA 71DF0F              andb    port1,#11011111B         ; clear P1.5
22ED F3                  POPF
22EE F0                  ret



                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ;;;;;;                   SOFTWARE TIMER ROUTINE 2                     ;;;;;;;;;;;
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

2380                     CSEG AT 2380H

2380             swt2_expired:
2380 F2                  pushf
2381 B13A06              ldb     hso_command,#3AH         ; set swt_2
2384 44800A04            add     hso_time,timer1,swt2_dly

2388 91040F              orb     port1,#00000100B         ; set port 1.2
238B 89FF075E            cmp     out_ptr,#7ffH
238F D104                bnh     pulsing
2391 A1F0015E            ld      out_ptr,#1f0H
```

270061-A3

```
2395           pulsing:
2395 306E0C             jbc     tr_col,0,swt2_done

2398 C25F32             st      position+2,[out_ptr]+   ;  position high, position low
239B C25F30             st      position,[out_ptr]+

239E C25F68             st      direct,[out_ptr]+
23A1 C25F6C             st      pwm_pwr,[out_ptr]+

                                                        ; store 8 bytes externally

23A4           swt2_done:
23A4 48560A5C           sub     tmp1,timer1,last1_time
23A8 8900185C           cmp     tmp1,#1800H
23AC D104               jnh     swt2_ret        ; keep (Timer1-last1_time)<2000H

23AE 65001056           add     last1_time,#1000H
23B2           swt2_ret:
23B2 71FB0F             andb    port1,#11111011B        ; clear port1.2
23B5 F3                 popf
23B6 F0                 ret

               $EJECT
               ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
               ;;;;;            HSI DATA AVAILABLE INTERRUPT ROUTINE          ;;;;;;;;;;;;;
               ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

               ;  This routine keeps track of the current time and position of the motor.
               ;  The upper word of information is provided by the timer overflow routine.

2400                    CSEG AT 2400H
2400 20CE      now_mode_1:      br      in_mode_1       ; used to save execution time for
2402 20C7      no_int1:         br      no_int          ; worst case loop

2404 F2        hsi_data_int:    pushf
2405 91400F             orb     port1,#01000000B        ; set P1.6
2408 717F6D             andb    ios1_bak,#01111111B     ; Clear ios1_bak.7
240B 90166D             orb     ios1_bak,ios1
240E 376DF1             jbc     ios1_bak,7,no_int1      ; If hsi is not triggered then
                                                        ; jump to no_int
2411           get_values:
2411 A00C28             ld      timer_2,TIMER2
2414 5155066A           andb    hsi_s0,HSI_STATUS,#01010101B
2418 A00440             ld      time, HSI_TIME

241B 380FE2             jbs     port1,0,now_mode_1      ; jump if in mode 1

241E           In_mode_0:
241E 386A0B             jbs     hsi_s0,0,a_rise
```

270061-A4

```
2421 3A6A2C              jbs     hsi_s0,2,a_fall
2424 3C6A4D              jbs     hsi_s0,4,b_rise
2427 3E6A5A              jbs     hsi_s0,6,b_fall
242A 2094                br      no_cnt

242C A05658     a_rise:  ld      last2_time,last1_time
242F A04056              ld      last1_time,time
2432 685840              sub     time,last2_time
2435 888240              cmp     time,min_hsi
2438 D906                jh      tst_statr
                ;set mode1-
243A 91010F              orb     Port1,#00000001B         ; Set P1.0 (in mode 1)
243D B10515              ldb     IOC0,#00000101B          ; Enable HSI 0 and 1
2440            tst_statr:
2440 3E6B5B              jbs     last_stat,6,going_fwd
2443 3C6B67              jbs     last_stat,4,going_rev
2446 3A6B50              jbs     last_stat,2,change_dir
2449 98006B              cmpb    last_stat,zero
244C DF46                je      first_time               ; first time in mode0
244E 27B2                br      no_int1

2450 A05658     a_fall:  ld      last2_time,last1_time
2453 A04056              ld      last1_time,time
2456 685840              sub     time,last2_time
2459 888240              cmp     time,min_hsi
245C D906                jh      tst_statf
                ;set mode1-
245E 91010F              orb     Port1,#00000001B         ; Set P1.0 (in mode 1)
2461 B10515              ldb     IOC0,#00000101B          ; Enable HSI 0 and 1
                $EJECT
2464            tst_statf:
2464 3C6B37              jbs     last_stat,4,going_fwd
2467 3E6B43              jbs     last_stat,6,going_rev
246A 386B2C              jbs     last_stat,0,change_dir
246D 98006B              cmpb    last_stat,zero
2470 DF22                je      first_time               ; first time in mode0
2472 2057                br      no_int

2474 386B27     b_rise:  jbs     last_stat,0,going_fwd
2477 3A6B33              jbs     last_stat,2,going_rev
247A 3E6B1C              jbs     last_stat,6,change_dir
247D 98006B              cmpb    last_stat,zero
2480 DF12                je      first_time               ; first time in mode0
2482 2047                br      no_int

2484 3A6B17     b_fall:  jbs     last_stat,2,going_fwd
2487 386B23              jbs     last_stat,0,going_rev
248A 3C6B0C              jbs     last_stat,4,change_dir
248D 98006B              cmpb    last_stat,zero
2490 DF02                je      first_time               ; first time in mode0
```

270061-A5

```
2492 2037                       br      no_int

2494                first_time:
2494 C46B6A                     stb     hsi_s0,last_stat
2497 2072                       br      done_chk        ; add delta position


2499                change_dir:
2499 1268                       notb    direct
249B 30680F         no_inc:     jbc     direct,0,going_rev

249E                going_fwd:
249E 914010                     orb     PORT2,#01000000B        ; set P2.6
24A1 B10168                     ldb     direct,#01              ; direction = forward
24A4 65010030                   add     position,#01
24A8 A40032                     addc    position+2,zero
24AB 200D                       br      st_stat
24AD                going_rev:
24AD 71BF10                     andb    PORT2,#10111111B        ; clear P2.6
24B0 B10068                     ldb     direct,#00              ; direction = reverse
24B3 69010030                   sub     position,#01
24B7 A80032                     subc    position+2,zero

24BA                st_stat:
24BA C46B6A                     stb     hsi_s0,last_stat
24BD                load_lasts:
24BD A0282C                     ld      tmr2_old,timer_2
24C0 717F6D         no_cnt:     andb    ios1_bak,#01111111B     ; clr bit 7
24C3 90166D                     orb     ios1_bak,ios1
24C6 376D02                     jbc     ios1_bak,7,no_int
24C9 2746           again:      br      get_values

24CB 71BF0F         no_int:     andb    port1,#10111111B        ; Clear P1.6
24CE F3                         popf
24CF F0                         ret     ; end of hsi_data interrupt routine
                                        ; Routine for mode 1 follows and then returns to "load_lasts"
                    $EJECT


24D0                In_mode_1:                  ; mode 1 HSI routine

24D0 51506A5C                   andb    tmp1,hsi_s0,#01010000B
24D4 D7EA                       jne     no_cnt
24D6                cmp_time:                                   ; Procedure which sets mode 1 also
                                                                ; sets times to pass the tests
24D6 A05658                     ld      last2_time,last1_time
24D9 A04056                     ld      last1_time,time

24DC 4858405C       cmp1:       sub     tmp1,time,last2_time
24E0 88845C                     cmp     tmp1,min_hsi1
```

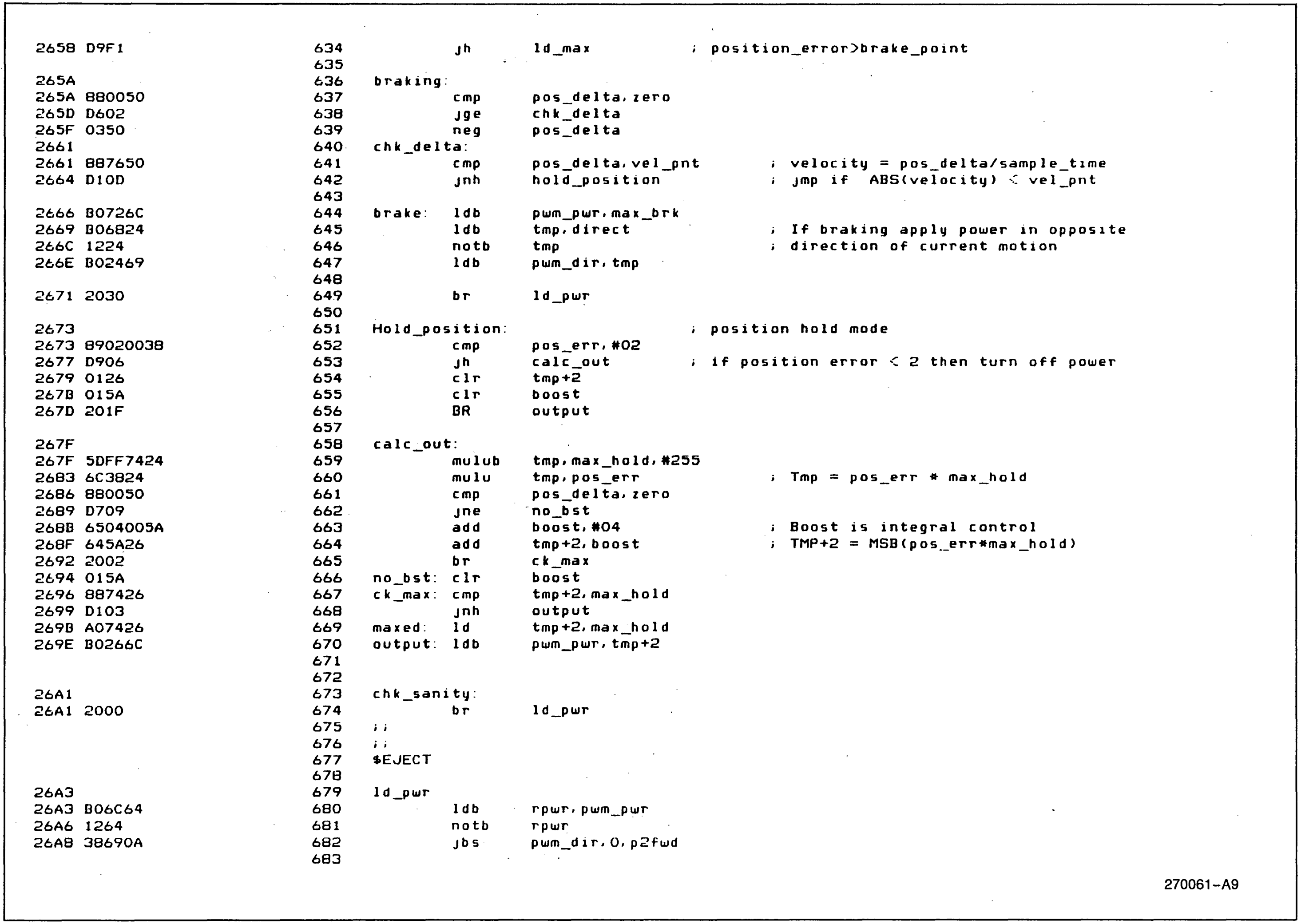
270061-A6

```
24E3 D914                          jh      check_max_time

24E5                       set_mode_2:
24E5 91020F                        orb     Port1,#00000010B        ; Set P1.1 (in mode 2)
24E8 B10015                        ldb     IOCO,#00000000B         ; Disable all HSI
24EB A00400                mt_hsi: ld      zero,hsi_time           ; empty the hsi fifo
24EE 717F6D                        andb    ios1_bak,#01111111B             ; clear bit 7
24F1 90166D                        orb     ios1_bak,ios1
24F4 3F6DF4                        jbs     ios1_bak,7,mt_hsi       ; If hsi is triggered then clear hsi
24F7 2012                          br      done_chk

24F9                       check_max_time:
24F9 4858405C                      sub     tmp1,time,last2_time
24FD 88865C                        cmp     tmp1,max_hsi1           ; max_hsi = addition to min_hsi for
                                                                   ; total time
2500 D109                          jnh     done_chk

2502                       set_mode_0:
2502 71FC0F                        andb    Port1,#11111100B        ; clear P1.0,1  set mode 00
2505 B15515                        ldb     IOCO,#01010101B         ; Enable all HSI
2508 B0006B                        ldb     last_stat,zero

250B                       done_chk:
250B 482C283C                      sub     delta_p,timer_2,tmr2_old        ; get timer2 count difference
250F 306808                        jbc     direct,0,add_rev
2512                       add_fwd:
2512 643C30                        add     position,delta_p
2515 A40032                        addc    position+2,zero
2518 27A3                          br      load_lasts
251A                       add_rev:
251A 683C30                        sub     position,delta_p
251D A80032                        subc    position+2,zero
2520 279B                          br      load_lasts

                           $eject
                           ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                           ;;;;;;                  SOFTWARE TIMER ROUTINE 1               ;;;;;;;;;;;
                           ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

2600                       CSEG AT 2600H

2600                       swt1_expired:

2600 F2                            pushf
2601 91800F                        orb     port1,#10000000B        ; set port1.7

2604 B10D08                        ldb     int_mask,#00001101B     ; enable HSI, Tovf, HSO

2607 B13906                        ldb     HSO_COMMAND,#39H
260A 447E0A04                      add     HSO_TIME,TIMER1,swt1_dly
```

270061-A7

```

260E A0464A                    ld      time_err+2,des_time+2   ; Calculate time & position error
2611 A0363A                    ld      pos_err+2,des_pos+2
2614 48404448                  sub     time_err,des_time,time          ; values are set
2618 A8424A                    subc    time_err+2,time+2
261B 48303438                  sub     pos_err,des_pos,position
261F A8323A                    subc    pos_err+2,position+2

2622 FB                        EI

2623 48484C52                  sub     time_delta,last_time_err,time_err
2627 A0484C                    ld      last_time_err,time_err

262A 48384E50                  sub     pos_delta,last_pos_err,pos_err
262E A0384E                    ld      last_pos_err,pos_err

                      ;;;;;            Time_err = Desired time to finish - current time
                      ;;;;;            Pos_err  = Desired position to finish - current position
                      ;;;;;            Pos_delta  = Last position error - Curent position error
                      ;;;;;            Time_delta = Last time error - Current time error
                      ;;;;;               note that errors should get smaller so deltas will be
                      ;;;;;               positive for forward motion (time is always forward)


2631                  chk_dir:
2631 88003A                    cmp     pos_err+2,zero
2634 D60D                      jge     go_forward

2636                  go_backward:
2636 0338                      neg     pos_err            ; Pos_err = ABS VAL (pos_err)
2638 B10069                    ldb     pwm_dir,#00h
263B 89FFFF3A                  cmp     pos_err+2,#0ffffH
263F D70A                      jne     ld_max
2641 200D                      br      chk_brk

2643                  go_forward:
2643 B10169                    ldb     pwm_dir,#01H
2646 88003A                    cmp     pos_err+2,zero
2649 DF05                      je      chk_brk
                      $EJECT

264B B0706C           ld_max:  ldb     pwm_pwr,max_pwr
264E 2051                      br      chk_sanity

2650                  Chk_brk:                            ;  Position_Error now = ABS(pos_err)
2650 887A38                    cmp     pos_err,pos_pnt
2653 D11E                      jnh     hold_position      ; position_error<position_control_point
2655 887838                    cmp     pos_err,brk_pnt
```

270061-A8



```
2658 D9F1                    jh      ld_max            ; position_error>brake_point

265A                braking:
265A 880050                  cmp     pos_delta,zero
265D D602                    jge     chk_delta
265F 0350                    neg     pos_delta
2661                chk_delta:
2661 887650                  cmp     pos_delta,vel_pnt         ; velocity = pos_delta/sample_time
2664 D10D                    jnh     hold_position             ; jmp if  ABS(velocity) < vel_pnt

2666 B0726C         brake:   ldb     pwm_pwr,max_brk
2669 B06824                  ldb     tmp,direct                ; If braking apply power in opposite
266C 1224                    notb    tmp                       ; direction of current motion
266E B02469                  ldb     pwm_dir,tmp

2671 2030                    br      ld_pwr

2673                Hold_position:                     ; position hold mode
2673 89020038                cmp     pos_err,#02
2677 D906                    jh      calc_out          ; if position error < 2 then turn off power
2679 0126                    clr     tmp+2
267B 015A                    clr     boost
267D 201F                    BR      output

267F                calc_out:
267F 5DFF7424                mulub   tmp,max_hold,#255
2683 6C3824                  mulu    tmp,pos_err               ; Tmp = pos_err * max_hold
2686 880050                  cmp     pos_delta,zero
2689 D709                    jne     no_bst
268B 6504005A                add     boost,#04                 ; Boost is integral control
268F 645A26                  add     tmp+2,boost               ; TMP+2 = MSB(pos_err*max_hold)
2692 2002                    br      ck_max
2694 015A           no_bst:  clr     boost
2696 887426         ck_max:  cmp     tmp+2,max_hold
2699 D103                    jnh     output
269B A07426         maxed:   ld      tmp+2,max_hold
269E B0266C         output:  ldb     pwm_pwr,tmp+2


26A1                chk_sanity:
26A1 2000                    br      ld_pwr
                    ;;
                    ;;
                    $EJECT

26A3                ld_pwr
26A3 B06C64                  ldb     rpwr,pwm_pwr
26A6 1264                    notb    rpwr
26A8 38690A                  jbs     pwm_dir,0,p2fwd

```

270061-A9

```
26AB FA              p2bkwd: DI
26AC 717F10                  andb    port2,#01111111B        ; clear P2.7
26AF B06417                  ldb     pwm_control,rpwr
26B2 FB                      EI
26B3 2008                    br      pwrset
26B5 FA              p2fwd:  DI
26B6 918010                  orb     port2,#10000000B        ; set P2.7
26B9 B06417                  ldb     pwm_control,rpwr
26BC FB                      EI

26BD                 pwrset:
26BD 88004A                  cmp     time_err+2,zero  ; do pos_table when err is negative
26C0 D225                    jgt     end_p
                     ;;;     br      end_p

26C2 89202962                cmp     nxt_pos,#(32+pos_table)
26C6 DE06                    jlt     get_vals                ; jump if lower
26C8 A1002962                ld      nxt_pos,#pos_table
26CC 0142                    clr     time+2
26CE                 get_vals:

26CE A26334                  ld      des_pos,[nxt_pos]+
26D1 A26336                  ld      des_pos+2,[nxt_pos]+
26D4 A26346                  ld      des_time+2,[nxt_pos]+
26D7 A26370                  ld      max_pwr,[nxt_pos]+
26DA A07072                  ld      max_brk,max_pwr
26DD 646034                  add     des_pos,offset
26E0 A40036                  addc    des_pos+2,zero
26E3 4830344E                sub     last_pos_err,des_pos,position

26E7 717F0F          end_p:  andb    port1,#01111111B        ; clear P1.7

26EA F3                      popf
26EB F0                      ret

                     $EJECT

                     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;       main  program          ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


2800                         CSEG at 2800H

2800                 MAIN_PROG:
2800 90166D                  orb     ios1_bak,ios1
2803 366D09                  jbc     ios1_bak,6,control
2806 71BF6D                  andb    ios1_bak,#10111111B     ; clear ios1_bak.6
2809 95100F                  xorb    Port1,#00010000B        ; Compl Bit P1.4
280C EFF5FB                  call    HSI_DATA_INT            ; prevent lockup
```

270061-B0

```
280F                   control:
280F 912D08                    orb     int_mask,#00101101B     ; enable hsi, hso, swt, tovf interrupts
2812 FD                        nop
2813 FD                        nop
2814 FD                        nop
2815 E06FFD                    djnz    main_dly,$
2818 FD                        nop
2819 95080F                    xorb    port1,#00001000B        ; compliment p1.3
281C 27E2                      BR      MAIN_PROG


2900                           CSEG AT 2900H

2900                   pos_table:

2900 00000000                  dcl     00000000H       ; position 0
2904 20008000                  dcw     0020H, 0080H    ; next time, power
2908 00C00000                  dcl     0000c000H       ; position 1
290C 40004000                  dcw     0040H, 0040H    ; next time, power
2910 00000000                  dcl     00000000H       ; position 2
2914 6000C000                  dcw     0060H, 00c0H    ; next time, power
2918 0080FFFF                  dcl     0FFFF8000H      ; position 3
291C 80008000                  dcw     0080H, 0080H    ; next time, power

2920 00080000                  dcl     00000800H       ; position 4
2924 58008000                  dcw     0058H, 0080H    ; next time, power
2928 00300000                  dcl     00003000H       ; position 5
292C 7000FF00                  dcw     0070H, 00ffH    ; next time, power
2930 00000000                  dcl     00000000H       ; position 6
2934 9000F000                  dcw     0090H, 00f0H    ; next time, power
2938 00000000                  dcl     00000000H       ; position 7
293C 9100F000                  dcw     0091H, 00f0H    ; next time, power


2940                           END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-B1

APPLICATION NOTE

AP-275

April 1987

An FFT Algorithm For MCS®-96 Products Including Supporting Routines and Examples

IRA HORDEN
MCO APPLICATIONS ENGINEER

Order Number: 270189-001

1.0 INTRODUCTION

Intel's 8096 is a 16-bit microcontroller with processing power sufficient to perform many tasks which were previously done by microprocessors or special building block computers. A new field of applications is opened by having this much power available on a single chip controller.

The 8096 can be used to increase the performance of existing designs based on 8051s or similar 8-bit controllers. In addition, it can be used for Digital Signal Processing (DSP) applications, as well as matrix manipulations and other processing oriented tasks. One of the tasks that can be performed is the calculation of a Fast Fourier Transform (FFT). The algorithm used is similar to that in many DSP and matrix manipulation applications, so while it is directly applicable to a specific set of applications, it is indirectly applicable to many more.

FFTs are most often used in determining what frequencies are present in an analog signal. By providing a tool to identify specific waveforms by their frequency components, FFTs can be used to compare signals to one another or to set patterns. This type of procedure is used in speech detection and engine knock sensors. FFTs also have uses in vision systems where they identify objects by comparing their outlines, and in radar units to detect the dopler shift created by moving objects.

This application note discusses how FFTs can be calculated using Intel's MCS®-96 microcontrollers. A review of fourier analysis is presented, along with the specific code required for a 64 point real FFT. Throughout this application note, it is assumed that the reader has a working knowledge of the 8096. For those without this background the following two publications will be helpful:

1986 Microcontroller Handbook
Using the 8096, AP-248

These books are listed in the bibliography, along with other good sources of information on the MCS-96 product family and on Fast Fourier Transforms.

2.0 PROGRAM OVERVIEW

This application note contains program modules which are combined to create a program which performs an FFT on an analog signal sampled by the on-board ADC (Analog to Digital Converter) of the 8097. The results of the FFT are then provided over the serial channel to a printer or terminal which displays the results. In the applications listed in the previous section, the data from this FFT program would be used directly by another program instead of being plotted. However, the plotted results are used here to provide an example of what the FFT does. There are four program modules discussed in this application note:

FFTRUN - Runs a 64 point FFT on its data buffer. It produces 32 14-bit complex output values and 32 14-bit output magnitudes. A fast square root routine and log conversion routine are included.

A2DCON - Fills one of two buffers with analog values at a set sample rate. The sample time can be as fast as 50 microseconds using 8x9xBH components.

PLOTSP - Plots the contents of a buffer to a serially connected printer. Routines are provided for console out and hexadecimal to decimal conversion and printing.

FTMAIN - The main module which controls the other modules.

Each of the modules will be described separately. In order to better understand how the programs work together, a brief tutorial on FFTs will be presented first, followed by descriptions of the programs in the order listed above.

The final program uses 64 real data points, taken from either a table or analog input 1. Each of the data points is a 16-bit signed number. The processing takes 12.5 milliseconds when internal RAM is used as the data space. If external RAM is used, 14 milliseconds are required. Larger FFTs can be performed by slightly modifying the programs. A 256-point FFT would take approximately 65 milliseconds, and a 1024-point version would require about 300 milliseconds.

In the program presented, the analog sampling time is set for 1 sample every 100 microseconds, providing the 64 samples in 6.4 milliseconds. The sampling time can be reduced to around 60 microseconds per point by changing a variable, and less than 50 microseconds by using the 8x9xBH series of parts, since they have a 22 microsecond A to D conversion time.

The programs are set up to be run in a sequence instead of concurrently. This provides the fastest operation if the sampling speed were reduced to the minimum possible. For the fastest operation above about 80 microseconds a sample, the programs could be run concurrently, but this would require some minor modifications of the program. Figure 1 shows the timing of the program as presented.

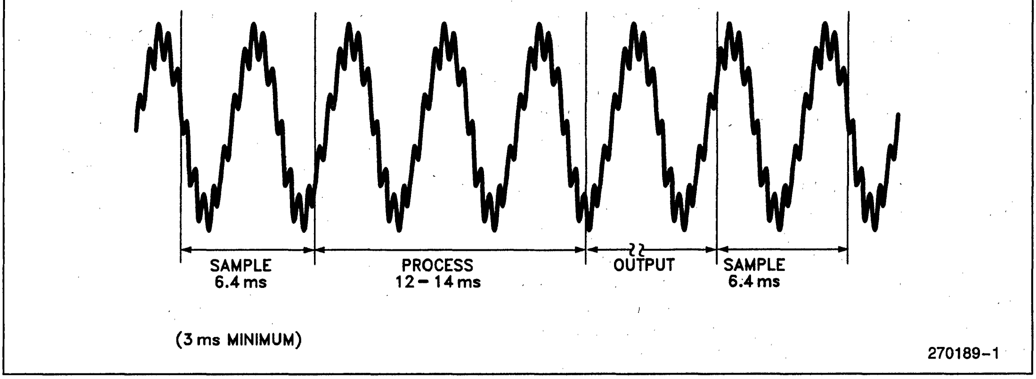


Figure 1. Timing of the FFT Program

These programs have run in the Intel Microcontroller Operation Application's Lab and produced the results presented in this application note. Since the programs have not undergone any further testing, we cannot guarantee them to be bug proof. We, therefore, recommend that they be thoroughly tested before being used for other than demonstration purposes.

3.0 FOURIER TRANSFORMS

A Fourier Transform is a useful analytical tool that is frequently ignored due to its mathematically oriented derivations. This is unfortunate, since Fourier transforms can be used without fully understanding the mathematics behind them. Of course, if one understands the theory behind these transforms, they become much more powerful.

The majority of this application note deals with how a Fast Fourier Transform (FFT) can be used for spectrum analysis. This procedure takes an input signal and separates it into its frequency components. One can almost treat the FFT as a black box, which has as its output, the frequency components and magnitudes of the input signal, much like a spectrum analyzer.

From a mathematical standpoint, Fourier Transforms change information in the time domain into the frequency domain. The theory behind the Fourier transform stems from Fourier analysis, also called frequency analysis.

There are many books on the topic of Fourier analysis, several of which are listed in the bibliography. In this application note, only the pertinent formulas and uses will be presented, not their derivations.

The main idea in Fourier analysis is that a function can be expressed as a summation of sinusoidal functions of different frequencies, phase angles, and magnitudes. This idea is represented by the Fourier Integral:

$$H(f) = \int_{-\infty}^{\infty} h(t)\, e^{-j2\pi ft} \tag{1}$$

Where: H(f) is a function of frequency
h(t) is a function of time

Since

$$e^{-j\theta} = \mathrm{COS}\ \theta - j\ \mathrm{SIN}\theta \tag{2}$$

$$H(f) = \int_{-\infty}^{\infty} h(t)\,(\cos(2\pi ft) - j \sin(2\pi ft))\, dt \tag{3}$$

Figure 2 shows a rectangular pulse and its Fourier transform. Note that the results in the frequency domain are continuous rather than discrete.

In a simplified case, the varying phase angles can be removed, and the integral changed to a summation, known as a Fourier Series. All periodic functions can be described in this way. This series, as shown below, can help provide a more graphical understanding of Fourier analysis.

$$y(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(2\pi nf_0t) + b_n \sin(2\pi nf_0t)] \tag{4}$$

for n = 1 to ∞

Where $f_0 = \frac{1}{T_0}$, the fundamental frequency.

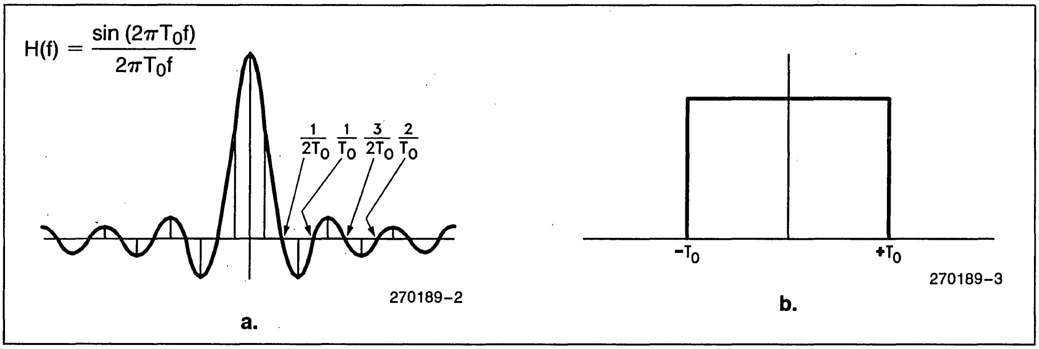


Figure 2. Rectangular Pulse and Its Fourier Transform

This formula can also be represented in complex form as:

$$\sum_{n=-\infty}^{\infty} \alpha n \, e^{j\pi n f_0 t} \qquad (5)$$

The Fourier series for a square wave is

$$\sum_{K=0}^{\infty} \frac{\sin((2k + 1) 2\pi f_0 t)}{(2k + 1)} \qquad (6)$$

If these sinusoids are summed, a square wave will be formed. Figure 3 shows the graphical summation of the first 3 terms of the series. Since the higher frequencies contribute to the squareness of the waveform at the corners, it is reasonable to compare only the flatness of the top of the waveform. The sharpness or risetime of the waveform can be determined by the highest frequency term being summed. With rise and fall times of 10% of the period, the waveform generated by the first 3 terms is within 20% of ideal. At 7 terms it is within 10%, and at 20 terms it is within 5%. With a 5% risetime, it is within 20% of ideal after 5 terms, 10% after 13 terms and 5% after 32 terms. Figure 4 shows the resultant waveforms after the summation of 7, 15 and 30 terms.

Fourier analysis can be used on equation 4 to find the coefficients a_n and b_n. To make this process easier to use with a computer, a discrete form, rather than a continuous one, must be used. The discrete Fourier transform, shown in Equation 7, is a good approximation to the continuous version. The closeness of the approximation depends on several conditions which will be discussed later. The input to this transform is a set of N equally spaced samples of a waveform taken over a period of NT. The period NT is frequently referred to as the "Sampling Window".

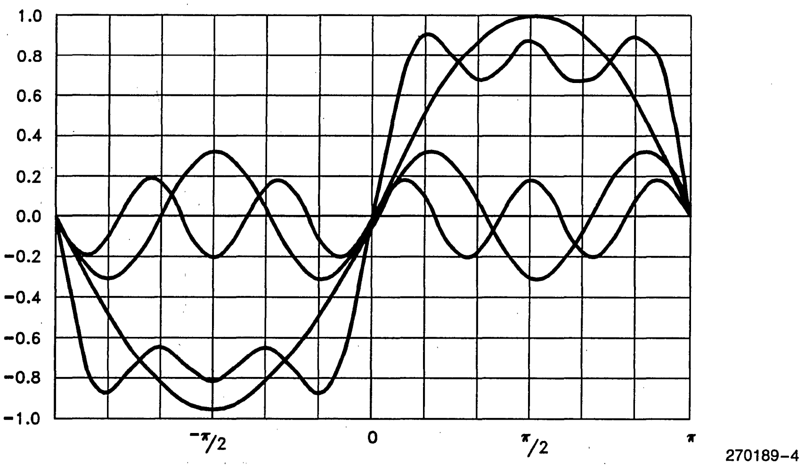
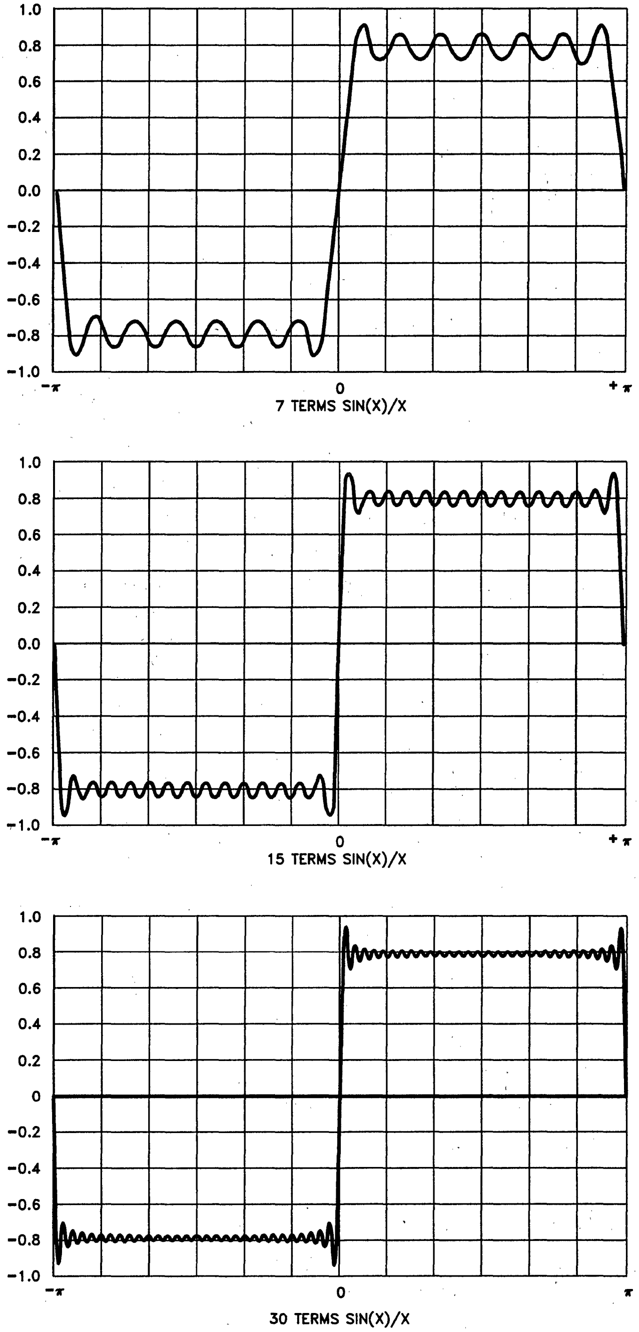


Figure 3. Graphical Summation of Sinewaves



7 TERMS SIN(X)/X

270189–5

15 TERMS SIN(X)/X

270189–6

30 TERMS SIN(X)/X

270189–7

Figure 4. Square Wave from Sinusoids

$$H\left(\frac{n}{NT}\right) = \sum_{k=0}^{N-1} h(kT)e^{-j2\pi nk/N}$$

$$n = 0, 1, \ldots, N-1 \qquad (7)$$

Where: H(f) is a function of frequency

h(t) is a function of time

T is the time span between samples

N is the number of samples in the window

n =0,1,2 ... N-1

This transform is used for many applications, including Fourier Harmonic Analysis. This procedure uses the transform to calculate the coefficients used in Equation 5. In order to do this, the factor T/NT must be added to the transform as follows:

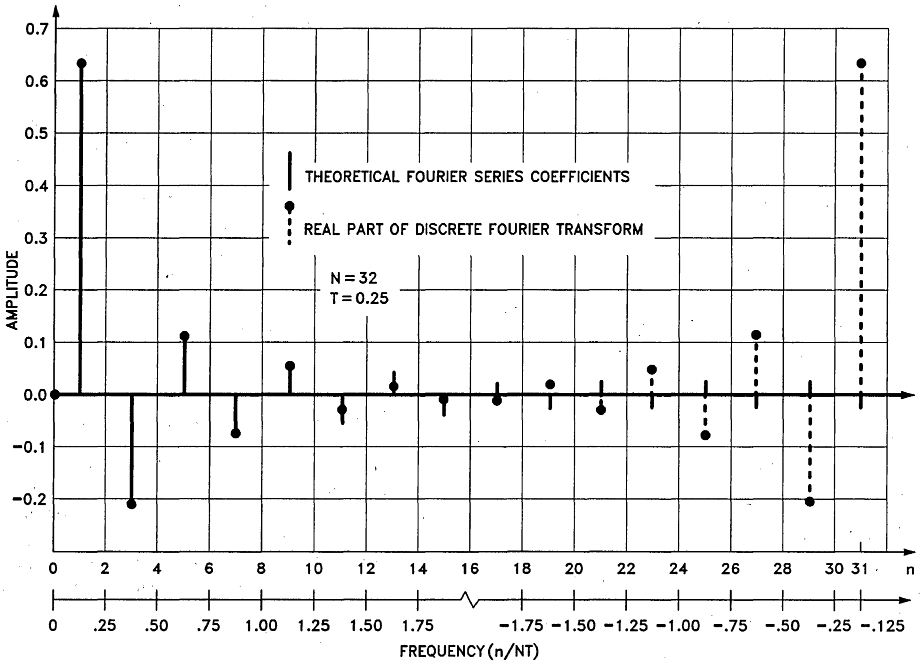
$$H\left(\frac{n}{NT}\right) = \frac{T}{(NT)} \sum_{k=0}^{N-1} h(kT)\, e^{-j2\pi nk/N}$$

$$n=0, 1, 2, 3, \ldots, N-1 \qquad (8)$$

The factor provides compensation for the number of samples taken. Note that the functions H(f) and h(t) are complex variables, so the simplicity of the equation can be misleading. Once the values of h(t) are known, (ie. the value of the input at the discrete times (t)), the Fourier Transform can be used to find the magnitude and phase shift of the signal at the frequencies (f).

A spectrum analyzer can provide similar information on an analog input signal by using analog filters to separate the frequency components. Regardless of its source, the information on component frequencies of a signal can be used to detect specific frequencies present in a signal or to compare one signal to another. Many lab experiments and product development tests can make use of this type of information. Using these methods, the purity of signals can be measured, specific harmonics can be detected in mechanical equipment, and noise bursts can be classified. All of this information can be obtained while still treating the FFT process as a black box.

Consider the discrete transform of a square wave as shown in Figure 5. Note that the component magnitudes, as shown in the series of Equation 6, are shown in a mirrored form in the transform. This will happen whenever only real data is used as the FFT input, if both real and imaginary data were used the output would not be guaranteed to be symmetrical. For this reason, there is duplicate information in the transform for many applications. Later in this section a method to make the most of this characteristic is discussed.



270189-8

Figure 5. Discrete Transform of a Square Wave

If one looks at Equation 8, it can be seen that the calculation of a discrete Fourier transform requires N squared complex multiplications. If N is large, the calculation time can easily become unrealistic for real-time applications. For example, if a complex multiplication takes 40 microseconds, at N = 16, 10 milliseconds would be used for calculation, while at N = 128, over half a second would be needed. A Fast Fourier Transform is an algorithm which uses less multiplications, and is therefore faster. To calculate the actual time savings, it is first necessary to understand how a FFT works.

4.0 THE FFT ALGORITHM

The FFT algorithm makes use of the periodic nature of waveforms and some matrix algebra tricks to reduce the number of calculations needed for a transform. A more complete discussion of this is in Appendix A, however, the areas that need to be understood to follow the algorithm are presented here. This information need not be read if the reader's intent is to use the program and not to understand the mathematical process of the algorithm

To simplify notation the following substitutions are made in Equation 8.

$$W = e^{-j2\pi/N}$$

$$k = kT$$

$$n = \frac{n}{NT}$$

The resultant equation being

$$x(n) = \sum_{k=0}^{N-1} n(k)W^{nk} \qquad (9)$$

Expressed as a matrix operation

$$\begin{bmatrix} X(1) \\ X(2) \\ X(3) \\ \vdots \\ X(N\text{-}1) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & \dots & W^0 \\ W^0 & W^1 & W^2 & \dots & W^N \\ W^0 & W^2 & W^4 & \dots & W^{2N} \\ \vdots & \vdots & \vdots & & \vdots \\ W^0 & W^{(N-1)} & W^{2(N-1)} & \dots & W^{(N-1)^2} \end{bmatrix} \begin{bmatrix} X_0(0) \\ X_0(1) \\ X_0(2) \\ \vdots \\ X_0(N-1) \end{bmatrix}$$

A brief review of matrix properties can be found in Appendix A. Because of the periodic nature of W the following is true:

$$W^{nk}\,\mathrm{MOD}\,N = W^{nk} \qquad (10)$$

$$= \mathrm{COS}\,(2\pi\, nk/N) - j\,\mathrm{SIN}\,(2\pi nk/N)$$

$W^0 = 1$ therefore, if $nk\ \mathrm{MOD}\ N = 0$, $W^{nk} = 1$

This reduces the calculations as several of the W terms go to 1 and the highest power of W is N. All of W values are complex, so most of the operations will have to be complex operations. We will continue to use only the W, X(n) and X0(k) symbols to represent these complex quantities.

The FFT algorithm we will use requires that N be an integral power of 2. Other FFT algorithms do not have this restriction, but they are more complex to understand and develop. Additionally, for the relatively small values of N we are using this restriction should not provide much of a problem. We will define EXPONENT as log base 2 of N. Therefore,

$$N = 2^{\mathrm{EXPONENT}}$$

The magic of the FFT, (as detailed in Appendix A), involves factoring the matrix into EXPONENT matrices, each of which has all zeros except for a 1 and a W^{nk} term in each row. When these matrices are multiplied together the result is the same as that of the multiplication indicated in Equation 9, except that the rows are interchanged and there are fewer non-trivial multiplications. To reorder the rows, and thus make the information useful, it is necessary to perform a procedure called "Bit Reversal".

This process requires that N first be converted to a binary number. The least significant bit (lsb) is swapped with the most significant bit (msb). Then the next lsb is swapped with the next msb, and so on until all bits have been swapped once. For N = 8, 3 bits are used, and the values for N and their bit reversals are shown below:

Number	Binary	Bit Reversal	Decimal BR
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Recall that the FFT of real data provides a mirrored image output, but the FFT algorithm can accept inputs with both real and imaginary components. Since the inputs for harmonic analysis provided by a single A to D are real, the FFT algorithm is doing a lot of calculations with one input term equal to zero. This is obviously not very efficient. More information for a given size transform can be obtained by using a few more tricks.

It is possible to perform the FFT of two real functions at the same time by using the imaginary input values to the FFT for the second real function. There is then a post processing performed on the FFT results which separate the FFTs of the two functions. Using a similar procedure one can perform a transform on 2N real samples using an N complex sample transform.

The procedure involves alternating the real sample values between the real and imaginary inputs to the FFT. If, as in our example, the input to the FFT is a 2 by 32 array containing the complex values for 32 inputs, the 64 real samples would be loaded into it as follows:

N	00 01 02 03 04 05 06 07 ..... 30 31
REAL	00 02 04 06 08 10 12 14 ..... 60 62
IMAGINARY	01 03 05 07 09 11 13 15 ..... 61 63

This procedure is referred to as a pre-weave. In order to derive the desired results, the FFT is run, and then a post-weave operation is performed. The formula for the post-weave is shown below:

$$X_r(n) = \left[\frac{R(n)}{2} + \frac{R(N-n)}{2}\right] + \cos\frac{\pi n}{N}\left[\frac{I(N)}{2} + \frac{I(N-n)}{2}\right] -$$

$$\sin\frac{\pi n}{N}\left[\frac{R(n)}{2} - \frac{R(N-n)}{2}\right] \quad n = 0, 1, \ldots, N-1$$

$$X_i(n) = \left[\frac{I(n)}{2} - \frac{I(N-n)}{2}\right] - \sin\frac{\pi n}{N}\left[\frac{I(n)}{2} + \frac{I(N-n)}{2}\right] -$$

$$\cos\frac{\pi n}{N}\left[\frac{R(n)}{2} - \frac{R(N-n)}{2}\right] \quad n = 0, 1, \ldots, N-1 \qquad (11)$$

Where R(n) is the real FFT output value

I(n) is the imaginary FFT output value

Xr(n) is the real post-weave output

Xi(n) is the imaginary post-weave output

Note that the output is now one-sided instead of mirrored around the center frequency as it is in Figure 5. The magnitude of the signal at each frequency is calculated by taking the square root of the sum of the squares. The magnitude can now be plotted against frequency, where the frequency steps are defined as:

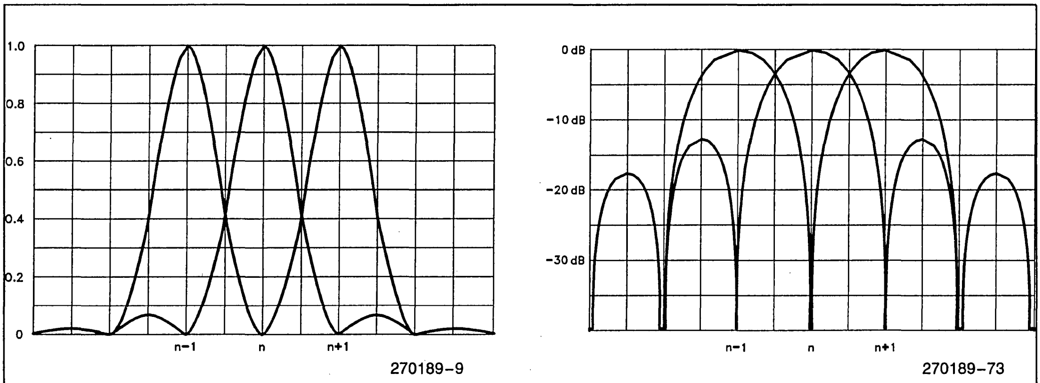
$$\frac{n}{NT} \quad n = 0, 1, 2, 3, \ldots, N-1$$

Where N is the number of complex samples (ie. 32 in this case) T is the time between samples

A value of zero on the frequency scale corresponds to the DC component of the waveform. Most signal analysis is done using Decibels (dB), the conversion is dB = 10 LOG (Magnitude squared). Decibels are not used as an absolute measure, instead signals are compared by the difference in decibels. If the ratio between two signals is 1:2 then there will be a 3 dB difference in their power.

5.0 USING THE FFT

There are several things to be aware of when using FFTs, but with the proper cautions, the FFT output can be used just like that of a spectrum analyzer. The



(a.) Relative Power of Windows (Side Lobes of Side Bins Removed for Clarity).

(b.) 10 Log Relative Power of Windows (Side Lobes of Side Bins Removed for Clarity).

Figure 6. Bin Windows

first precaution is that the FFT is a discrete approximation to a continuous Fourier Transform, so the output will seldom fit the theoretical values exactly, but it will be very close.

Since the programs in this application note generate a one-sided transform with N = 32, the frequency granularity is fairly course. Each of the frequency components output from the FFT is actually the sum of all energy within a narrow band centered on that frequency. This band of sensitivity is referred to as a "bin". The reported magnitude is the actual magnitude multiplied by the value of the bin window at the actual frequency. Figure 6 shows several bin windows. Note that these windows overlap, so that a frequency midway between the two center frequencies will be reported as energy split between both windows. Be careful not to confuse the *sampling window* NT with *bin windows* or with the *windowing function.*

Another area of caution is the relationship of the sampling window to the frequency of the waveform. For the best accuracy, the window should cover an exact multiple of the period of the waveform being analyzed. If it covers less than one period, the results will be invalid. Other variations from ideal will not produce invalid results, just additional noise in the output.

If the sampling window does not cover an exact multiple of all of the frequency components of a waveform, the FFT results will be noisy. The reason for this is the sharp edge that the FFT sees when the edges of the window cut off the input waveform. Figure 7 shows a waveform that is an exact multiple of the window and

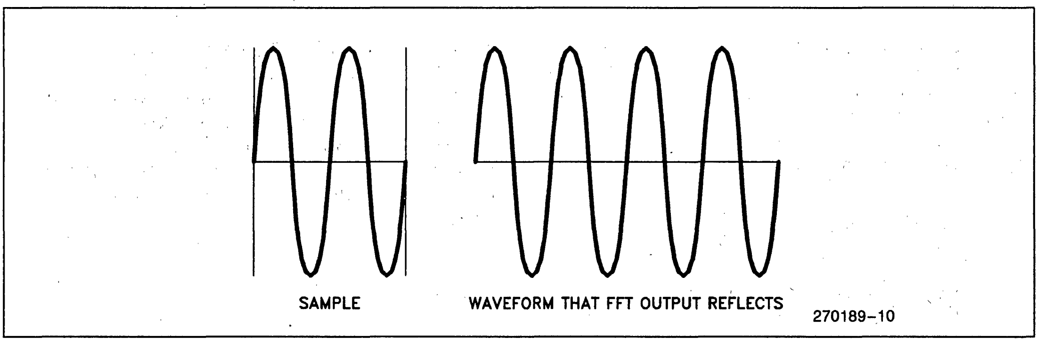


Figure 7. Waveform is a Multiple of the Window

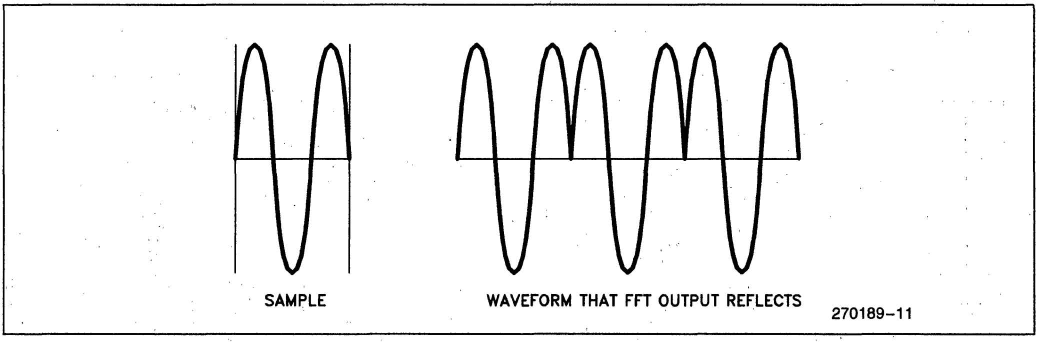
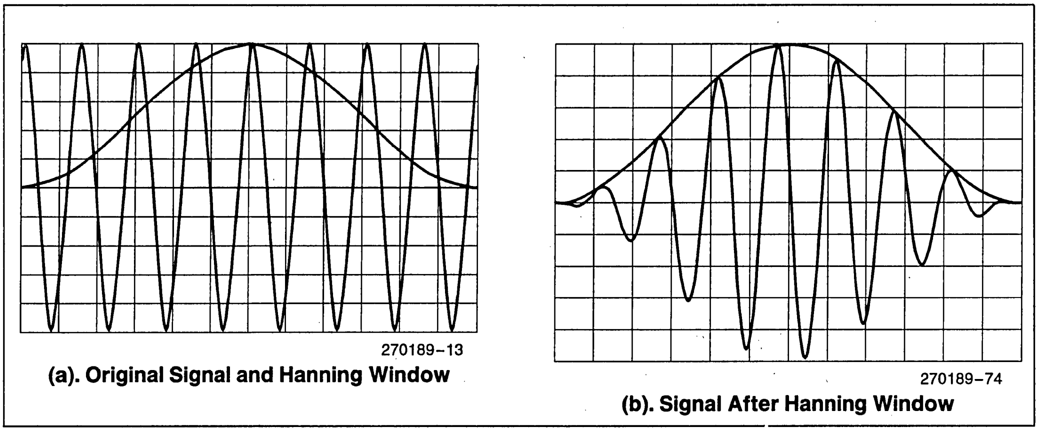


Figure 8. Waveform is Not a Multiple of the Window

the periodic waveform that the FFT output reflects. In Figure 8, the waveform is not a multiple of the window and the waveform that the FFT output reflects has discontinuities. These discontinuities contribute to the noise in an FFT output. This noise is called "spectral leakage", or simply "leakage", since it is leakage between one frequency spectrum and another which is caused by digitization of an analog process.

To reduce this leakage, a process called windowing is used. In this procedure the input data is multiplied by specific values before being used in the FFT. The term "windowing" is used because these values act as a window through which the input data passes. If the input window goes smoothly to zero at both endpoints of the sampling window, there can be no discontinuities. Figure 9 shows a Hanning window and its effect on the input to an FFT. The Hanning window was named after its creator, Julius Von Hann, and is one of the most commonly used windows. More information on windowing and the types of windows can be found in the paper by Harris listed in the bibliography. As expected, the results of the FFT are changed because of the input windowing, but it is in a very predictable way.

Using the Hanning window results in bin windows which are wider and lower in magnitude than normal, as can be seen by comparing Figure 6 with Figure 10. For an input frequency which is equal to the center frequency of a window, the attenuation will be 6 dB on the center frequency. Since the bin windows are wider



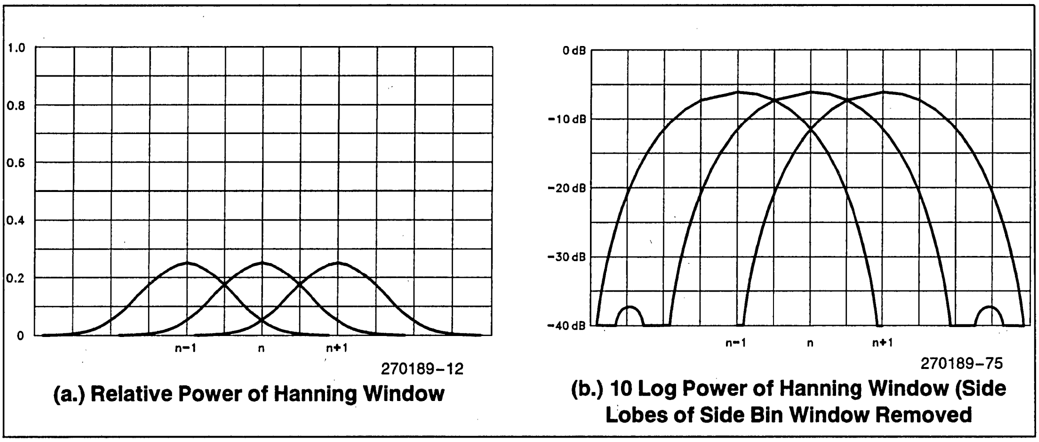
270189-13

(a). Original Signal and Hanning Window

270189-74

(b). Signal After Hanning Window

Figure 9. Effect of Hanning Window on FFT Input



270189-12

(a.) Relative Power of Hanning Window

270189-75

(b.) 10 Log Power of Hanning Window (Side Lobes of Side Bin Window Removed

Figure 10. Bin Windows after Using Hanning Input Window

than normal, the input frequency will also have energy which falls into the bins on either side of center. These side bins will show a reading of 6 dB below the center window. The disadvantage of this spreading is far less than the advantage of removing leakage from the FFT output.

A set of FFT output plots are included in the Appendix. These plots show the effect of windowing on various signals. There are examples of all of the cases described above. A brief discussion of the plots is also presented.

Applications which can make use of this frequency magnitude information include a wide range of signal processing and detection tasks. Many of these tasks use digital filtering and signature analysis to match signals to a standard. This technique has been applied to anti-knock sensors for automobile engines, object identification for vision systems, cardiac arrhythmia detectors, noise separation and many other applications. The ability to do this on a single-chip computer opens a door to new products which would have not been possible or cost effective previously.

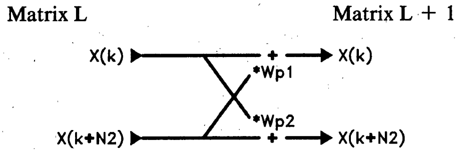
The next four sections of this application note cover the operation of the programs on a line by line basis. Section 6 shows an implementation of the FFT algorithm in BASIC. This code is used as a template to write the ASM96 code in Section 7. Sections 8, 9, and 10 cover the code sections which support the FFT module. After all of the code sections are discussed, an overview of how to use the program is presented in Section 11.

6.0 BASIC PROGRAM FOR FFTS

The algorithm for this FFT is shown in the flowchart in Figure 11 and the BASIC program in Listing 1. There are four sections to this program: initialization, pre-weaving, transform calculation, and post-weaving. The flowchart is generalized, however, the BASIC program has been optimized for assembly language conversion with 64 real samples.

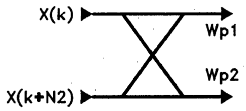
On the flowchart, the initialization and pre-weaving sections are incorporated as "Read in Data". The data to be read includes the raw data as well as the size of the array and the scaling factor. The details for pre-weaving have been discussed earlier, and initialization varies from computer to computer. LOOP COUNT keeps track of which of the factored matrices are being multiplied. SHIFT is the shift count which is used to determine the power of W (as defined earlier) which will be used in the loop.

For each loop N calculations are performed in sets of two. Each calculation set is referred to as a butterfly and has the following form:



270189-15

Also Shown as:



270189-16

OR

$$X1(k) = X_0(k+N2)*Wp1 + X_0(k)$$

$$X1(k+N2) = X_0(k)*Wp2 + X_0(k+N2)$$

In general, the W factors are not the same. However, for the case of this FFT algorithm, Wp1 will always equal (−Wp2). This is because of the way in which "p" is calculated, and the fact that W(x) is a sinusoidal function.

The inner loop in the flowchart is performed N2 times. For LOOP=1, N2=N/2 and if INCNT=N2 then k=N2 and k+N2=N, so the first loop is done and parameters LOOP, N2, and SHIFT are updated. For the first loop, all N/2 sets of calculations are performed contiguously. As LOOP increases, the number of contiguous calculations are cut in half, until LOOP=EXPONENT.

When LOOP=EXPONENT, N2=1, the butterfly is then performed on adjacent variables. Figure 12 shows the butterfly arrangement for a calculation where N=8, so that EXPONENT=3.

The BASIC program follows this flowchart, but operations have been grouped to make it easier to convert it to assembly language. Also not shown in the flowchart are several divide by 2 operations. There are five in the main section, one per loop. These provide the T/NT factor in equation 8 for N=32 ($2^5=32$). There is also an extra divide by two in the post-weave section. It is required to prevent overflows when performing the 16-bit signed arithmetic in the ASM96 program. As a result of these operations, the input scale factor is ±1 = ±32767 and the output scaling is ±1 = ±16384. Note, the maximum input values are ±0.99997.

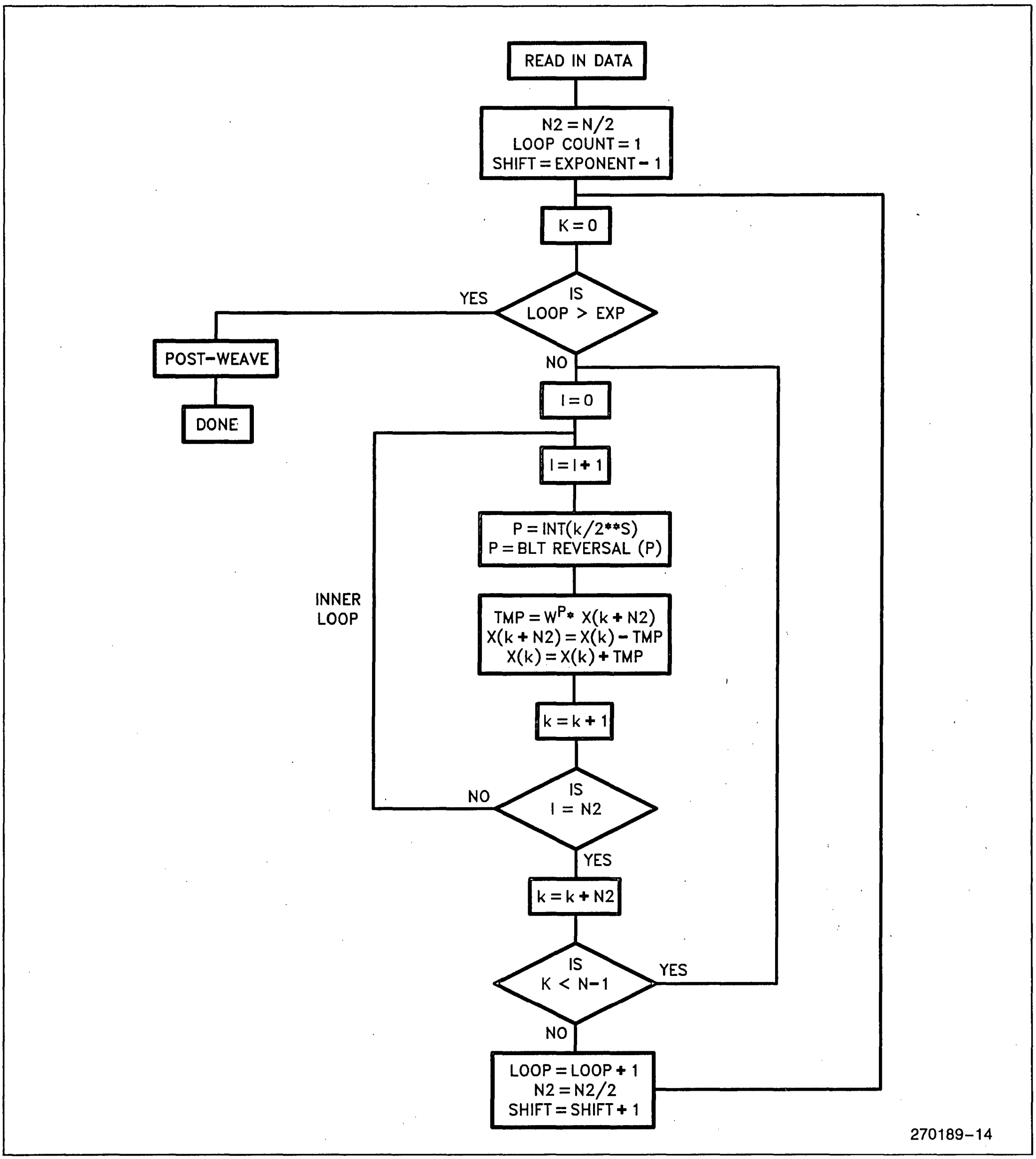
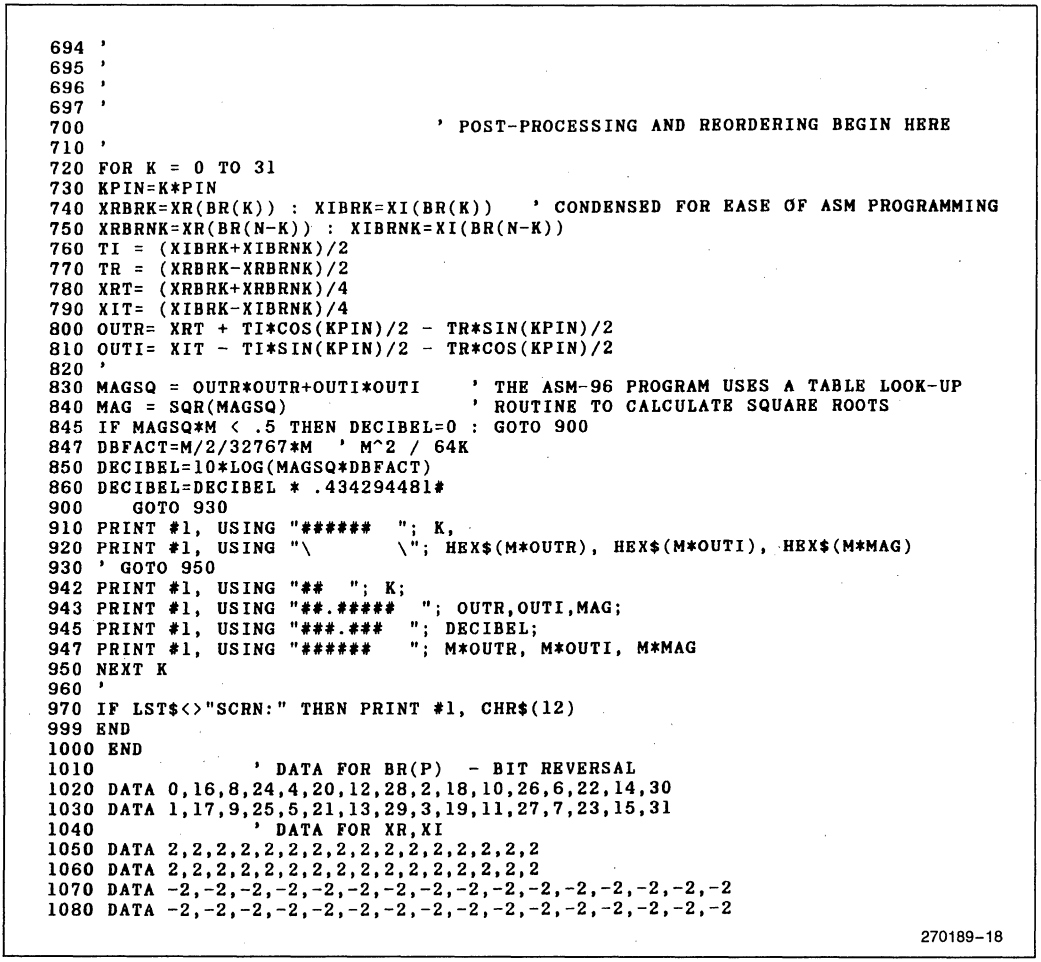


Figure 11. Flowchart of Basic Program

```
100 '  THIS IS FFT13, FEBRUARY 4, 1986
105 '
110    ' COPYRIGHT INTEL CORPORATION, 1985
115    ' BY IRA HORDEN, MCO APPLICATIONS
120 '
125 '   THIS PROGRAM PERFORMS A FAST FOURIER TRANSFORM ON 64 REAL DATA POINTS
130 ' USING A 2N-POINTS WITH AN N-POINT TRANSFORM ALGORITHM.  THE FIRST
135 ' SECTION OF THE PROGRAM PERFORMS A STANDARD TRANSFORM ON DATA THAT HAS
140 ' BEEN INTERLEAVED BETWEEN THE REAL AND IMAGINARY INPUT VALUES.  THE
145 ' RESULTS OF THAT TRANSFORM ARE THEN POST-PROCESSED IN THE SECOND SECTION
150 ' OF THE PROGRAM TO PROVIDE THE 32 OUTPUT BUCKETS.  THE OUTPUT VALUES ARE
155 ' MULTIPLIED BY "M" TO MAKE IT EASY TO COMPARE WITH THE ASM-96 PROGRAM
160 '
165 INPUT "NAME OF LIST FILE"; LST$
170 PRINT
175 OPEN LST$ FOR OUTPUT AS #1
180 '
200                                                   ' SET UP VARIABLES FOR BASIC
210 DIM XR(32),XI(32),WR(32),WI(32),BR(32)
220 M=16383                                    ' M=MULT. FACTOR FOR SCALING
230 N=32 : N1=31 : N2=N/2                      ' N=NUMBER OF DATA POINTS
240 LOOP=1  : K=0 : EXPONENT=5 : SHIFT=EXPONENT-1       ' 2**E=N
250 PI=3.141592654# : TPN=2*PI/N : PIN=PI/N
260 '
270                                                    ' READ IN CONSTANTS
280 FOR P=0 TO 31 : PN=P*TPN
290 WR(P)=COS(PN) : WI(P)=-SIN(PN) : READ BR(P)
300 NEXT P
310 '
320 FOR K=0 TO 31                                      ' READ IN DATA
330 READ XR(K)  : READ XI(K)
350 NEXT K
360 '
400              ' INITIALIZATION OF LOOP
410 K=0
420 IF LOOP>EXPONENT THEN 700
430 INCNT=0
440              ' ACTUAL CALCULATIONS BEGIN HERE
445 '
450 INCNT=INCNT+1
460 P=BR(INT(K/(2^SHIFT)))
470 WRP=WR(P) : WIP=WI(P) : KN2=K+N2         ' WRP AND WIP ARE CONSTANTS BASED ON
480 TMPR= (WRP*XR(KN2) - WIP*XI(KN2))/2      ' SINES AND COSINES OF BIT REVERSED
490 TMPI= (WRP*XI(KN2) + WIP*XR(KN2))/2      ' VALUES OF K SHIFTED RIGHT S TIMES
500 TMPR1=XR(K)/2 : TMPI1=XI(K)/2
510 XR(K+N2) = TMPR1 - TMPR      ' TMPR, TMPI ARE THE REAL AND IMAGINARY
520 XI(K+N2) = TMPI1 - TMPI      ' RESULTS OF A COMPLEX MULTIPLICATION
530 XR(K) = TMPR1 + TMPR
540 XI(K) = TMPI1 + TMPI
550 '
560 K=K+1
570 IF INCNT<N2 THEN GOTO 450
580 K=K+N2                      ' SINCE THE ARRAY IS PROCESSED 2 POINTS AT A TIME,
590 IF K<N1 THEN GOTO 430       ' ONLY N/2 LOOPS NEED TO BE MADE.  ON EACH PASS,
600 LOOP=LOOP+1 : N2=N2/2       ' THE VALUE OF N2 CHANGES AND SMALLER CONSECUTIVE
605 SHIFT=SHIFT-1               ' SECTIONS ARE PROCESSED.
610 GOTO 400
620 '
690 '
691 '
692 '
693 '
```

270189-17

Listing 1—BASIC FFT Program



```
694 '
695 '
696 '
697 '
700                               ' POST-PROCESSING AND REORDERING BEGIN HERE
710 '
720 FOR K = 0 TO 31
730 KPIN=K*PIN
740 XRBRK=XR(BR(K)) : XIBRK=XI(BR(K))    ' CONDENSED FOR EASE ØF ASM PROGRAMMING
750 XRBRNK=XR(BR(N-K)) : XIBRNK=XI(BR(N-K))
760 TI = (XIBRK+XIBRNK)/2
770 TR = (XRBRK-XRBRNK)/2
780 XRT= (XRBRK+XRBRNK)/4
790 XIT= (XIBRK-XIBRNK)/4
800 OUTR= XRT + TI*COS(KPIN)/2 - TR*SIN(KPIN)/2
810 OUTI= XIT - TI*SIN(KPIN)/2 - TR*COS(KPIN)/2
820 '
830 MAGSQ = OUTR*OUTR+OUTI*OUTI      ' THE ASM-96 PROGRAM USES A TABLE LOOK-UP
840 MAG = SQR(MAGSQ)                 ' ROUTINE TO CALCULATE SQUARE ROOTS
845 IF MAGSQ*M < .5 THEN DECIBEL=0 : GOTO 900
847 DBFACT=M/2/32767*M  ' M^2 / 64K
850 DECIBEL=10*LOG(MAGSQ*DBFACT)
860 DECIBEL=DECIBEL * .434294481#
900    GOTO 930
910 PRINT #1, USING "######   "; K,
920 PRINT #1, USING "\         \"; HEX$(M*OUTR), HEX$(M*OUTI), HEX$(M*MAG)
930 ' GOTO 950
942 PRINT #1, USING "##  "; K;
943 PRINT #1, USING "##.#####  "; OUTR,OUTI,MAG;
945 PRINT #1, USING "###.###  "; DECIBEL;
947 PRINT #1, USING "######   "; M*OUTR, M*OUTI, M*MAG
950 NEXT K
960 '
970 IF LST$<>"SCRN:" THEN PRINT #1, CHR$(12)
999 END
1000 END
1010             ' DATA FOR BR(P)  - BIT REVERSAL
1020 DATA 0,16,8,24,4,20,12,28,2,18,10,26,6,22,14,30
1030 DATA 1,17,9,25,5,21,13,29,3,19,11,27,7,23,15,31
1040             ' DATA FOR XR,XI
1050 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1060 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1070 DATA -2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2
1080 DATA -2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2
```

270189-18

Listing 1—BASIC FFT Program (Continued)

Lines 165-175 set up the file for printing the data, this can be SCRN:, LPT1:, or any other file.

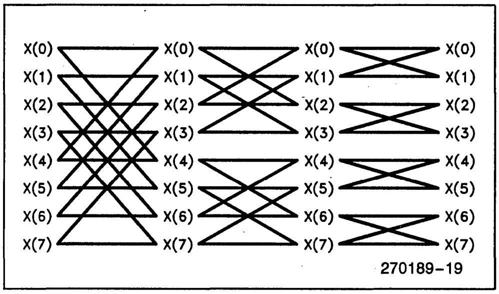


Figure 12. Butterflies with N = 8

Lines 200-310 set up the constants and calculate the WP terms which are stored in the matrices WR(p) and WI(p), for the real and imaginary component respectively.

Lines 320-350 read in the data, alternately placing it into the real and imaginary arrays. The data is scaled by 2 to make the data table simpler.

Lines 410-430 initialize the loop and test for completion.

Lines 450-620 perform the FFT algorithm. Note that all calculations are complex, with the suffixes "R" and "I" indicating real and imaginary components respectively.

The variables on line 470, TMPR1 and TMPI1 would normally not be used in a BASIC program as more than one operation can be performed on each line. However, indirect table lookups always use a separate line of assembly code, so separate lines have been used here.

Lines 700-810 perform the post-weave. This is not in the flowchart, but can be found in Equation 11. Once again, table look-ups are separated and additional variables are used for clarity. The variables BR(x) are the bit reversal values of x.

Line 830 calculates the magnitude of the harmonic components.

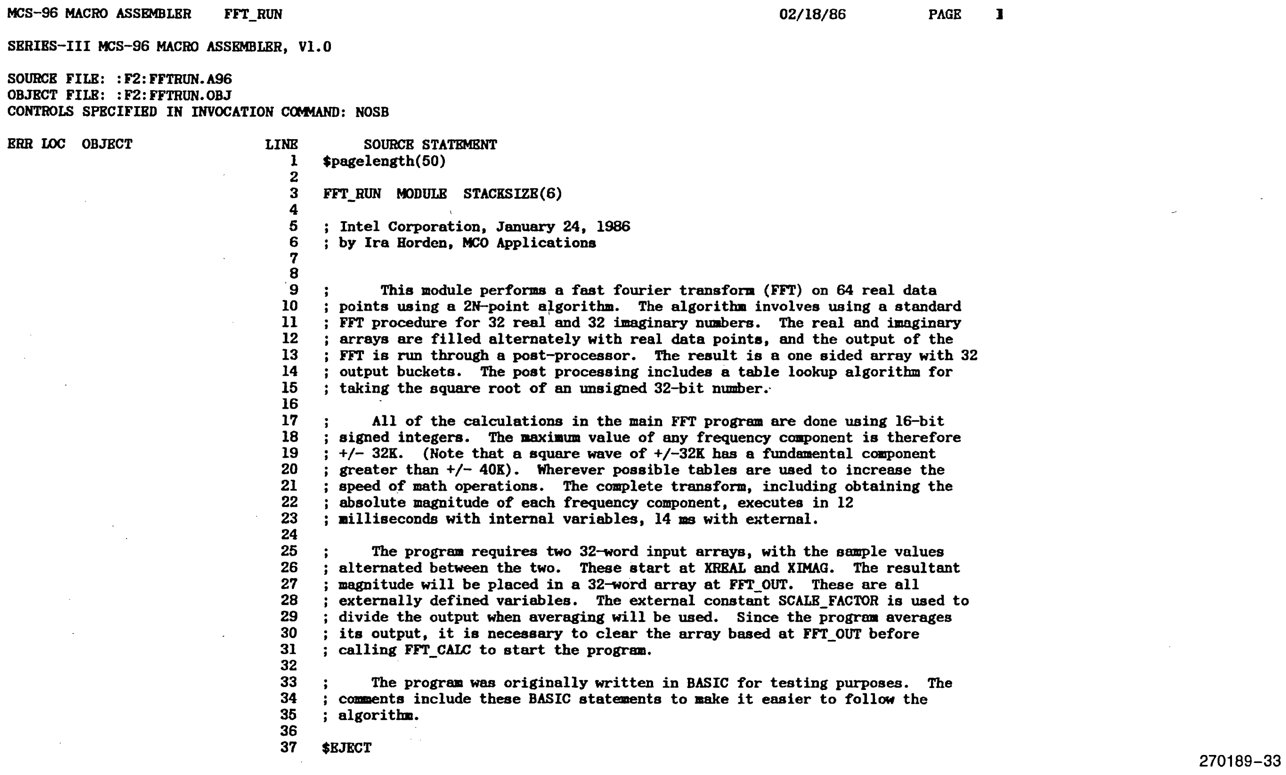
Lines 900-950 print the results of the calculations, with line 900 determining if the print-out should be in hex or decimal.

Lines 1000-1080 are the data for the bit reversal values and input datapoints. The input waveform is one cycle of a square-wave.

7.0 ASM96 PROGRAM FOR FFTS

The BASIC program just presented has been used as an outline for the ASM96 program shown in Listing 2. There are many advantages to using the BASIC program as a model, the main ones being debugging and testing. Since the BASIC program is so similar in program flow to the ASM96 program, it's possible to stop the ASM96 program at almost any point and verify that the results are correct.

Listing 2—ASM96 FFT Program



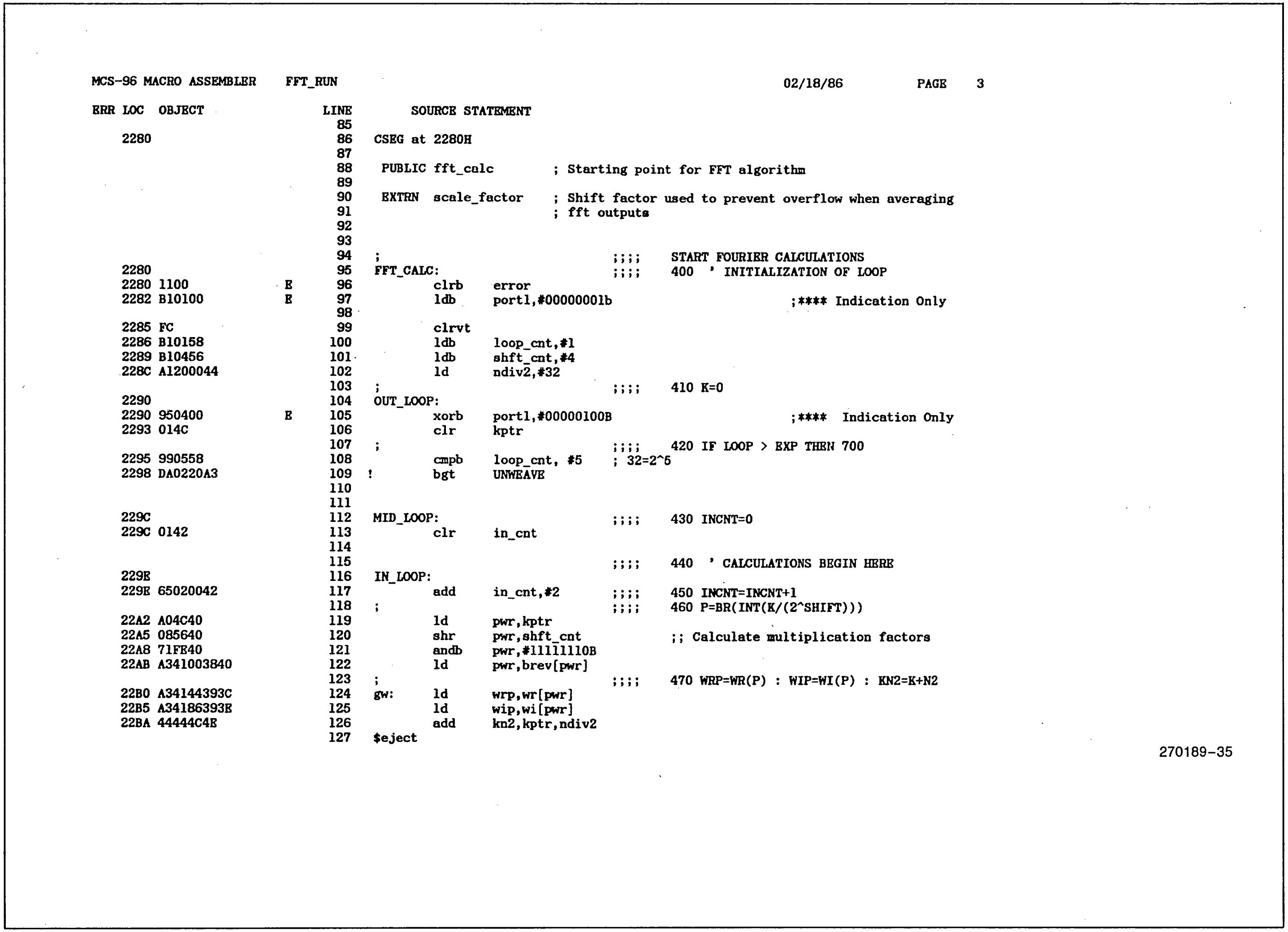
```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                          02/18/86          PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:FFTRUN.A96
OBJECT FILE: :F2:FFTRUN.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                        $pagelength(50)

                                        FFT_RUN  MODULE  STACKSIZE(6)

                                        ; Intel Corporation, January 24, 1986
                                        ; by Ira Horden, MCO Applications


                                        ;      This module performs a fast fourier transform (FFT) on 64 real data
                                        ; points using a 2N-point algorithm.  The algorithm involves using a standard
                                        ; FFT procedure for 32 real and 32 imaginary numbers.  The real and imaginary
                                        ; arrays are filled alternately with real data points, and the output of the
                                        ; FFT is run through a post-processor.  The result is a one sided array with 32
                                        ; output buckets.  The post processing includes a table lookup algorithm for
                                        ; taking the square root of an unsigned 32-bit number.

                                        ;     All of the calculations in the main FFT program are done using 16-bit
                                        ; signed integers.  The maximum value of any frequency component is therefore
                                        ; +/- 32K.  (Note that a square wave of +/-32K has a fundamental component
                                        ; greater than +/- 40K).  Wherever possible tables are used to increase the
                                        ; speed of math operations.  The complete transform, including obtaining the
                                        ; absolute magnitude of each frequency component, executes in 12
                                        ; milliseconds with internal variables, 14 ms with external.

                                        ;     The program requires two 32-word input arrays, with the sample values
                                        ; alternated between the two.  These start at XREAL and XIMAG.  The resultant
                                        ; magnitude will be placed in a 32-word array at FFT_OUT.  These are all
                                        ; externally defined variables.  The external constant SCALE_FACTOR is used to
                                        ; divide the output when averaging will be used.  Since the program averages
                                        ; its output, it is necessary to clear the array based at FFT_OUT before
                                        ; calling FFT_CALC to start the program.

                                        ;     The program was originally written in BASIC for testing purposes.  The
                                        ; comments include these BASIC statements to make it easier to follow the
                                        ; algorithm.

                                        $EJECT
```

270189-33

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                        02/18/86          PAGE    2

ERR LOC  OBJECT                   LINE        SOURCE STATEMENT
                                    38
    0000                            39   RSEG
                                    40    EXTRN  portl, zero, error
                                    41
    0024                            42   OSEG at 24H
    0024                            43          TMPR:   dsl     1       ; Temporary register, Real
    0028                            44          TMPI:   dsl     1       ; Temporary register, Imaginary
    002C                            45          TMPRl:  dsl     1       ; Temporary registerl, Real
    0030                            46          TMPIl:  dsl     1       ; Temporary registerl, Imaginary
    0034                            47          XRTMP:  dsl     1       ; Temporary data register, Real
    0038                            48          XITMP:  dsl     1       ; Temporary data register, Imaginary
    003C                            49          XRRK:   dsl     1
    0040                            50          XRRNK:  dsl     1
    0044                            51          XIRK:   dsl     1
    0048                            52          XIRNK:  dsl     1
      003C                          53          diff    equ     xrrk    :long   ; Table difference for square root
      0040                          54          sqrt    equ     xrrnk   :long   ; Square root
      0040                          55          log     equ     xrrnk   :long   ; 10 Log magnitude^2
      0044                          56          nxtloc  equ     xirk    :long   ; Next location in table
                                    57
      003C                          58          WRP     equ     xrrk    :word   ; Multiplication factor, Real
      003E                          59          WIP     equ     xrrk+2  :word   ; Multiplication factor, Imaginary
      0040                          60          PWR     equ     xrrnk   :word
      0042                          61          IN_CNT  equ     xrrnk+2 :word
      0044                          62          NDIV2   equ     xirk    :word   ; n divided by 2  (0 < n < N) *2
                                    63
    004C                            64          KPTR:           dsw     1       ; K for counter *2 to index words
    004E                            65          KN2:            dsw     1       ; KPTR + NDIV2
    0050                            66          N_SUB_K:        dsw     1       ; N-K *2 to index words
    0052                            67          RK:             dsw     1       ; Bit reversed pointer of KPTR
    0054                            68          RNK:            dsw     1       ; Bit reversed pointer of N_SUB_K
    0056                            69          SHFT_CNT:       dsw     1
    0058                            70          LOOP_CNT:       dsb     1
      004E                          71          ptr     equ     kn2     :word   ; Pointer for square root table
    0000                            72   DSEG
                                    73
                                    74   EXTRN   FFT_MODE       ; FFT_MODE: mode for FFT input and graphing
                                    75   EXTRN   XREAL, XIMAG   ; XREAL, XIMAG: Base  addresses for 32 16-bit signed
                                    76                          ; entries for real and imaginary numbers respectively.
                                    77   EXTRN   FFT_OUT        ; FFT_OUT: Starting address for 32 word array
                                    78                          ; of magnitude information.
                                    79
    0000                            80          OUTR:   dsw     32      ; Real component of fft
    0040                            81          OUTI:   dsw     32      ; Imaginary component of waveform
                                    82    PUBLIC OUTR,OUTI
                                    83
                                    84   $EJECT
```

270189-34

Listing 2—ASM96 FFT Program (Continued)



```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                          02/18/86          PAGE    3

ERR LOC  OBJECT               LINE        SOURCE STATEMENT
                                85
    2280                        86   CSEG at 2280H
                                87
                                88    PUBLIC fft_calc        ; Starting point for FFT algorithm
                                89
                                90    EXTRN  scale_factor    ; Shift factor used to prevent overflow when averaging
                                91                           ; fft outputs
                                92
                                93
                                94   ;                                 ;;;;    START FOURIER CALCULATIONS
    2280                        95   FFT_CALC:                         ;;;;    400  ' INITIALIZATION OF LOOP
    2280 1100              E    96           clrb    error
    2282 B10100            E    97           ldb     port1,#00000001b                       ;**** Indication Only
                                98
    2285 FC                     99           clrvt
    2286 B10158                100           ldb     loop_cnt,#1
    2289 B10456                101           ldb     shft_cnt,#4
    228C A1200044              102           ld      ndiv2,#32
                               103   ;                                 ;;;;    410 K=0
    2290                       104   OUT_LOOP:
    2290 950400            E   105           xorb    port1,#00000100B                       ;****  Indication Only
    2293 014C                  106           clr     kptr
                               107   ;                                 ;;;;    420 IF LOOP > EXP THEN 700
    2295 990558                108           cmpb    loop_cnt, #5      ; 32=2^5
    2298 DA0220A3              109 !         bgt     UNWEAVE
                               110
                               111
    229C                       112   MID_LOOP:                         ;;;;    430 INCNT=0
    229C 0142                  113           clr     in_cnt
                               114
                               115                                     ;;;;    440  ' CALCULATIONS BEGIN HERE
    229E                       116   IN_LOOP:
    229E 65020042              117           add     in_cnt,#2         ;;;;    450 INCNT=INCNT+1
                               118   ;                                 ;;;;    460 P=BR(INT(K/(2^SHIFT)))
    22A2 A04C40                119           ld      pwr,kptr
    22A5 085640                120           shr     pwr,shft_cnt              ;; Calculate multiplication factors
    22A8 71FE40                121           andb    pwr,#11111110B
    22AB A341003840            122           ld      pwr,brev[pwr]
                               123   ;                                 ;;;;    470 WRP=WR(P) : WIP=WI(P) : KN2=K+N2
    22B0 A341443930            124   gw:     ld      wrp,wr[pwr]
    22B5 A34186393E            125           ld      wip,wi[pwr]
    22BA 44444C4E              126           add     kn2,kptr,ndiv2
                               127   $eject
```

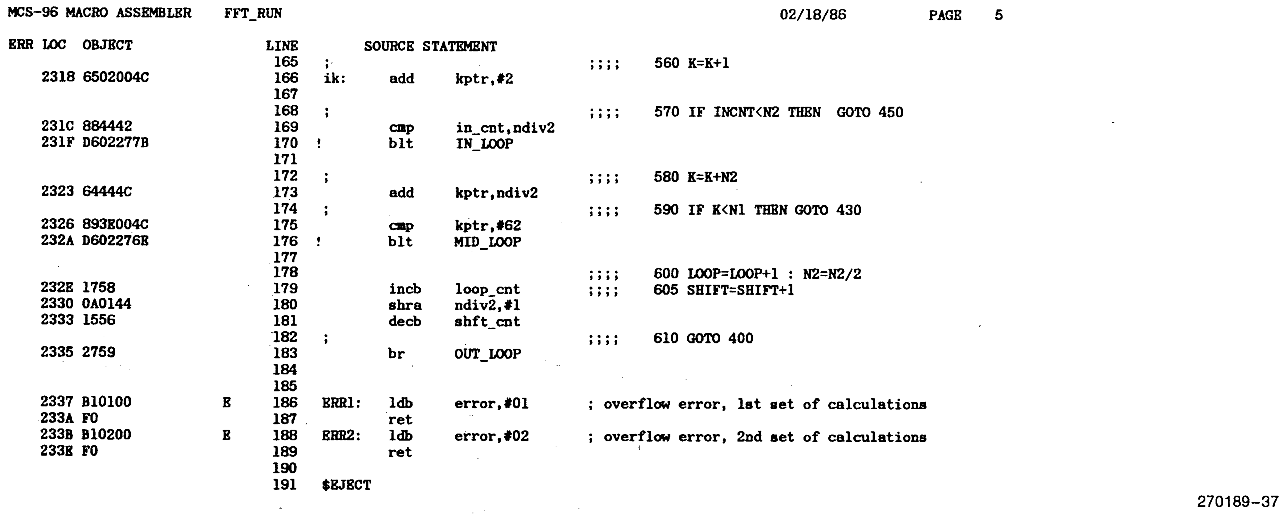
270189-35

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                              02/18/86         PAGE    4

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                128          ;; Complex multiplication follows
                                129
                                130    ;                               ;;;;     480 TMPR= (WRP*XR(KN2) - WIP*XI(KN2))/2
    22BE FE4F4F00003C24   E     131    gm:      mul     tmpr,wrp,xreal[kn2]
    22C5 FE4F4F00003E28   E     132             mul     tmpi,wip,ximag[kn2]
    22CC 682A26                 133             sub     tmpr+2,tmpi+2
                                134    ;                               ;;;;     490 TMPI= (WRP*XI(KN2) + WIP*XR(KN2))/2
    22CF FE4F4F00003C2C   E     135             mul     tmprl,wrp,ximag[kn2]
    22D6 FE4F4F00003E28   E     136             mul     tmpi,wip,xreal[kn2]
    22DD 642E2A                 137             add     tmpi+2,tmprl+2
                                138
                                139                             ;;  using the high byte only of a signed multiply
                                140                             ;;  provides an effective divide by two
                                141
    22E0 DC55                   142             BVT     ERR1    ; Branch on error in complex multiplications
                                143
    22E2 A34D00002C       E     144             ld      tmprl,xreal[kptr]      ;;;;    500 TMPR1=XR(K)/2 :
    22E7 0A012C                 145             shra    tmprl,#1               ;;;;        TMPI1=XI(K)/2
    22EA A34D000030       E     146             ld      tmpil,ximag[kptr]
    22EF 0A0130                 147             shra    tmpil,#1
                                148
                                149    ;                               ;;;;     510 XR(KN2) = TMPR1 - TMPR
    22F2 48262C34               150    gr2:     sub     xrtmp,tmprl,tmpr+2
    22F6 C34F000034       E     151             st      xrtmp,xreal[kn2]
                                152    ;                               ;;;;     520 XI(KN2) = TMPI1 - TMPI
    22FB 482A3038               153    gx2:     sub     xitmp,tmpil,tmpi+2
    22FF C34F000038       E     154             st      xitmp,ximag[kn2]
                                155    ;                               ;;;;     530 XR(K) = TMPR1 + TMPR
    2304 44262C34               156             add     xrtmp,tmprl,tmpr+2
    2308 C34D000034       E     157             st      xrtmp,xreal[kptr]
                                158    ;                               ;;;;     540 XI(K) = TMPI1 + TMPI
    230D 442A3038               159    gx:      add     xitmp,tmpil,tmpi+2
    2311 C34D000038       E     160             st      xitmp,ximag[kptr]
                                161
    2316 DC23                   162             BVT     ERR2    ; Branch on error in complex additions
                                163
                                164    $eject
```

270189-36

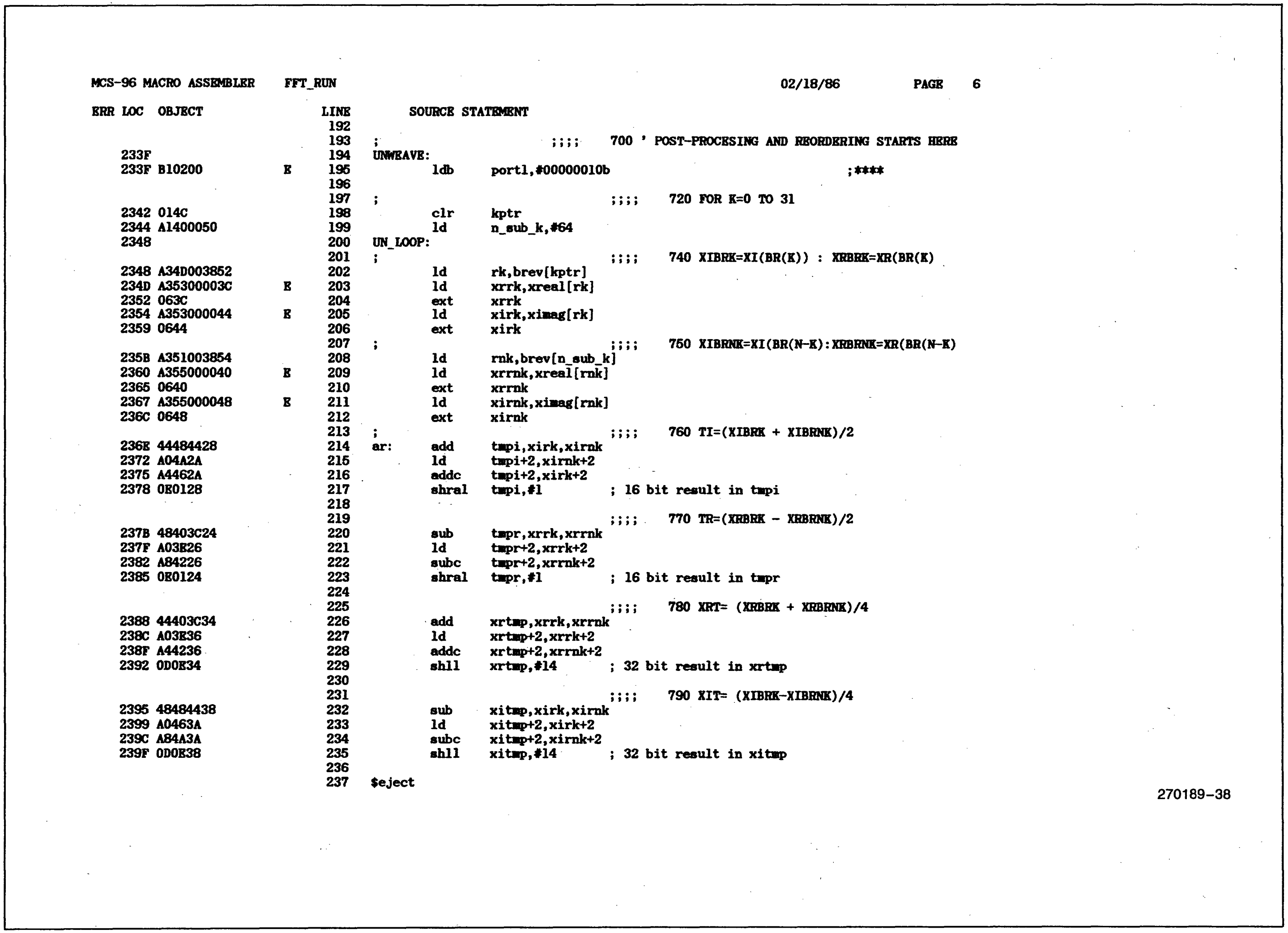
Listing 2—ASM96 FFT Program (Continued)



```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                             02/18/86         PAGE    5

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                               165   ;                                 ;;;;    560 K=K+1
    2318 6502004C              166   ik:     add      kptr,#2
                               167
                               168   ;                                 ;;;;    570 IF INCNT<N2 THEN  GOTO 450
    231C 884442                169           cmp      in_cnt,ndiv2
    231F D602277B              170  !        blt      IN_LOOP
                               171
                               172   ;                                 ;;;;    580 K=K+N2
    2323 64444C                173           add      kptr,ndiv2
                               174   ;                                 ;;;;    590 IF K<N1 THEN GOTO 430
    2326 893E004C              175           cmp      kptr,#62
    232A D602276E              176  !        blt      MID_LOOP
                               177
                               178                                     ;;;;    600 LOOP=LOOP+1 : N2=N2/2
    232E 1758                  179           incb     loop_cnt         ;;;;    605 SHIFT=SHIFT+1
    2330 0A0144                180           shra     ndiv2,#1
    2333 1556                  181           decb     shft_cnt
                               182   ;                                 ;;;;    610 GOTO 400
    2335 2759                  183           br       OUT_LOOP
                               184
                               185
    2337 B10100          E     186   ERR1:   ldb      error,#01        ; overflow error, 1st set of calculations
    233A F0                    187           ret
    233B B10200          E     188   ERR2:   ldb      error,#02        ; overflow error, 2nd set of calculations
    233E F0                    189           ret
                               190
                               191   $EJECT
```

270189-37

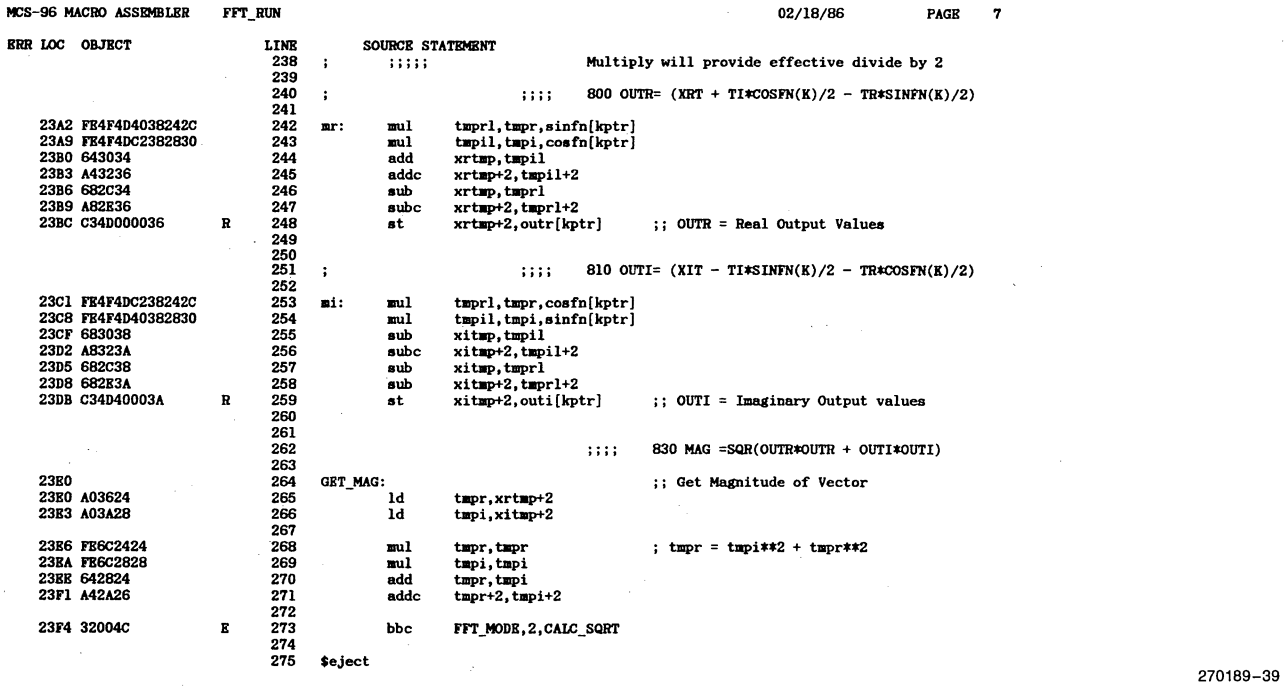
Listing 2—ASM96 FFT Program (Continued)



```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                          02/18/86         PAGE    6

ERR LOC  OBJECT                 LINE        SOURCE STATEMENT
                                 192
                                 193  ;                        ;;;;   700 ' POST-PROCESING AND REORDERING STARTS HERE
    233F                         194  UNWEAVE:
    233F B10200            E     195          ldb     port1,#00000010b                          ;****
                                 196
                                 197  ;                                ;;;;   720 FOR K=0 TO 31
    2342 014C                    198          clr     kptr
    2344 A1400050                199          ld      n_sub_k,#64
    2348                         200  UN_LOOP:
                                 201  ;                                ;;;;   740 XIBRK=XI(BR(K)) : XRBRK=XR(BR(K)
    2348 A34D003852              202          ld      rk,brev[kptr]
    234D A35300003C        E     203          ld      xrrk,xreal[rk]
    2352 063C                    204          ext     xrrk
    2354 A353000044        E     205          ld      xirk,ximag[rk]
    2359 0644                    206          ext     xirk
                                 207  ;                                ;;;;   750 XIBRNK=XI(BR(N-K):XRBRNK=XR(BR(N-K)
    235B A351003854              208          ld      rnk,brev[n_sub_k]
    2360 A355000040        E     209          ld      xrrnk,xreal[rnk]
    2365 0640                    210          ext     xrrnk
    2367 A355000048        E     211          ld      xirnk,ximag[rnk]
    236C 0648                    212          ext     xirnk
                                 213  ;                                ;;;;   760 TI=(XIBRK + XIBRNK)/2
    236E 44484428                214  ar:     add     tmpi,xirk,xirnk
    2372 A04A2A                  215          ld      tmpi+2,xirnk+2
    2375 A4462A                  216          addc    tmpi+2,xirk+2
    2378 0E0128                  217          shral   tmpi,#1         ; 16 bit result in tmpi
                                 218
                                 219                                   ;;;;   770 TR=(XRBRK - XRBRNK)/2
    237B 48403C24                220          sub     tmpr,xrrk,xrrnk
    237F A03E26                  221          ld      tmpr+2,xrrk+2
    2382 A84226                  222          subc    tmpr+2,xrrnk+2
    2385 0E0124                  223          shral   tmpr,#1         ; 16 bit result in tmpr
                                 224
                                 225                                   ;;;;   780 XRT= (XRBRK + XRBRNK)/4
    2388 44403C34                226          add     xrtmp,xrrk,xrrnk
    238C A03E36                  227          ld      xrtmp+2,xrrk+2
    238F A44236                  228          addc    xrtmp+2,xrrnk+2
    2392 0D0E34                  229          shll    xrtmp,#14       ; 32 bit result in xrtmp
                                 230
                                 231                                   ;;;;   790 XIT= (XIBRK-XIBRNK)/4
    2395 48484438                232          sub     xitmp,xirk,xirnk
    2399 A0463A                  233          ld      xitmp+2,xirk+2
    239C A84A3A                  234          subc    xitmp+2,xirnk+2
    239F 0D0E38                  235          shll    xitmp,#14       ; 32 bit result in xitmp
                                 236
                                 237  $eject
```

270189-38

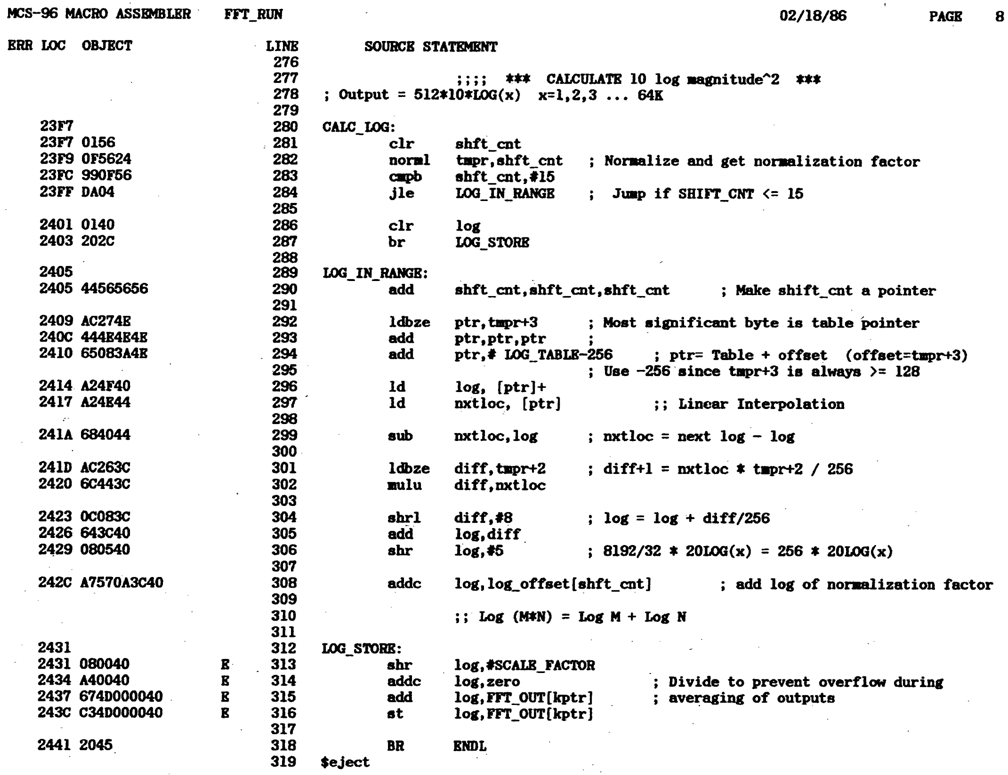
Listing 2—ASM96 FFT Program (Continued)



```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                        02/18/86        PAGE    7

ERR LOC  OBJECT               LINE       SOURCE STATEMENT
                               238  ;       ;;;;;                   Multiply will provide effective divide by 2
                               239
                               240  ;                       ;;;;    800 OUTR= (XRT + TI*COSFN(K)/2 - TR*SINFN(K)/2)
                               241
    23A2 FE4F4D4038242C        242  mr:     mul     tmprl,tmpr,sinfn[kptr]
    23A9 FE4F4DC2382830        243          mul     tmpil,tmpi,cosfn[kptr]
    23B0 643034                244          add     xrtmp,tmpil
    23B3 A43236                245          addc    xrtmp+2,tmpil+2
    23B6 682C34                246          sub     xrtmp,tmprl
    23B9 A82E36                247          subc    xrtmp+2,tmprl+2
    23BC C34D000036       R    248          st      xrtmp+2,outr[kptr]      ;; OUTR = Real Output Values
                               249
                               250
                               251  ;                       ;;;;    810 OUTI= (XIT - TI*SINFN(K)/2 - TR*COSFN(K)/2)
                               252
    23C1 FE4F4DC238242C        253  mi:     mul     tmprl,tmpr,cosfn[kptr]
    23C8 FE4F4D40382830        254          mul     tmpil,tmpi,sinfn[kptr]
    23CF 683038                255          sub     xitmp,tmpil
    23D2 A8323A                256          subc    xitmp+2,tmpil+2
    23D5 682C38                257          sub     xitmp,tmprl
    23D8 682E3A                258          sub     xitmp+2,tmprl+2
    23DB C34D40003A       R    259          st      xitmp+2,outi[kptr]      ;; OUTI = Imaginary Output values
                               260
                               261
                               262                                  ;;;;    830 MAG =SQR(OUTR*OUTR + OUTI*OUTI)
                               263
    23E0                       264  GET_MAG:                                ;; Get Magnitude of Vector
    23E0 A03624                265          ld      tmpr,xrtmp+2
    23E3 A03A28                266          ld      tmpi,xitmp+2
                               267
    23E6 FE6C2424              268          mul     tmpr,tmpr               ; tmpr = tmpi**2 + tmpr**2
    23EA FE6C2828              269          mul     tmpi,tmpi
    23EE 642824                270          add     tmpr,tmpi
    23F1 A42A26                271          addc    tmpr+2,tmpi+2
                               272
    23F4 32004C           E    273          bbc     FFT_MODE,2,CALC_SQRT
                               274
                               275  $eject
```

270189-39

Listing 2—ASM96 FFT Program (Continued)



```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                                            02/18/86          PAGE    8

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                276
                                277                   ;;;;  ***  CALCULATE 10 log magnitude^2  ***
                                278  ; Output = 512*10*LOG(x)  x=1,2,3 ... 64K
                                279
    23F7                        280  CALC_LOG:
    23F7 0156                   281          clr     shft_cnt
    23F9 0F5624                 282          norml   tmpr,shft_cnt   ; Normalize and get normalization factor
    23FC 990F56                 283          cmpb    shft_cnt,#15
    23FF DA04                   284          jle     LOG_IN_RANGE    ;  Jump if SHIFT_CNT <= 15
                                285
    2401 0140                   286          clr     log
    2403 202C                   287          br      LOG_STORE
                                288
    2405                        289  LOG_IN_RANGE:
    2405 44565656               290          add     shft_cnt,shft_cnt,shft_cnt     ; Make shft_cnt a pointer
                                291
    2409 AC274E                 292          ldbze   ptr,tmpr+3      ; Most significant byte is table pointer
    240C 444E4E4E               293          add     ptr,ptr,ptr     ;
    2410 65083A4E               294          add     ptr,# LOG_TABLE-256    ; ptr= Table + offset  (offset=tmpr+3)
                                295                                  ; Use -256 since tmpr+3 is always >= 128
    2414 A24F40                 296          ld      log, [ptr]+
    2417 A24E44                 297          ld      nxtloc, [ptr]          ;; Linear Interpolation
                                298
    241A 684044                 299          sub     nxtloc,log      ; nxtloc = next log - log
                                300
    241D AC263C                 301          ldbze   diff,tmpr+2     ; diff+1 = nxtloc * tmpr+2 / 256
    2420 6C443C                 302          mulu    diff,nxtloc
                                303
    2423 0C083C                 304          shrl    diff,#8         ; log = log + diff/256
    2426 643C40                 305          add     log,diff
    2429 080540                 306          shr     log,#5          ; 8192/32 * 20LOG(x) = 256 * 20LOG(x)
                                307
    242C A7570A3C40             308          addc    log,log_offset[shft_cnt]       ; add log of normalization factor
                                309
                                310                  ;; Log (M*N) = Log M + Log N
                                311
    2431                        312  LOG_STORE:
    2431 080040           E     313          shr     log,#SCALE_FACTOR
    2434 A40040           E     314          addc    log,zero               ; Divide to prevent overflow during
    2437 674D000040       E     315          add     log,FFT_OUT[kptr]      ; averaging of outputs
    243C C34D000040       E     316          st      log,FFT_OUT[kptr]
                                317
    2441 2045                   318          BR      ENDL
                                319  $eject
```

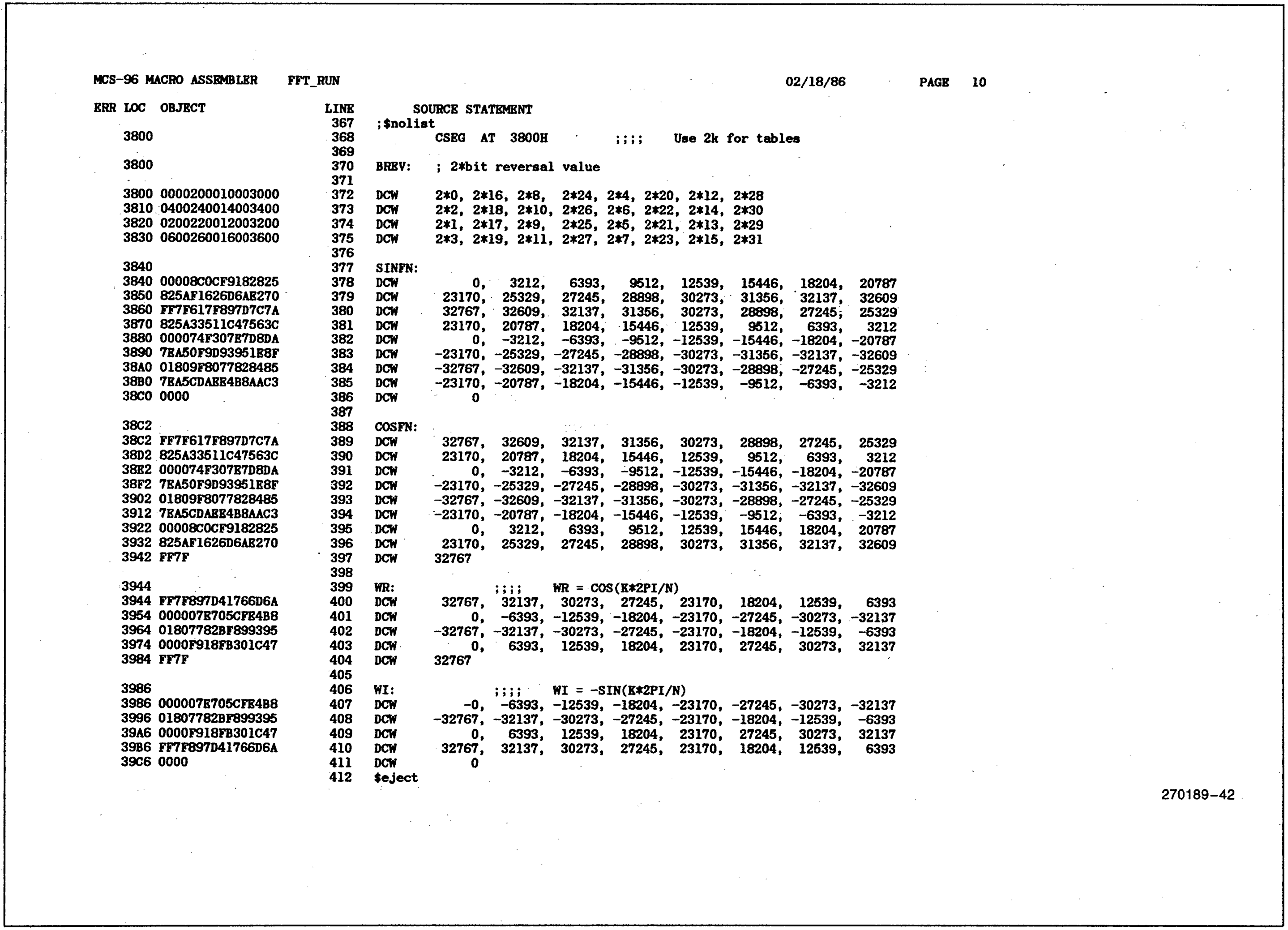
270189-40

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                                  02/18/86         PAGE    9

ERR LOC  OBJECT              LINE        SOURCE STATEMENT
                              320                          ;;;;  ***  CALCULATE SQUARE  ROOT  ***
    2443                      321   CALC_SQRT:
                              322
    2443 0156                 323   clr      shft_cnt
    2445 0F5624               324            norml   tmpr,shft_cnt   ; Normalize and get normalization factor
                              325
    2448 D705                 326            jne     SQRT_IN_RANGE   ;  Jump if tmpr > 0
    244A C04200          E    327            st      zero,sqrt+2
    244D 2029                 328            br      SQRT_STORE
                              329
    244F                      330   SQRT_IN_RANGE:
    244F AC274E               331            ldbze   ptr,tmpr+3      ; Most significant byte is table pointer
    2452 444E4E4E             332            add     ptr,ptr,ptr
    2456 6508394E             333            add     ptr,# SQ_TABLE-256      ; ptr= Table + offset  (offset=tmpr+3)
                              334                                    ; Use -256 since tmpr+3 is always >= 128
    245A A24F40               335            ld      sqrt, [ptr]+
    245D A24E44               336            ld      nxtloc, [ptr]           ;; Linear Interpolation
                              337
    2460 684044               338            sub     nxtloc,sqrt     ; nxtloc =  sqrt - next sqrt
                              339
    2463 AC263C               340            ldbze   diff,tmpr+2     ; diff+1 = nxtloc * tmpr+2 / 256
    2466 6C443C               341            mulu    diff,nxtloc
                              342
    2469 AC3D3C               343            ldbze   diff,diff+1     ; sqrt = sqrt + delta   (diff < 0FFH)
    246C 643C40               344            add     sqrt,diff
                              345
    246F 44565656             346            add     shft_cnt,shft_cnt,shft_cnt
                              347
    2473 6F57C83940           348            mulu    sqrt,tab_sqr[shft_cnt]  ; divide by normalization factor
                              349
                              350                                    ;; mulu acts as divide since if tab2=0FFFFH
                              351                                    ;; sqrt would remain essentialy unchanged
    2478                      352   SQRT_STORE:
    2478 080042          E    353            shr     sqrt+2,#SCALE_FACTOR
    247B A40042          E    354            addc    sqrt+2,zero             ; Divide to prevent overflow during
    247E 674D000042      E    355            add     sqrt+2,FFT_OUT[kptr]    ; averaging of outputs
    2483 C34D000042      E    356            st      sqrt+2,FFT_OUT[kptr]
                              357
                              358   ;                ;;;;    ***  END  OF  LOOP  ***
                              359
                              360                                    ;;;;    950 NEXT K
    2488 6502004C             361   ENDL:    add     kptr,#2
    248C 69020050             362            sub     n_sub_k,#2
    2490 DF0226B4             363  !         bne     UN_LOOP
                              364
    2494 F0                   365            RET
                              366   $eject
```

270189-41

Listing 2—ASM96 FFT Program (Continued)



```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                          02/18/86         PAGE  10

ERR LOC  OBJECT              LINE        SOURCE STATEMENT
                              367   ;$nolist
    3800                      368         CSEG  AT  3800H           ;;;;   Use 2k for tables
                              369
    3800                      370   BREV:   ; 2*bit reversal value
                              371
    3800 0000200010003000     372   DCW     2*0, 2*16, 2*8,  2*24, 2*4, 2*20, 2*12, 2*28
    3810 0400240014003400     373   DCW     2*2, 2*18, 2*10, 2*26, 2*6, 2*22, 2*14, 2*30
    3820 0200220012003200     374   DCW     2*1, 2*17, 2*9,  2*25, 2*5, 2*21, 2*13, 2*29
    3830 0600260016003600     375   DCW     2*3, 2*19, 2*11, 2*27, 2*7, 2*23, 2*15, 2*31
                              376
    3840                      377   SINFN:
    3840 00008C0CF9182825     378   DCW          0,   3212,   6393,   9512,  12539,  15446,  18204,  20787
    3850 825AF1626D6AE270     379   DCW      23170,  25329,  27245,  28898,  30273,  31356,  32137,  32609
    3860 FF7F617F897D7C7A     380   DCW      32767,  32609,  32137,  31356,  30273,  28898,  27245,  25329
    3870 825A33511C47563C     381   DCW      23170,  20787,  18204,  15446,  12539,   9512,   6393,   3212
    3880 000074F307E7D8DA     382   DCW          0,  -3212,  -6393,  -9512, -12539, -15446, -18204, -20787
    3890 7EA50F9D93951E8F     383   DCW     -23170, -25329, -27245, -28898, -30273, -31356, -32137, -32609
    38A0 01809F8077828485     384   DCW     -32767, -32609, -32137, -31356, -30273, -28898, -27245, -25329
    38B0 7EA5CDAEE4B8AAC3     385   DCW     -23170, -20787, -18204, -15446, -12539,  -9512,  -6393,  -3212
    38C0 0000                 386   DCW          0
                              387
    38C2                      388   COSFN:
    38C2 FF7F617F897D7C7A     389   DCW      32767,  32609,  32137,  31356,  30273,  28898,  27245,  25329
    38D2 825A33511C47563C     390   DCW      23170,  20787,  18204,  15446,  12539,   9512,   6393,   3212
    38E2 000074F307E7D8DA     391   DCW          0,  -3212,  -6393,  -9512, -12539, -15446, -18204, -20787
    38F2 7EA50F9D93951E8F     392   DCW     -23170, -25329, -27245, -28898, -30273, -31356, -32137, -32609
    3902 01809F8077828485     393   DCW     -32767, -32609, -32137, -31356, -30273, -28898, -27245, -25329
    3912 7EA5CDAEE4B8AAC3     394   DCW     -23170, -20787, -18204, -15446, -12539,  -9512,  -6393,  -3212
    3922 00008C0CF9182825     395   DCW          0,   3212,   6393,   9512,  12539,  15446,  18204,  20787
    3932 825AF1626D6AE270     396   DCW      23170,  25329,  27245,  28898,  30273,  31356,  32137,  32609
    3942 FF7F                 397   DCW     32767
                              398
    3944                      399   WR:              ;;;;    WR = COS(K*2PI/N)
    3944 FF7F897D41766D6A     400   DCW      32767,  32137,  30273,  27245,  23170,  18204,  12539,   6393
    3954 000007E705CFE4B8     401   DCW          0,  -6393, -12539, -18204, -23170, -27245, -30273, -32137
    3964 01807782BF899395     402   DCW     -32767, -32137, -30273, -27245, -23170, -18204, -12539,  -6393
    3974 0000F918FB301C47     403   DCW          0,   6393,  12539,  18204,  23170,  27245,  30273,  32137
    3984 FF7F                 404   DCW     32767
                              405
    3986                      406   WI:              ;;;;    WI = -SIN(K*2PI/N)
    3986 000007E705CFE4B8     407   DCW         -0,  -6393, -12539, -18204, -23170, -27245, -30273, -32137
    3996 01807782BF899395     408   DCW     -32767, -32137, -30273, -27245, -23170, -18204, -12539,  -6393
    39A6 0000F918FB301C47     409   DCW          0,   6393,  12539,  18204,  23170,  27245,  30273,  32137
    39B6 FF7F897D41766D6A     410   DCW      32767,  32137,  30273,  27245,  23170,  18204,  12539,   6393
    39C6 0000                 411   DCW          0
                              412   $eject
```

270189-42

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                                      02/18/86         PAGE   11

ERR LOC  OBJECT                    LINE        SOURCE STATEMENT
                                    413
                                    414
    39C8                            415   TAB_SQR:        ; 65535/(square root of 2**SHFT_CNT) ;  0<=SHFT_CNT<32
                                    416
                                    417     ;;    1       2       4       8       16      32      64      128
    39C8 FFFF04B50080825A           418   DCW     65535,  46340,  32768,  23170,  16384,  11585,  8192,   5793
                                    419
                                    420     ;;    256     512     1024    2048    4096    8192    16384   32768
    39D8 0010500B0008A805           421   DCW     4096,   2896,   2048,   1448,   1024;   724,    512,    362
                                    422
                                    423     ;;    65536,  131072, 262144, 524288, ...
    39E8 0001B50080005B00           424   DCW     256,    181,    128,    91,     64,     45,     32,     23
    39F8 10000B0008000600           425   DCW     16,     11,     8,      6,      4,      3,      2,      1
                                    426
                                    427
    3A08                            428   SQ_TABLE:       ; square root of n * 2**24  N=128, 129, 130 ... 255
                                    429
    3A08 05B5BAB56EB621B7           430   DCW        46341, 46522, 46702, 46881, 47059, 47237, 47415, 47591
    3A18 97BA46BBF5BBA3BC           431   DCW        47767, 47942, 48117, 48291, 48465, 48637, 48809, 48981
    3A28 00C0AAC054C1FDC1           432   DCW        49152, 49322, 49492, 49661, 49830, 49998, 50166, 50332
    3A38 43C5E9C58EC633C7           433   DCW        50499, 50665, 50830, 50995, 51159, 51323, 51486, 51649
    3A48 63CA04CBA6CB46CC           434   DCW        51811, 51972, 52134, 52294, 52454, 52614, 52773, 52932
    3A58 62CF00D09DD03AD1           435   DCW        53090, 53248, 53405, 53562, 53719, 53874, 54030, 54185
    3A68 44D4DED477D511D6           436   DCW        54340, 54494, 54647, 54801, 54954, 55106, 55258, 55410
    3A78 09D9A0D936DACCDA           437   DCW        55561, 55712, 55862, 56012, 56162, 56311, 56459, 56608
    3A88 B4DD47DEDBDE6EDF           438   DCW        56756, 56903, 57051, 57198, 57344, 57490, 57636, 57781
    3A98 46E2D7E267E3F7E3           439   DCW        57926, 58071, 58215, 58359, 58503, 58646, 58789, 58931
    3AA8 C1E64FE7DDE76AE8           440   DCW        59073, 59215, 59357, 59498, 59639, 59779, 59919, 60059
    3AB8 27EBB2EB3DECC7EC           441   DCW        60199, 60338, 60477, 60615, 60754, 60891, 61029, 61166
    3AC8 77EF00F088F010F1           442   DCW        61303, 61440, 61576, 61712, 61848, 61984, 62119, 62254
    3AD8 B4F33BF4C1F446F5           443   DCW        62388, 62523, 62657, 62790, 62924, 63057, 63190, 63323
    3AE8 DFF763F8E7F86AF9           444   DCW        63455, 63587, 63719, 63850, 63982, 64113, 64243, 64374
    3AF8 F8FB7AFCFBFC7DFD           445   DCW        64504, 64634, 64763, 64893, 65022, 65151, 65280, 65408
                                    446
                                    447   $eject
```

270189-43

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                                    02/18/86          PAGE   12

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                  448
    3B08                          449   LOG_TABLE:  ;    16384*10*LOG(n/128)     n=128,129,130 ... 256
                                  450
    3B08 00002A024F047006         451   DCW            0,   554,  1103,  1648,  2190,  2727,  3260,  3789
    3B18 DA10E312E914EA16         452   DCW         4314,  4835,  5353,  5866,  6376,  6883,  7386,  7885
    3B28 BD20A92292247826         453   DCW         8381,  8873,  9362,  9848, 10330, 10810, 11286, 11758
    3B38 C42F973166333335         454   DCW        12228, 12695, 13158, 13619, 14076, 14531, 14983, 15432
    3B48 063EC13F7A413043         455   DCW        15878, 16321, 16762, 17200, 17635, 18067, 18497, 18925
    3B58 954B3C4DDF4E8150         456   DCW        19349, 19772, 20191, 20609, 21024, 21436, 21846, 22254
    3B68 8458175AA85B365D         457   DCW        22660, 23063, 23464, 23862, 24259, 24653, 25045, 25435
    3B78 DE646066E0675D69         458   DCW        25822, 26208, 26592, 26973, 27353, 27730, 28106, 28479
    3B88 B370247294730275         459   DCW        28851, 29220, 29588, 29954, 30318, 30680, 31040, 31399
    3B98 0B7C6E7DCF7E2F80         460   DCW        31755, 32110, 32463, 32815, 33165, 33512, 33859, 34203
    3BA8 F28647889B89ED8A         461   DCW        34546, 34887, 35227, 35565, 35902, 36236, 36570, 36901
    3BB8 7091B892FF934595         462   DCW        37232, 37560, 37887, 38213, 38537, 38860, 39181, 39501
    3BC8 8B9BC89C049E3E9F         463   DCW        39819, 40136, 40452, 40766, 41079, 41390, 41700, 42009
    3BD8 4CA57EA6AFA7DEA8         464   DCW        42316, 42622, 42927, 43230, 43533, 43833, 44133, 44431
    3BE8 B9AEE0AF07B12CB2         465   DCW        44729, 45024, 45319, 45612, 45905, 46196, 46486, 46774
    3BF8 D6B7F4B811BA2DBB         466   DCW        47062, 47348, 47633, 47917, 48200, 48482, 48763, 49042
    3C08 A9C0                     467   DCW        49321
                                  468
    3C0A                          469   LOG_OFFSET:      ; 512*10*LOG(2**(15-n))     n= 0,1,2,3 ... 15
                                  470                    ; 512*10*LOG(0.5)           n= 16,17,18 ... 31
                                  471
    3C0A 4F5A4A54454E3F48         472   DCW        23119, 21578, 20037, 18495, 16954, 15413, 13871, 12330
    3C1A 252A20241A1E1518         473   DCW        10789,  9248,  7706,  6165,  4624,  3083,  1541,     0
                                  474
    3C2A                          475   END

ASSEMBLY COMPLETED,    NO ERROR(S) FOUND.
```

270189-44

Listing 2—ASM96 FFT Program (Continued)

The BASIC program is used as comments in the ASM96 program. Some of the variables in the ASM96 program have slightly different names than their counter-parts in the BASIC program. This was to make the comments fit into the ASM96 code. Highlights in this section of code are a table driven square root routine and log conversion routine which can easily be adapted for use by any program.

Both the square root routine and the log conversion routine use the 32-bit value in the variable TMPR. The square root routine calculates the square root of that value in the variable SQRT+2, a 16-bit variable. In this program, the square root value is averaged and stored in a table.

The log conversion routine divides the value in TMPR by 65536 (2^{16}) and uses table lookup to provide the common log. The result is a 16-bit number with the value 512 * 10 Log (TMPR/65536) stored in the variable LOG. This calculation is used to present the results of the FFT in decibels instead of magnitude. With an input of 63095, the output is 512*48 dB. The graph program, (Section 10), prints the output value of the plot as INPUT/512 dB.

The following descriptions of the ASM code point out some of the highlights and not-so-obvious coding:

Lines 1-104 initialize the code and declare variables. The input and output arrays of the program are declared external. Note that many of the registers are overlayable, use caution when implementing this routine with others with overlayable registers.

Lines 116–124 calculate the power of W to be used. Note that KPTR is always incremented by 2. The multiple right shift followed by the AND mask creates an even address and the indirect look to the BR (Bit Reversal) table quickly calculates the power PWR.

Lines 130-138 perform the complex multiplications. Since WIP and WRP range from −32767 to +32767, the multiplication is easy to handle. The automatic divide by two which occurs when using the upper word only of the 32-bit result is a feature in this case.

Lines 144-163 use right shifts for a fast divide, then add or subtract the desired variables and store them in the array. Note that the upper word of TMPR and TMPI is used, and the same array is used for both the input and output of the operations.

Lines 165-189 update the loop variables and then check for errors on the complex multiplications and additions. If there are no overflows at this time the data will run smoothly through the rest of the program.

Lines 200-212 load variables with values based on the bit reversed values of pointers.

Lines 214-236 perform additions and subtractions to prepare for the next set of formulas. Note that XITMP and XRTMP are 32-bit values.

Lines 240-260 perform multiplies and summations resulting in 32-bit variables. This saves a bit or two of accuracy. The upper words are then stored as the results.

Lines 263-272 generate the squared magnitude of the harmonic component as a 32-bit value.

Lines 278-310 calculate 10 Log (TMPR/65536). The 32-bit register TMPR is divided by 65536 so that the output range would be reasonable.

First, the number is normalized. (It is shifted left until a 1 is in the most significant bit, the number of shifts required is placed in SHFT__CNT.) If it had to be shifted more than 15 times the output is set to zero.

Next, the most significant BYTE is used as a reference for the look-up table, providing a 16-bit result. The next most significant BYTE is then used to perform linear interpolation between the referenced table value and the one above it. The interpolated value is added to the directly referenced one.

The 16-bit result of this table look-up and interpolation is then added to the Log of the normalization factor, which is also stored in a table. This table look-up approach works fast and only uses 290 bytes of table space.

Lines 321-357 calculate the square root of the 32-bit register TMPR using a table look-up approach.

First, the number is normalized. Next, the most significant BYTE is used as a reference for the look-up table, providing a 16-bit result. The next most significant BYTE is then used to perform linear interpolation between the referenced table value and the one above it. The interpolated value is added to the directly referenced one.

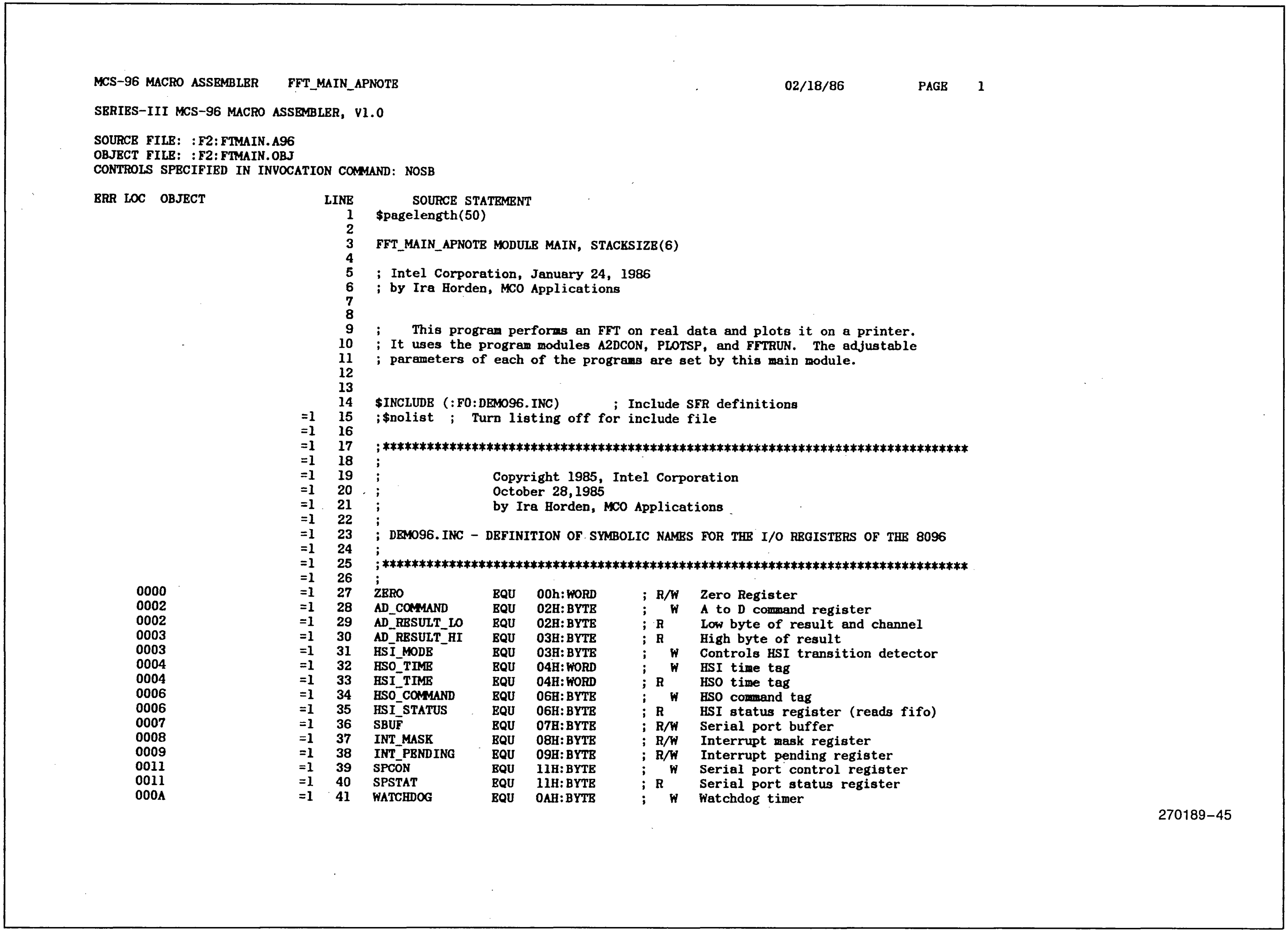
The 16-bit result of this table look-up and interpolation is then divided by the square root of the normalization factor, which is also stored in a table. This table look-up approach works fast and only uses 320 bytes of table space. The results are valid to near 14-bits, more than enough for the FFT algorithm.

Lines 352-360 average the magnitude value, if multiple passes are being performed, and then store the value in the array. The loop-counters are incremented and the process repeats itself.

This concludes the FFT routine. In order to use it, it must be called from a main program. The details for calling this routine are covered in the next section.

8.0 BACKGROUND CONTROL PROGRAM

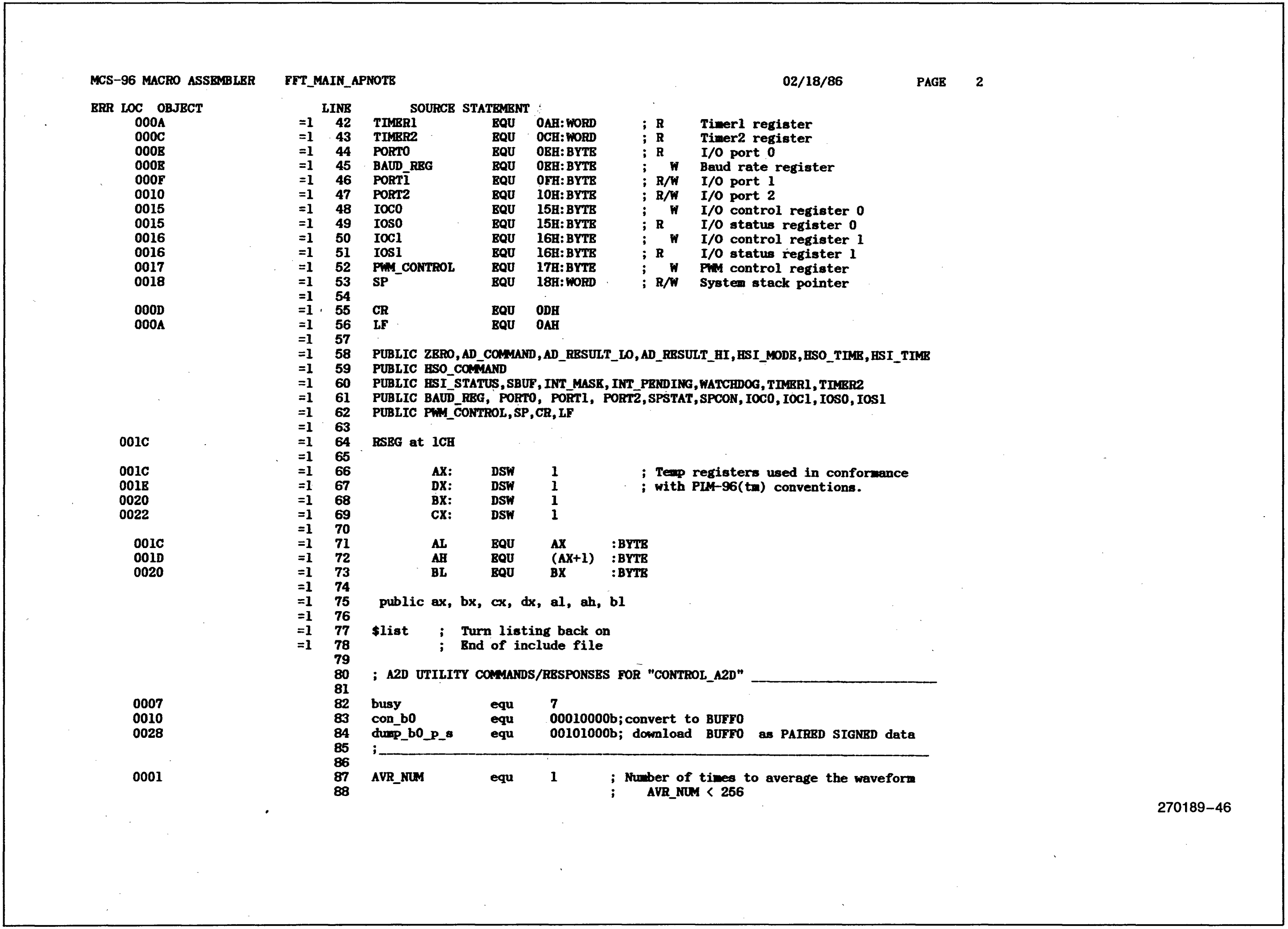
The main routine is shown in Listing 3. It begins with declarations that can be used in almost any program. Note that these are similar, but not identical, to other 8096 include files that have been published. Comments on controlling the Analog to Digital converter routine follow the declarations.



```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                    02/18/86         PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:FTMAIN.A96
OBJECT FILE: :F2:FTMAIN.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                  1   $pagelength(50)
                                  2
                                  3   FFT_MAIN_APNOTE MODULE MAIN, STACKSIZE(6)
                                  4
                                  5   ; Intel Corporation, January 24, 1986
                                  6   ; by Ira Horden, MCO Applications
                                  7
                                  8
                                  9   ;    This program performs an FFT on real data and plots it on a printer.
                                 10   ; It uses the program modules A2DCON, PLOTSP, and FFTRUN.  The adjustable
                                 11   ; parameters of each of the programs are set by this main module.
                                 12
                                 13
                                 14   $INCLUDE (:F0:DEMO96.INC)        ; Include SFR definitions
                             =1  15   ;$nolist  ;  Turn listing off for include file
                             =1  16
                             =1  17   ;*******************************************************************************
                             =1  18   ;
                             =1  19   ;                Copyright 1985, Intel Corporation
                             =1  20   ;                October 28,1985
                             =1  21   ;                by Ira Horden, MCO Applications
                             =1  22   ;
                             =1  23   ; DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096
                             =1  24   ;
                             =1  25   ;*******************************************************************************
                             =1  26   ;
      0000                   =1  27   ZERO             EQU    00h:WORD      ; R/W   Zero Register
      0002                   =1  28   AD_COMMAND       EQU    02H:BYTE      ;   W   A to D command register
      0002                   =1  29   AD_RESULT_LO     EQU    02H:BYTE      ; R     Low byte of result and channel
      0003                   =1  30   AD_RESULT_HI     EQU    03H:BYTE      ; R     High byte of result
      0003                   =1  31   HSI_MODE         EQU    03H:BYTE      ;   W   Controls HSI transition detector
      0004                   =1  32   HSO_TIME         EQU    04H:WORD      ;   W   HSI time tag
      0004                   =1  33   HSI_TIME         EQU    04H:WORD      ; R     HSO time tag
      0006                   =1  34   HSO_COMMAND      EQU    06H:BYTE      ;   W   HSO command tag
      0006                   =1  35   HSI_STATUS       EQU    06H:BYTE      ; R     HSI status register (reads fifo)
      0007                   =1  36   SBUF             EQU    07H:BYTE      ; R/W   Serial port buffer
      0008                   =1  37   INT_MASK         EQU    08H:BYTE      ; R/W   Interrupt mask register
      0009                   =1  38   INT_PENDING      EQU    09H:BYTE      ; R/W   Interrupt pending register
      0011                   =1  39   SPCON            EQU    11H:BYTE      ;   W   Serial port control register
      0011                   =1  40   SPSTAT           EQU    11H:BYTE      ; R     Serial port status register
      000A                   =1  41   WATCHDOG         EQU    0AH:BYTE      ;   W   Watchdog timer
```

270189-45

Listing 3—Main Routine



```
MCS-96 MACRO ASSEMBLER     FFT_MAIN_APNOTE                                          02/18/86          PAGE    2

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
      000A                  =1   42    TIMER1          EQU    0AH:WORD        ; R       Timer1 register
      000C                  =1   43    TIMER2          EQU    0CH:WORD        ; R       Timer2 register
      000E                  =1   44    PORT0           EQU    0EH:BYTE        ; R       I/O port 0
      000E                  =1   45    BAUD_REG        EQU    0EH:BYTE        ;   W     Baud rate register
      000F                  =1   46    PORT1           EQU    0FH:BYTE        ; R/W     I/O port 1
      0010                  =1   47    PORT2           EQU    10H:BYTE        ; R/W     I/O port 2
      0015                  =1   48    IOC0            EQU    15H:BYTE        ;   W     I/O control register 0
      0015                  =1   49    IOS0            EQU    15H:BYTE        ; R       I/O status register 0
      0016                  =1   50    IOC1            EQU    16H:BYTE        ;   W     I/O control register 1
      0016                  =1   51    IOS1            EQU    16H:BYTE        ; R       I/O status register 1
      0017                  =1   52    PWM_CONTROL     EQU    17H:BYTE        ;   W     PWM control register
      0018                  =1   53    SP              EQU    18H:WORD        ; R/W     System stack pointer
                            =1   54
      000D                  =1   55    CR              EQU    0DH
      000A                  =1   56    LF              EQU    0AH
                            =1   57
                            =1   58    PUBLIC ZERO,AD_COMMAND,AD_RESULT_LO,AD_RESULT_HI,HSI_MODE,HSO_TIME,HSI_TIME
                            =1   59    PUBLIC HSO_COMMAND
                            =1   60    PUBLIC HSI_STATUS,SBUF,INT_MASK,INT_PENDING,WATCHDOG,TIMER1,TIMER2
                            =1   61    PUBLIC BAUD_REG, PORT0, PORT1, PORT2,SPSTAT,SPCON,IOC0,IOC1,IOS0,IOS1
                            =1   62    PUBLIC PWM_CONTROL,SP,CR,LF
                            =1   63
    001C                    =1   64    RSEG at 1CH
                            =1   65
    001C                    =1   66            AX:     DSW     1            ; Temp registers used in conformance
    001E                    =1   67            DX:     DSW     1            ; with PLM-96(tm) conventions.
    0020                    =1   68            BX:     DSW     1
    0022                    =1   69            CX:     DSW     1
                            =1   70
      001C                  =1   71            AL      EQU     AX      :BYTE
      001D                  =1   72            AH      EQU     (AX+1)  :BYTE
      0020                  =1   73            BL      EQU     BX      :BYTE
                            =1   74
                            =1   75     public ax, bx, cx, dx, al, ah, bl
                            =1   76
                            =1   77    $list    ;  Turn listing back on
                            =1   78             ;  End of include file
                                 79
                                 80    ; A2D UTILITY COMMANDS/RESPONSES FOR "CONTROL_A2D" _______________________
                                 81
      0007                       82    busy            equ     7
      0010                       83    con_b0          equ     00010000b;convert to BUFF0
      0028                       84    dump_b0_p_s     equ     00101000b; download  BUFF0  as PAIRED SIGNED data
                                 85    ;____________________________________________________________________________
                                 86
      0001                       87    AVR_NUM         equ     1        ; Number of times to average the waveform
                                 88                                     ;     AVR_NUM < 256
```

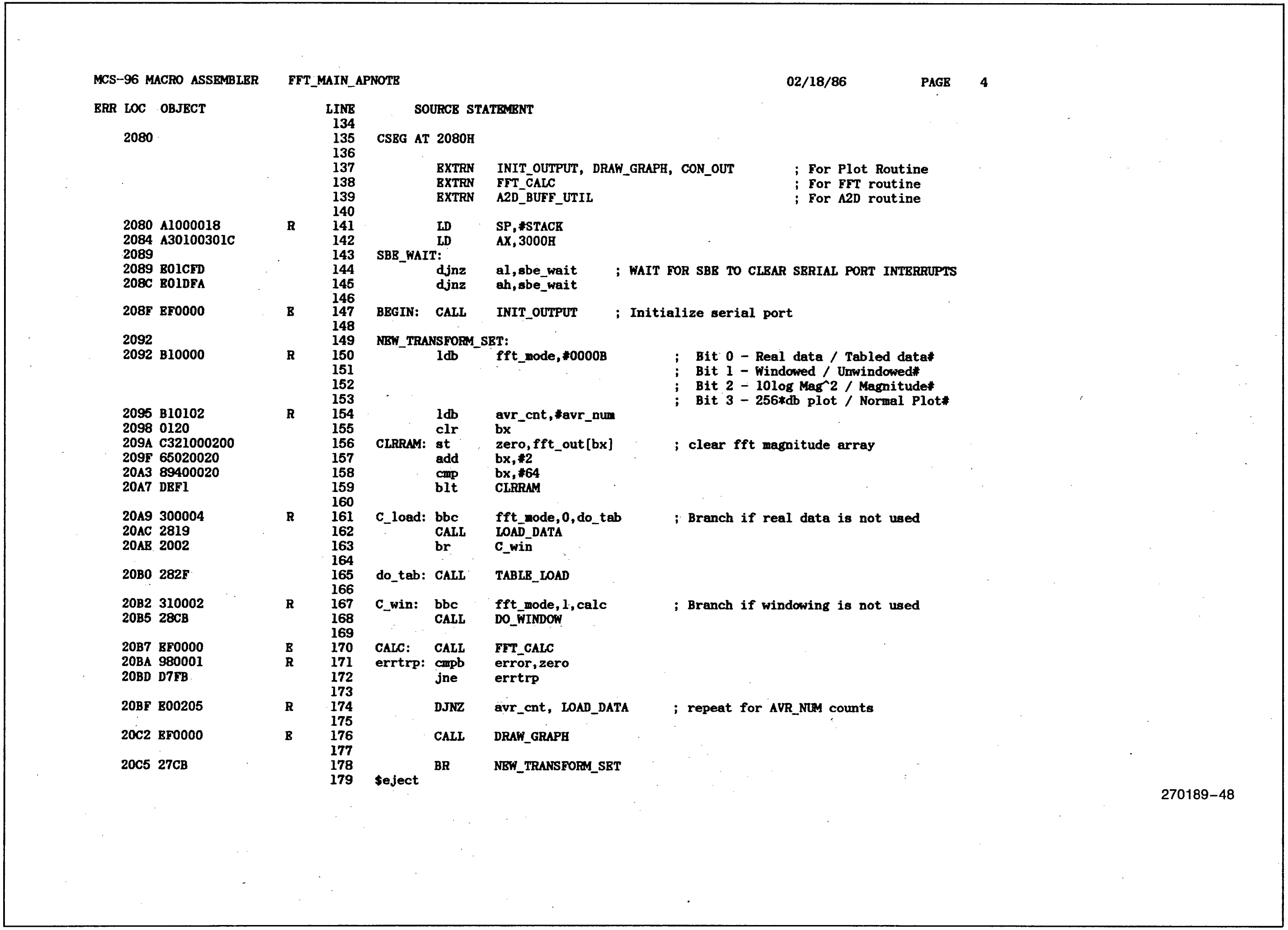
270189-46

Listing 3—Main Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                          02/18/86          PAGE    3

ERR LOC  OBJECT                 LINE       SOURCE STATEMENT
                                  89
     0000                         90   SCALE_FACTOR    equ     0       ; Number of rights shifts performed on
                                  91                                   ; output of FFT.  Used to prevent overflow
                                  92                                   ; on summation
                                  93
     0100                         94   PLOT_RES        equ     256             ; Number of input units per plot unit
     0080                         95   PLOT_RES_2      equ     plot_res/2
     9100                         96   PLOT_MAX        equ     plot_res*145    ; 145 chrs/row
                                  97
                                  98    PUBLIC scale_factor, plot_res, plot_res_2, plot_max
                                  99
                                 100
   0024                          101   OSEG at 24H     ; common oseg area
                                 102
   0024                          103           tmpreal:        dsl     1
   0028                          104           tmpimag:        dsl     1
   002C                          105           wndptr:         dsw     1
   002E                          106           varptr:         dsw     1
                                 107
   0000                          108   RSEG
   0000                          109           fft_mode:       dsb     1
   0001                          110           error:          dsb     1
   0002                          111           avr_cnt:        dsb     1
                                 112    PUBLIC error, fft_mode
                                 113
                                 114           EXTRN   sample_period, control_a2d
                                 115
                                 116
   0080                          117   DSEG at 80h
   0080                          118           XREAL:                          ; For FFT routine
   0080                          119           DEST_BUFF_BASE: DSW     64      ; For A2D routine
     00C0                        120           XIMAG equu      XREAL+64        ; For FFT routine
                                 121
                                 122    PUBLIC DEST_BUFF_BASE, XREAL, XIMAG
                                 123
                                 124
   0200                          125   DSEG AT 200H
                                 126
   0200                          127           PLOT_IN:
   0200                          128           FFT_OUT:        DSW     32      ; For FFT routine
   0240                          129           BUFF0_BASE:     DSW     64      ; For A2D routine
   02C0                          130           BUFF1_BASE:     DSW     64      ; For A2D routine
                                 131
                                 132           PUBLIC  BUFF0_BASE, BUFF1_BASE, FFT_OUT, PLOT_IN
                                 133   $eject
```

270189-47

Listing 3—Main Routine (Continued)



```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                          02/18/86        PAGE    4

ERR LOC  OBJECT              LINE        SOURCE STATEMENT

    2080                          CSEG AT 2080H

                                         EXTRN   INIT_OUTPUT, DRAW_GRAPH, CON_OUT        ; For Plot Routine
                                         EXTRN   FFT_CALC                                ; For FFT routine
                                         EXTRN   A2D_BUFF_UTIL                           ; For A2D routine

    2080 A1000018       R                LD      SP,#STACK
    2084 A30100301C                      LD      AX,3000H
    2089                          SBE_WAIT:
    2089 E01CFD                          djnz    al,sbe_wait     ; WAIT FOR SBE TO CLEAR SERIAL PORT INTERRUPTS
    208C E01DFA                          djnz    ah,sbe_wait

    208F EF0000         E         BEGIN: CALL    INIT_OUTPUT     ; Initialize serial port

    2092                          NEW_TRANSFORM_SET:
    2092 B10000         R                ldb     fft_mode,#0000B         ;  Bit 0 - Real data / Tabled data#
                                                                         ;  Bit 1 - Windowed / Unwindowed#
                                                                         ;  Bit 2 - 10log Mag^2 / Magnitude#
                                                                         ;  Bit 3 - 256*db plot / Normal Plot#
    2095 B10102         R                ldb     avr_cnt,#avr_num
    2098 0120                            clr     bx
    209A C321000200               CLRRAM: st     zero,fft_out[bx]        ; clear fft magnitude array
    209F 65020020                        add     bx,#2
    20A3 89400020                        cmp     bx,#64
    20A7 DEF1                            blt     CLRRAM

    20A9 300004         R         C_load: bbc    fft_mode,0,do_tab       ; Branch if real data is not used
    20AC 2819                            CALL    LOAD_DATA
    20AE 2002                            br      C_win

    20B0 282F                     do_tab: CALL   TABLE_LOAD

    20B2 310002         R         C_win:  bbc    fft_mode,1,calc         ; Branch if windowing is not used
    20B5 28CB                            CALL    DO_WINDOW

    20B7 EF0000         E         CALC:  CALL    FFT_CALC
    20BA 980001         R         errtrp: cmpb   error,zero
    20BD D7FB                            jne     errtrp

    20BF E00205         R                DJNZ    avr_cnt, LOAD_DATA      ; repeat for AVR_NUM counts

    20C2 EF0000         E                CALL    DRAW_GRAPH

    20C5 27CB                            BR      NEW_TRANSFORM_SET
                                  $eject
```

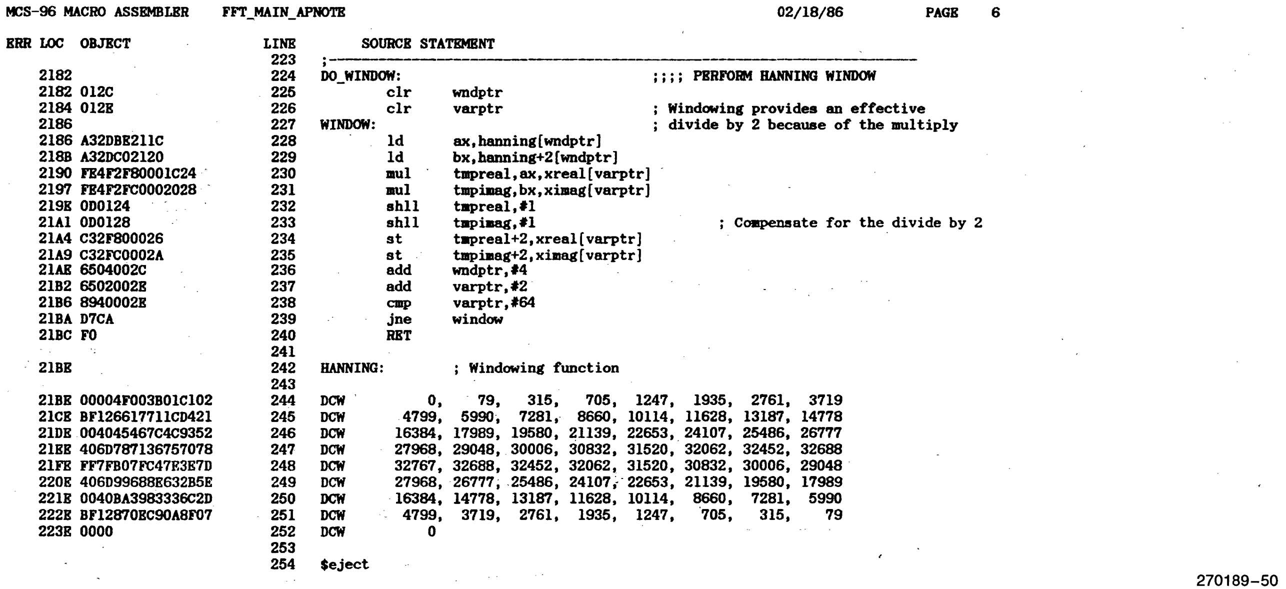
270189-48

Listing 3—Main Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                    02/18/86          PAGE    5

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                180   ;---------------------------------------------------------------------
    20C7                        181   LOAD_DATA:                        ;;;;    LOAD DATA INTO RAM
                                182
    20C7 B1000F                 183          ldb     port1,#00                        ;**** FOR INDICATION ONLY
                                184
    20CA                        185   SET_A2D:
    20CA B11000           E     186          ldb     control_a2d,#con_b0     ; Set converter for buffer0
    20CD 910100           E     187          orb     control_a2d,#01         ; Convert channel 1
    20D0 A1320000         E     188          ld      sample_period,#50       ; 100 us sample period
                                189
    20D4 EF0000           E     190          CALL    a2d_buff_util           ; Start the conversion process
    20D7 3F00FD           E     191          jbs     control_a2d,busy,$      ; wait for all conversions to be done
                                192
    20DA                        193   Down_load:
    20DA B12800           E     194          ldb     control_a2d,#dump_b0_p_s        ; download b0 paired/signed
    20DD EF0000           E     195          CALL    a2d_buff_util
    20E0 F0                     196          RET
                                197
                                198   ;---------------------------------------------------------------------
    20E1                        199   TABLE_LOAD:
    20E1 0120                   200          clr     bx
    20E3 A102211C               201          ld      ax,#DATA0        ; Load tabled data for testing
    20E7 A21D22                 202   load:  ld      cx,[ax]+
    20EA A21D1E                 203          ld      dx,[ax]+
    20ED C321800022             204          st      cx,xreal[bx]
    20F2 C321C0001E             205          st      dx,ximag[bx]
    20F7 65020020               206          add     bx,#2
    20FB 89400020               207          cmp     bx,#64
    20FF DEE6                   208          blt     LOAD
    2101 F0                     209          RET
                                210
    2102                        211   DATA0:             ; SQUARE WAVE
                                212
    2102 FF7FFF7FFF7FFF7F       213   DCW     32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2112 FF7FFF7FFF7FFF7F       214   DCW     32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2122 FF7FFF7FFF7FFF7F       215   DCW     32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2132 FF7FFF7FFF7FFF7F       216   DCW     32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2142 0180018001800180       217   DCW    -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
    2152 0180018001800180       218   DCW    -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
    2162 0180018001800180       219   DCW    -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
    2172 0180018001800180       220   DCW    -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
                                221
                                222   $eject
```

270189–49

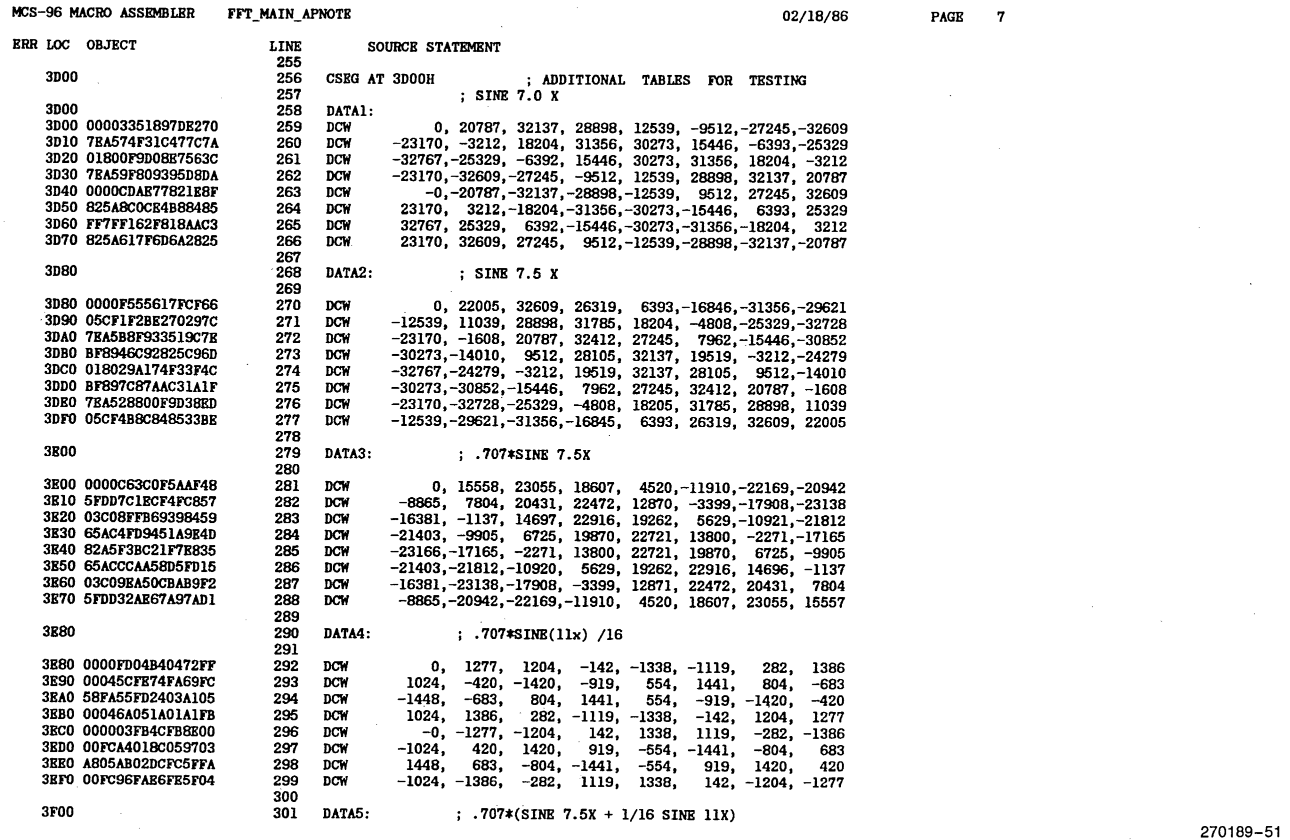
Listing 3—Main Routine (Continued)



```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                    02/18/86          PAGE    6

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                223   ;-----------------------------------------------------------------------
    2182                        224   DO_WINDOW:                              ;;;; PERFORM HANNING WINDOW
    2182 012C                   225           clr     wndptr
    2184 012E                   226           clr     varptr                  ; Windowing provides an effective
    2186                        227   WINDOW:                                 ; divide by 2 because of the multiply
    2186 A32DBE211C             228           ld      ax,hanning[wndptr]
    218B A32DC02120             229           ld      bx,hanning+2[wndptr]
    2190 FE4F2F80001C24         230           mul     tmpreal,ax,xreal[varptr]
    2197 FE4F2FC0002028         231           mul     tmpimag,bx,ximag[varptr]
    219E 0D0124                 232           shll    tmpreal,#1
    21A1 0D0128                 233           shll    tmpimag,#1                      ; Compensate for the divide by 2
    21A4 C32F800026             234           st      tmpreal+2,xreal[varptr]
    21A9 C32FC0002A             235           st      tmpimag+2,ximag[varptr]
    21AE 6504002C               236           add     wndptr,#4
    21B2 6502002E               237           add     varptr,#2
    21B6 8940002E               238           cmp     varptr,#64
    21BA D7CA                   239           jne     window
    21BC F0                     240           RET
                                241
    21BE                        242   HANNING:        ; Windowing function
                                243
    21BE 00004F003B01C102       244   DCW          0,    79,   315,   705,  1247,  1935,  2761,  3719
    21CE BF126617711CD421       245   DCW       4799,  5990,  7281,  8660, 10114, 11628, 13187, 14778
    21DE 004045467C4C9352       246   DCW      16384, 17989, 19580, 21139, 22653, 24107, 25486, 26777
    21EE 406D787136757078       247   DCW      27968, 29048, 30006, 30832, 31520, 32062, 32452, 32688
    21FE FF7FB07FC47E3E7D       248   DCW      32767, 32688, 32452, 32062, 31520, 30832, 30006, 29048
    220E 406D99688E632B5E       249   DCW      27968, 26777, 25486, 24107, 22653, 21139, 19580, 17989
    221E 0040BA3983336C2D       250   DCW      16384, 14778, 13187, 11628, 10114,  8660,  7281,  5990
    222E BF12870EC90A8F07       251   DCW       4799,  3719,  2761,  1935,  1247,   705,   315,    79
    223E 0000                   252   DCW          0
                                253
                                254   $eject
```

270189-50

Listing 3—Main Routine (Continued)



```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                          02/18/86          PAGE    7

ERR LOC  OBJECT                 LINE       SOURCE STATEMENT
                                 255
    3D00                         256   CSEG AT 3D00H              ; ADDITIONAL  TABLES  FOR  TESTING
                                 257                       ; SINE 7.0 X
    3D00                         258   DATA1:
    3D00 00003351897DE270        259   DCW          0, 20787, 32137, 28898, 12539, -9512,-27245,-32609
    3D10 7EA574F31C477C7A        260   DCW     -23170, -3212, 18204, 31356, 30273, 15446, -6393,-25329
    3D20 01800F9D08E7563C        261   DCW     -32767,-25329, -6392, 15446, 30273, 31356, 18204, -3212
    3D30 7EA59F809395D8DA        262   DCW     -23170,-32609,-27245, -9512, 12539, 28898, 32137, 20787
    3D40 0000CDAE77821E8F        263   DCW         -0,-20787,-32137,-28898,-12539,  9512, 27245, 32609
    3D50 825A8C0CE4B88485        264   DCW      23170,  3212,-18204,-31356,-30273,-15446,  6393, 25329
    3D60 FF7FF162F818AAC3        265   DCW      32767, 25329,  6392,-15446,-30273,-31356,-18204,  3212
    3D70 825A617F6D6A2825        266   DCW      23170, 32609, 27245,  9512,-12539,-28898,-32137,-20787
                                 267
    3D80                         268   DATA2:              ; SINE 7.5 X
                                 269
    3D80 0000F555617FCF66        270   DCW          0, 22005, 32609, 26319,  6393,-16846,-31356,-29621
    3D90 05CF1F2BE270297C        271   DCW     -12539, 11039, 28898, 31785, 18204, -4808,-25329,-32728
    3DA0 7EA5B8F933519C7E        272   DCW     -23170, -1608, 20787, 32412, 27245,  7962,-15446,-30852
    3DB0 BF8946C92825C96D        273   DCW     -30273,-14010,  9512, 28105, 32137, 19519, -3212,-24279
    3DC0 018029A174F33F4C        274   DCW     -32767,-24279, -3212, 19519, 32137, 28105,  9512,-14010
    3DD0 BF897C87AAC31A1F        275   DCW     -30273,-30852,-15446,  7962, 27245, 32412, 20787, -1608
    3DE0 7EA528800F9D38ED        276   DCW     -23170,-32728,-25329, -4808, 18205, 31785, 28898, 11039
    3DF0 05CF4B8C848533BE        277   DCW     -12539,-29621,-31356,-16845,  6393, 26319, 32609, 22005
                                 278
    3E00                         279   DATA3:              ; .707*SINE 7.5X
                                 280
    3E00 0000C63C0F5AAF48        281   DCW          0, 15558, 23055, 18607,  4520,-11910,-22169,-20942
    3E10 5FDD7C1ECF4FC857        282   DCW      -8865,  7804, 20431, 22472, 12870, -3399,-17908,-23138
    3E20 03C08FFB69398459        283   DCW     -16381, -1137, 14697, 22916, 19262,  5629,-10921,-21812
    3E30 65AC4FD9451A9E4D        284   DCW     -21403, -9905,  6725, 19870, 22721, 13800, -2271,-17165
    3E40 82A5F3BC21F7E835        285   DCW     -23166,-17165, -2271, 13800, 22721, 19870,  6725, -9905
    3E50 65ACCCAA58D5FD15        286   DCW     -21403,-21812,-10920,  5629, 19262, 22916, 14696, -1137
    3E60 03C09EA50CBAB9F2        287   DCW     -16381,-23138,-17908, -3399, 12871, 22472, 20431,  7804
    3E70 5FDD32AE67A97AD1        288   DCW      -8865,-20942,-22169,-11910,  4520, 18607, 23055, 15557
                                 289
    3E80                         290   DATA4:              ; .707*SINE(11x) /16
                                 291
    3E80 0000FD04B40472FF        292   DCW          0,  1277,  1204,  -142, -1338, -1119,   282,  1386
    3E90 00045CFE74FA69FC        293   DCW       1024,  -420, -1420,  -919,   554,  1441,   804,  -683
    3EA0 58FA55FD2403A105        294   DCW      -1448,  -683,   804,  1441,   554,  -919, -1420,  -420
    3EB0 00046A051A01A1FB        295   DCW       1024,  1386,   282, -1119, -1338,  -142,  1204,  1277
    3EC0 000003FB4CFB8E00        296   DCW         -0, -1277, -1204,   142,  1338,  1119,  -282, -1386
    3ED0 00FCA4018C059703        297   DCW      -1024,   420,  1420,   919,  -554, -1441,  -804,   683
    3EE0 A805AB02DCFC5FFA        298   DCW       1448,   683,  -804, -1441,  -554,   919,  1420,   420
    3EF0 00FC96FAE6FE5F04        299   DCW      -1024, -1386,  -282,  1119,  1338,   142, -1204, -1277
                                 300
    3F00                         301   DATA5:              ; .707*(SINE 7.5X + 1/16 SINE 11X)
```

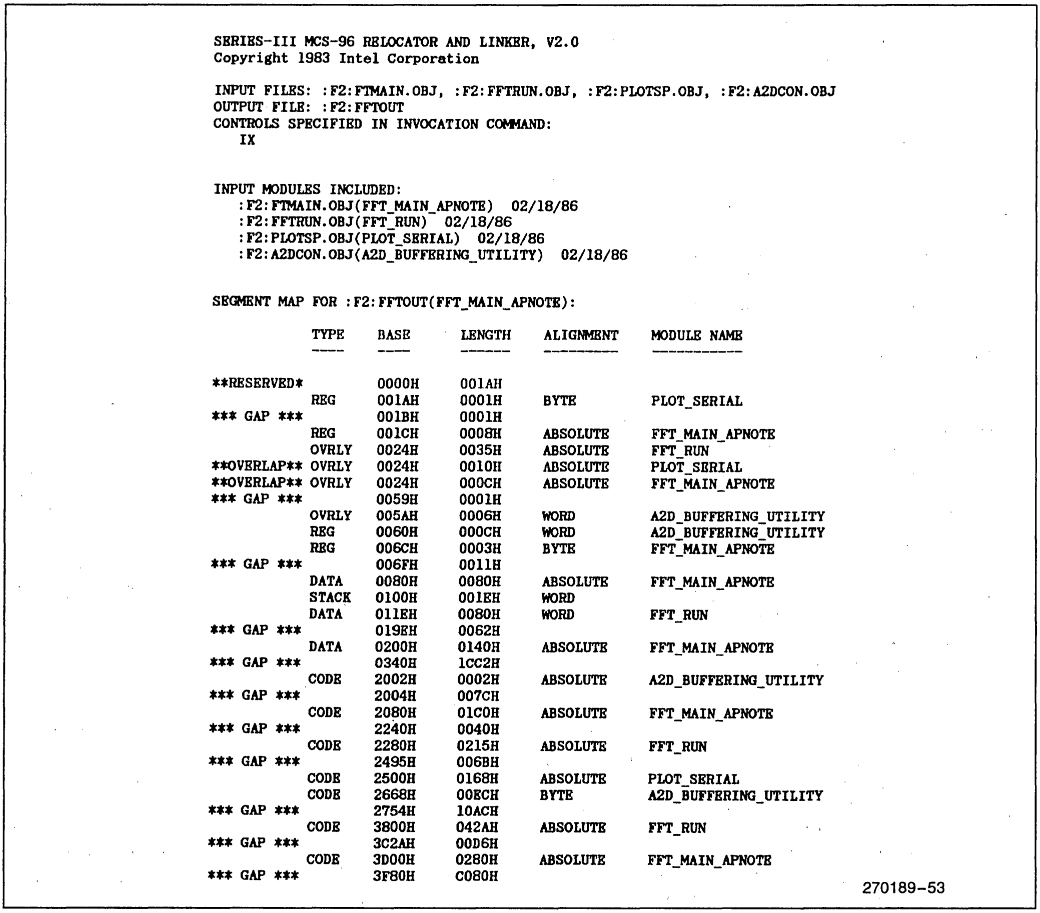
270189-51

Listing 3—Main Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                                   02/18/86          PAGE    8

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                302
    3F00 0000C241C35E2148       303    DCW        0, 16834, 24259, 18465,  3182,-13029,-21886,-19557
    3F10 5EE1D81C434A3154       304    DCW    -7842,  7384, 19011, 21553, 13425, -1958,-17103,-23821
    3F20 5BBAE5F88D3C245F       305    DCW   -17829, -1819, 15501, 24356, 19816,  4710,-12341,-22232
    3F30 65B0B9DE5F1B3F49       306    DCW   -20379, -8519,  7007, 18751, 21383, 13658, -1067,-15888
    3F40 82A5F6B76DF27636       307    DCW   -23166,-18442, -3475, 13942, 24059, 20990,  6442,-11290
    3F50 65A870ACE4DA9419       308    DCW   -22427,-21392, -9500,  6548, 18708, 21475, 13892,  -454
    3F60 ABC548A8E8B618ED       309    DCW   -14933,-22456,-18712, -4840, 12317, 23391, 21851,  8225
    3F70 5FD9C8A84DA8D9D5       310    DCW    -9889,-22328,-22451,-10791,  5857, 18749, 21851, 14281
                                311
                                312
    3F80                        313    END

ASSEMBLY COMPLETED,    NO ERROR(S) FOUND.
```

270189-52

Listing 3—Main Routine (Continued)



```
SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation

INPUT FILES: :F2:FTMAIN.OBJ, :F2:FFTRUN.OBJ, :F2:PLOTSP.OBJ, :F2:A2DCON.OBJ
OUTPUT FILE: :F2:FFTOUT
CONTROLS SPECIFIED IN INVOCATION COMMAND:
   IX


INPUT MODULES INCLUDED:
   :F2:FTMAIN.OBJ(FFT_MAIN_APNOTE)  02/18/86
   :F2:FFTRUN.OBJ(FFT_RUN)  02/18/86
   :F2:PLOTSP.OBJ(PLOT_SERIAL)  02/18/86
   :F2:A2DCON.OBJ(A2D_BUFFERING_UTILITY)  02/18/86


SEGMENT MAP FOR :F2:FFTOUT(FFT_MAIN_APNOTE):

            TYPE    BASE     LENGTH    ALIGNMENT    MODULE NAME
            ----    ----     ------    ---------    -----------

**RESERVED*         0000H    001AH
            REG     001AH    0001H     BYTE         PLOT_SERIAL
*** GAP ***         001BH    0001H
            REG     001CH    0008H     ABSOLUTE     FFT_MAIN_APNOTE
            OVRLY   0024H    0035H     ABSOLUTE     FFT_RUN
**OVERLAP** OVRLY   0024H    0010H     ABSOLUTE     PLOT_SERIAL
**OVERLAP** OVRLY   0024H    000CH     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***         0059H    0001H
            OVRLY   005AH    0006H     WORD         A2D_BUFFERING_UTILITY
            REG     0060H    000CH     WORD         A2D_BUFFERING_UTILITY
            REG     006CH    0003H     BYTE         FFT_MAIN_APNOTE
*** GAP ***         006FH    0011H
            DATA    0080H    0080H     ABSOLUTE     FFT_MAIN_APNOTE
            STACK   0100H    001EH     WORD
            DATA    011EH    0080H     WORD         FFT_RUN
*** GAP ***         019EH    0062H
            DATA    0200H    0140H     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***         0340H    1CC2H
            CODE    2002H    0002H     ABSOLUTE     A2D_BUFFERING_UTILITY
*** GAP ***         2004H    007CH
            CODE    2080H    01C0H     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***         2240H    0040H
            CODE    2280H    0215H     ABSOLUTE     FFT_RUN
*** GAP ***         2495H    006BH
            CODE    2500H    0168H     ABSOLUTE     PLOT_SERIAL
            CODE    2668H    00ECH     BYTE         A2D_BUFFERING_UTILITY
*** GAP ***         2754H    10ACH
            CODE    3800H    042AH     ABSOLUTE     FFT_RUN
*** GAP ***         3C2AH    00D6H
            CODE    3D00H    0280H     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***         3F80H    C080H
```

270189-53

Listing 3—Main Routine (Continued)

Several constants are then setup for other routines. The purpose of centrally locating these constants was the ease of modifying the operation of the routines. Note that AVR_NUM and SCALE_FACTOR must be changed at the same time. SCALE_FACTOR is the shift count used to divide each FFT output value before it is added to the output array. AVR_NUM must be less than 2**SCALE_FACTOR or an overflow could occur. Next, the public variables are declared for the arrays and a few other parameters.

The program then begins by setting the stack pointer and waiting for the SBE-96 to finish talking to the terminal. If this is not done, there may be serial port interrupts occurring for the first twenty five milliseconds of program operation.

Initialization of the plotter is next, followed by setting the FFT_MODE byte. This byte controls the graphing, loading and magnitude calculation of the FFT data. Since FFT_MODE is declared PUBLIC in this module, and EXTERNAL in the PLOT module and FFTRUN module, the extra bits available in this byte can be used for future enhancements.

The next step is to clear the FFT output array. Since the FFT program can be set to average its results by dividing the output before adding it to the magnitude array, the array must be cleared before beginning the program.

Data is then loaded into into the FFT input array by the code at LOAD_DATA, or the code at TABLE_LOAD, depending on the value of FFT_MODE bit 0. The tabled data located at DATA0 is a square wave of magnitude 1. This waveform provides a reasonable test of the FFT algorithm, as many harmonics are generated. The results are also easy to check as the pattern contains half zeros, imaginary values which are always the same, and real values which decrease. Figure 13 shows the output in fractions, hexadecimal and decimal. The hexadecimal and decimal values are based on an output of 16384 being equal to 1.00.

Note that the magnitude is

SQR (REAL^2 + IMAG^2)

and the dB value is

10 LOG ((REAL^2 + IMAG^2)/65536)

The divide by 65536 is used for the dB scale to provide a reasonable range for calculations. If this was not done, a 32-bit LOG function would have been needed.

After the data is loaded, the data is optionally windowed, based on FFT_MODE bit 1, and the FFT program is called. Once the loop has been performed AVR_CNT times, the graph is drawn by the plot routine.

Appended to the main routine is the FFTOUT.M96 Listing. This is provided by the relocator and linker, RL96. With this listing and the main program, it is possible to determine which sections of code are at which addresses.

Using the modular programming methods employed here, it is reasonably easy to debug code. By emulating the program in a relatively high level language, each routine can be checked for functionality against a known standard. The closer the high level implementation matches the ASM96 version, the more possible checkpoints there are between the two routines.

Once all of the program routines (modules) can be shown to work individually, the main program should work unless there is unwanted interaction between the modules. These interactions can be checked by verifying the inputs and outputs of each module. The assembly language locations to perform the program breaks can be retrieved by absolutely locating the main module. The other modules can be dynamically located by RL96.

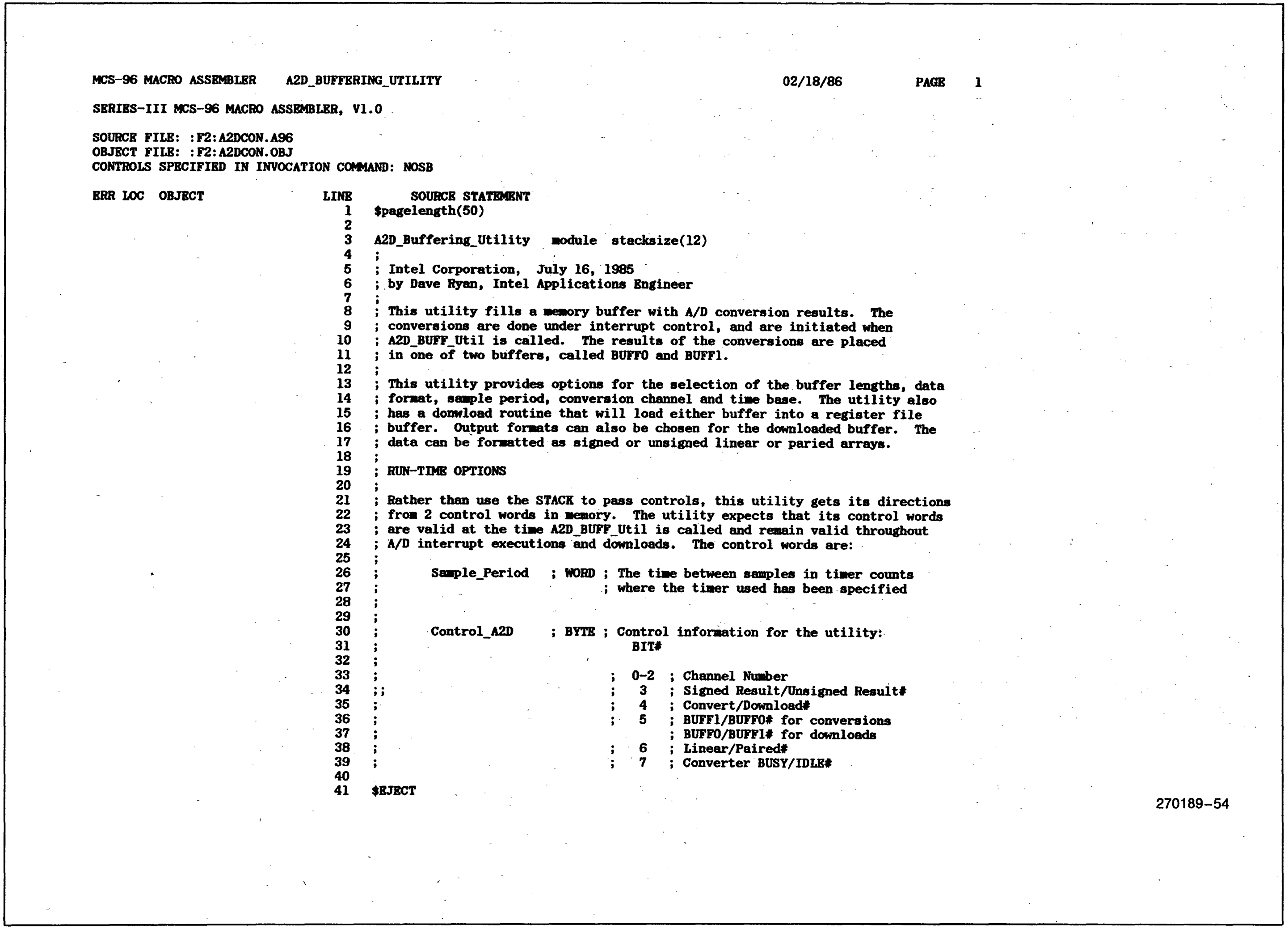
The more interactive program modules are, the more difficult the program becomes to debug. This is especially true when multiple interrupts are occurring, and several of the interrupt routines are themselves interruptable. In these cases, it may be necessary to use debugging equipment with trace capability, like the VLSiCE-96. If this type of equipment is not available, then using I/O ports to indicate the entering and leaving of each routine may be useful. In this way it will be possible to watch the action of the program on an oscilloscope or logic analyzer. There are several places within this code that I/O port toggling has been used as an aid to debugging the program. These lines of code are marked "FOR INDICATION ONLY."

K	Fractional			dB	Decimal			Hexadecimal		
	REAL	IMAG	MAG2		REAL	IMAG	MAG2	REAL	IMAG	MAG2
0	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
1	0.0625	−1.2722	1.2738	38.225	1024	−20843	20868	400	AE95	5184
2	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
3	0.0625	−0.4213	0.4260	28.710	1024	−6903	6978	400	E509	1B42
4	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
5	0.0625	−0.2495	0.2572	24.329	1024	−4088	4214	400	F008	1076
6	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
7	0.0625	−0.1747	0.1855	21.491	1024	−2862	3039	400	F4D2	BDF
8	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
9	0.0625	−0.1321	0.1462	19.421	1024	−2165	2395	400	F78B	95B
10	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
11	0.0625	−0.1043	0.1216	17.820	1024	−1708	1992	400	F954	7C8
12	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
13	0.0625	−0.0843	0.1049	16.540	1024	−1381	1719	400	FA9B	6B7
14	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
15	0.0625	−0.0690	0.0931	15.499	1024	−1130	1525	400	FB96	5F5
16	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
17	0.0625	−0.0566	0.0844	14.645	1024	−928	1382	400	FC60	566
18	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
19	0.0625	−0.0464	0.0778	13.944	1024	−759	1275	400	FD09	4FB
20	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
21	0.0625	−0.0375	0.0729	13.374	1024	−614	1194	400	FD9A	4AA
22	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
23	0.0625	−0.0296	0.0691	12.918	1024	−484	1133	400	FE1C	46D
24	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
25	0.0625	−0.0224	0.0664	12.564	1024	−366	1088	400	FE92	440
26	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
27	0.0625	−0.0157	0.0644	12.305	1024	−256	1056	400	FF00	420
28	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
29	0.0625	−0.0093	0.0632	12.135	1024	−152	1035	400	FF68	40B
30	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
31	0.0625	−0.0031	0.0626	12.051	1024	−50	1025	400	FFCE	401

Figure 13. FFT Output for a Square Wave Input

9.0 ANALOG TO DIGITAL CONVERTER MODULE

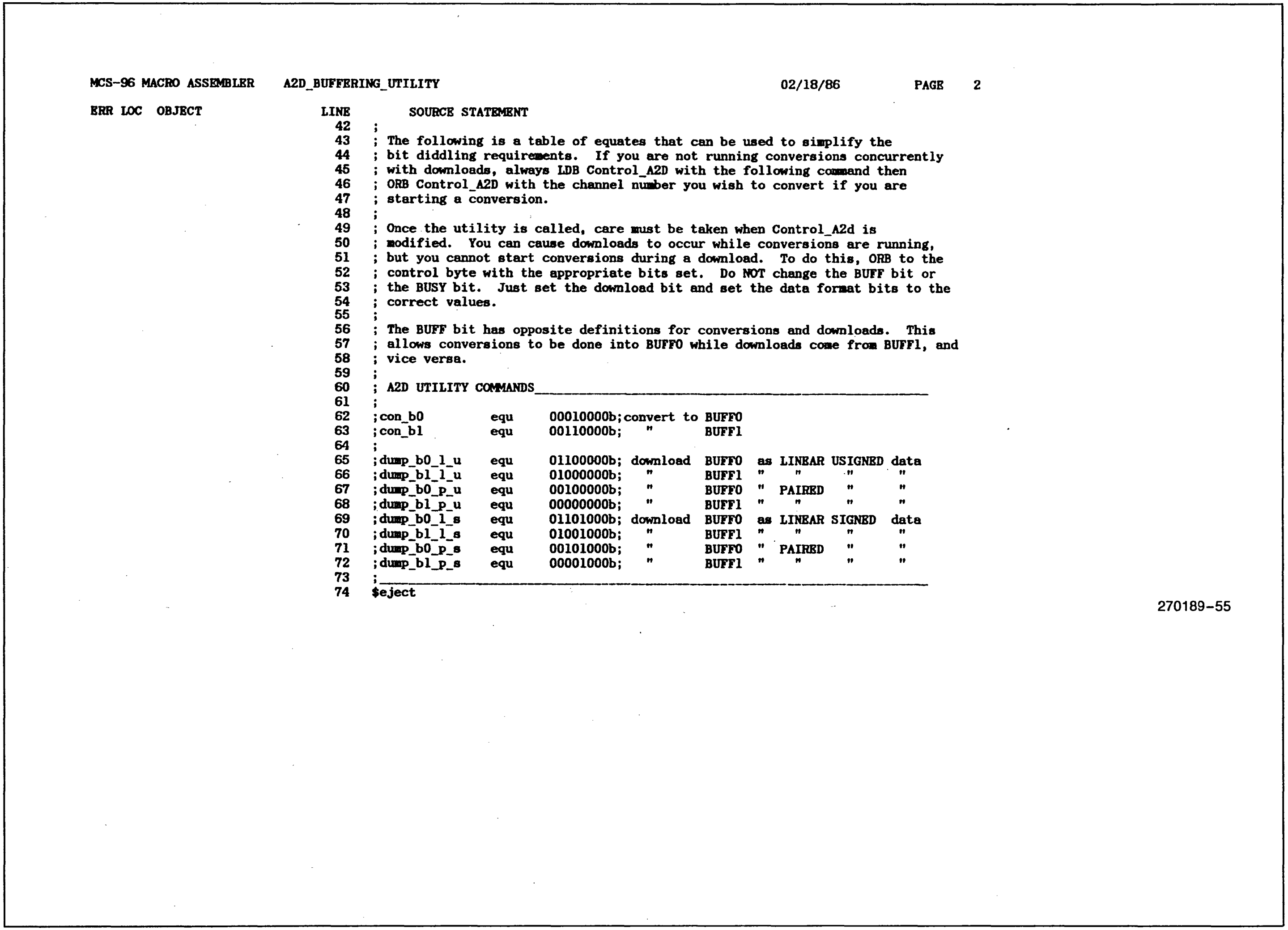
The module presented in Listing 4 is a general purpose one which converts analog values under interrupt control and stores them in one of two buffers. These buffers can then be downloaded to another buffer, such as the input buffer to the FFT program. During downloading, this module can convert the data into signed or unsigned formats, and fill a linear or a paired array. A paired array is like the one used in the FFT transform program. It requires N data points placed alternately in two arrays, one starting at zero and the other at N/2.



```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                                02/18/86         PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:A2DCON.A96
OBJECT FILE: :F2:A2DCON.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                 1   $pagelength(50)
                                 2
                                 3   A2D_Buffering_Utility   module  stacksize(12)
                                 4   ;
                                 5   ; Intel Corporation,  July 16, 1985
                                 6   ; by Dave Ryan, Intel Applications Engineer
                                 7   ;
                                 8   ; This utility fills a memory buffer with A/D conversion results.  The
                                 9   ; conversions are done under interrupt control, and are initiated when
                                10   ; A2D_BUFF_Util is called.  The results of the conversions are placed
                                11   ; in one of two buffers, called BUFF0 and BUFF1.
                                12   ;
                                13   ; This utility provides options for the selection of the buffer lengths, data
                                14   ; format, sample period, conversion channel and time base.  The utility also
                                15   ; has a domwload routine that will load either buffer into a register file
                                16   ; buffer.  Output formats can also be chosen for the downloaded buffer.  The
                                17   ; data can be formatted as signed or unsigned linear or paried arrays.
                                18   ;
                                19   ; RUN-TIME OPTIONS
                                20   ;
                                21   ; Rather than use the STACK to pass controls, this utility gets its directions
                                22   ; from 2 control words in memory.  The utility expects that its control words
                                23   ; are valid at the time A2D_BUFF_Util is called and remain valid throughout
                                24   ; A/D interrupt executions and downloads.  The control words are:
                                25   ;
                                26   ;       Sample_Period   ; WORD ; The time between samples in timer counts
                                27   ;                              ; where the timer used has been specified
                                28   ;
                                29   ;
                                30   ;       Control_A2D     ; BYTE ; Control information for the utility:
                                31   ;                                BIT#
                                32   ;
                                33   ;                               ;  0-2  ; Channel Number
                                34   ;;                              ;   3   ; Signed Result/Unsigned Result#
                                35   ;                               ;   4   ; Convert/Download#
                                36   ;                               ;   5   ; BUFF1/BUFF0# for conversions
                                37   ;                                       ; BUFF0/BUFF1# for downloads
                                38   ;                               ;   6   ; Linear/Paired#
                                39   ;                               ;   7   ; Converter BUSY/IDLE#
                                40
                                41   $EJECT
```

270189-54

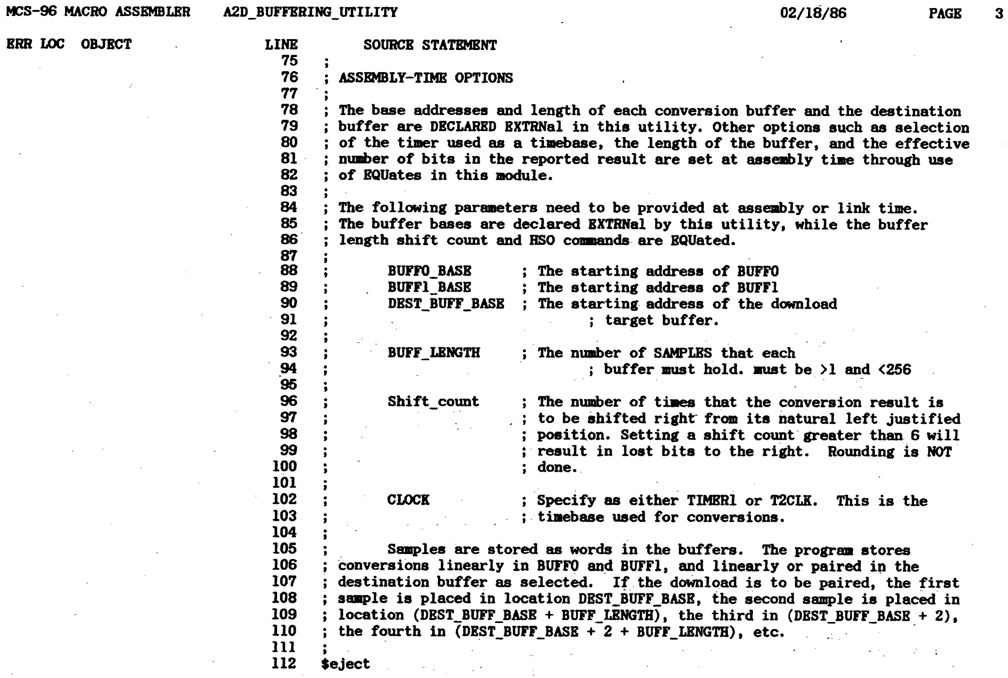
Listing 4—A to D Converter Routine



```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                                02/18/86          PAGE    2

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                         ;
                                         ; The following is a table of equates that can be used to simplify the
                                         ; bit diddling requirements.  If you are not running conversions concurrently
                                         ; with downloads, always LDB Control_A2D with the following command then
                                         ; ORB Control_A2D with the channel number you wish to convert if you are
                                         ; starting a conversion.
                                         ;
                                         ; Once the utility is called, care must be taken when Control_A2d is
                                         ; modified.  You can cause downloads to occur while conversions are running,
                                         ; but you cannot start conversions during a download.  To do this, ORB to the
                                         ; control byte with the appropriate bits set.  Do NOT change the BUFF bit or
                                         ; the BUSY bit.  Just set the download bit and set the data format bits to the
                                         ; correct values.
                                         ;
                                         ; The BUFF bit has opposite definitions for conversions and downloads.  This
                                         ; allows conversions to be done into BUFF0 while downloads come from BUFF1, and
                                         ; vice versa.
                                         ;
                                         ; A2D UTILITY COMMANDS_____________________________________________________
                                         ;
                                         ;con_b0          equ     00010000b;convert to BUFF0
                                         ;con_b1          equ     00110000b;   "       BUFF1
                                         ;
                                         ;dump_b0_l_u     equ     01100000b; download  BUFF0  as LINEAR USIGNED data
                                         ;dump_b1_l_u     equ     01000000b;   "       BUFF1  "     "      "      "
                                         ;dump_b0_p_u     equ     00100000b;   "       BUFF0  "  PAIRED    "      "
                                         ;dump_b1_p_u     equ     00000000b;   "       BUFF1  "     "      "      "
                                         ;dump_b0_l_s     equ     01101000b; download  BUFF0  as LINEAR SIGNED  data
                                         ;dump_b1_l_s     equ     01001000b;   "       BUFF1  "     "      "      "
                                         ;dump_b0_p_s     equ     00101000b;   "       BUFF0  "  PAIRED    "      "
                                         ;dump_b1_p_s     equ     00001000b;   "       BUFF1  "     "      "      "
                                         ;_________________________________________________________________________
                                         $eject
```

270189-55

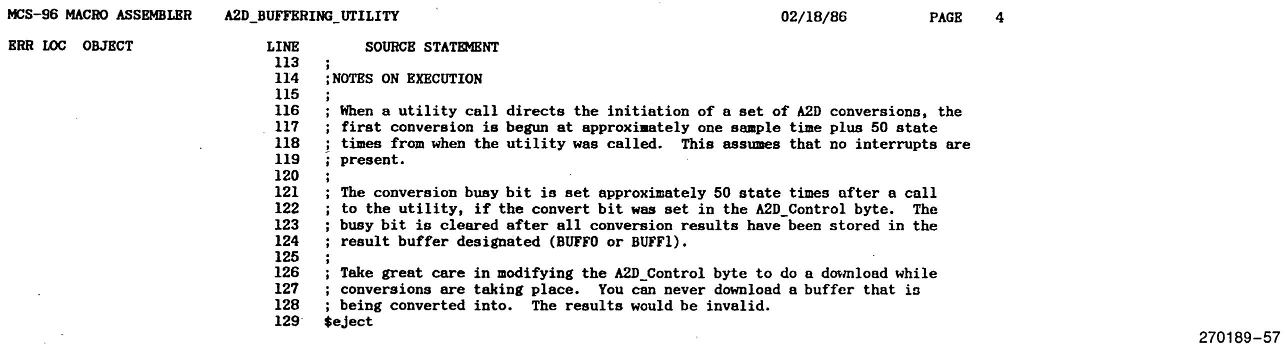
Listing 4—A to D Converter Routine (Continued)



```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                              02/18/86          PAGE    3

ERR LOC  OBJECT                  LINE       SOURCE STATEMENT
                                   75    ;
                                   76    ; ASSEMBLY-TIME OPTIONS
                                   77    ;
                                   78    ; The base addresses and length of each conversion buffer and the destination
                                   79    ; buffer are DECLARED EXTRNal in this utility. Other options such as selection
                                   80    ; of the timer used as a timebase, the length of the buffer, and the effective
                                   81    ; number of bits in the reported result are set at assembly time through use
                                   82    ; of EQUates in this module.
                                   83    ;
                                   84    ; The following parameters need to be provided at assembly or link time.
                                   85    ; The buffer bases are declared EXTRNal by this utility, while the buffer
                                   86    ; length shift count and HSO commands are EQUated.
                                   87    ;
                                   88    ;       BUFF0_BASE      ; The starting address of BUFF0
                                   89    ;       BUFF1_BASE      ; The starting address of BUFF1
                                   90    ;       DEST_BUFF_BASE  ; The starting address of the download
                                   91    ;                               ; target buffer.
                                   92    ;
                                   93    ;       BUFF_LENGTH     ; The number of SAMPLES that each
                                   94    ;                               ; buffer must hold. must be >1 and <256
                                   95    ;
                                   96    ;       Shift_count     ; The number of times that the conversion result is
                                   97    ;                       ; to be shifted right from its natural left justified
                                   98    ;                       ; position. Setting a shift count greater than 6 will
                                   99    ;                       ; result in lost bits to the right.  Rounding is NOT
                                  100    ;                       ; done.
                                  101    ;
                                  102    ;       CLOCK           ; Specify as either TIMER1 or T2CLK.  This is the
                                  103    ;                       ; timebase used for conversions.
                                  104    ;
                                  105    ;       Samples are stored as words in the buffers.  The program stores
                                  106    ; conversions linearly in BUFF0 and BUFF1, and linearly or paired in the
                                  107    ; destination buffer as selected.  If the download is to be paired, the first
                                  108    ; sample is placed in location DEST_BUFF_BASE, the second sample is placed in
                                  109    ; location (DEST_BUFF_BASE + BUFF_LENGTH), the third in (DEST_BUFF_BASE + 2),
                                  110    ; the fourth in (DEST_BUFF_BASE + 2 + BUFF_LENGTH), etc.
                                  111    ;
                                  112    $eject
```

270189-56

Listing 4—A to D Converter Routine (Continued)



```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86        PAGE    4

ERR LOC  OBJECT                 LINE        SOURCE STATEMENT
                                       ;
                                       ;NOTES ON EXECUTION
                                       ;
                                       ; When a utility call directs the initiation of a set of A2D conversions, the
                                       ; first conversion is begun at approximately one sample time plus 50 state
                                       ; times from when the utility was called.  This assumes that no interrupts are
                                       ; present.
                                       ;
                                       ; The conversion busy bit is set approximately 50 state times after a call
                                       ; to the utility, if the convert bit was set in the A2D_Control byte.  The
                                       ; busy bit is cleared after all conversion results have been stored in the
                                       ; result buffer designated (BUFF0 or BUFF1).
                                       ;
                                       ; Take great care in modifying the A2D_Control byte to do a download while
                                       ; conversions are taking place.  You can never download a buffer that is
                                       ; being converted into.  The results would be invalid.
                                       $eject
```

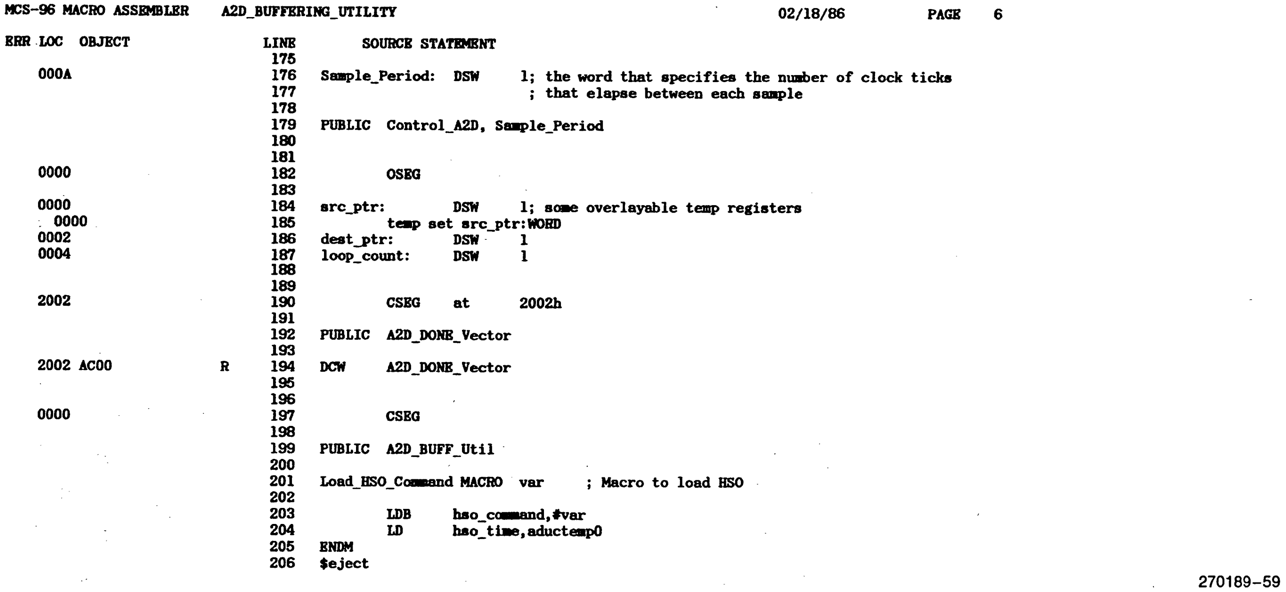
270189-57

Listing 4—A to D Converter Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                                             02/18/86          PAGE    5

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                130
    0000                        131           RSEG
                                132
                                133   EXTRN BUFF0_BASE, BUFF1_BASE, DEST_BUFF_BASE
                                134   EXTRN ad_command, ad_result_lo, ad_result_hi
                                135   EXTRN hso_command, hso_time,sp
                                136
      0040                      137   BUFF_LENGTH    EQU     64
      0001                      138   Shift_Count    EQU     1
      000A                      139   CLOCK          EQU     TIMER1
                                140
                                141   ; set up hso commands for correct timer **********************************
                                142
      000A                      143   TIMER1         equ     0AH
      000C                      144   T2CLK          equ     0CH
                                145
      0000                      146   MASK           equ     (10h*CLOCK)AND(40h)
                                147
      000F                      148   Start_A2D      equ     (00001111b)OR(MASK)
                                149                          ;start a2d based on timer 1, no interrupt
                                150
      0000                      151   HSO_0_Low      equ     (00000000b)OR(MASK)
                                152                          ; make hso.0 low based on timer1 no interrupt
                                153
      0020                      154   HSO_0_High     equ     (00100000b)OR(MASK)
                                155                          ; make hso.0 hi  based on timer1 no interrupt
                                156
                                157
                                158   ; set up storage *******************************************************
                                159
    0000                        160   adudtemp0:     DSW     1; temp register for download calls
                                161
    0002                        162   aductemp0:     DSW     1; temp registers for conversion calls
    0004                        163   aductemp1:     DSW     1
    0006                        164   top_of_buffer: DSW     1
    0008                        165   sample_count:  DSB     1
                                166
    0009                        167   Control_A2D:   DSB     1; the byte that controls the utility execution
      0003                      168                  DForm   equ     3       ; Signed/Unsigned#
      0004                      169                  Con_Dwn equ     4       ; Convert/Download#
      0005                      170                  B0_B1   equ     5       ; Buff1/Buff0# for conversions
                                171                                          ; Buff0/Buff1# for downloads
      0006                      172                  Lin_Par equ     6       ; Linear/Paired#
      0080                      173                  Busy    equ     10000000B       ; Bit 8
                                174   $eject
```

270189-58

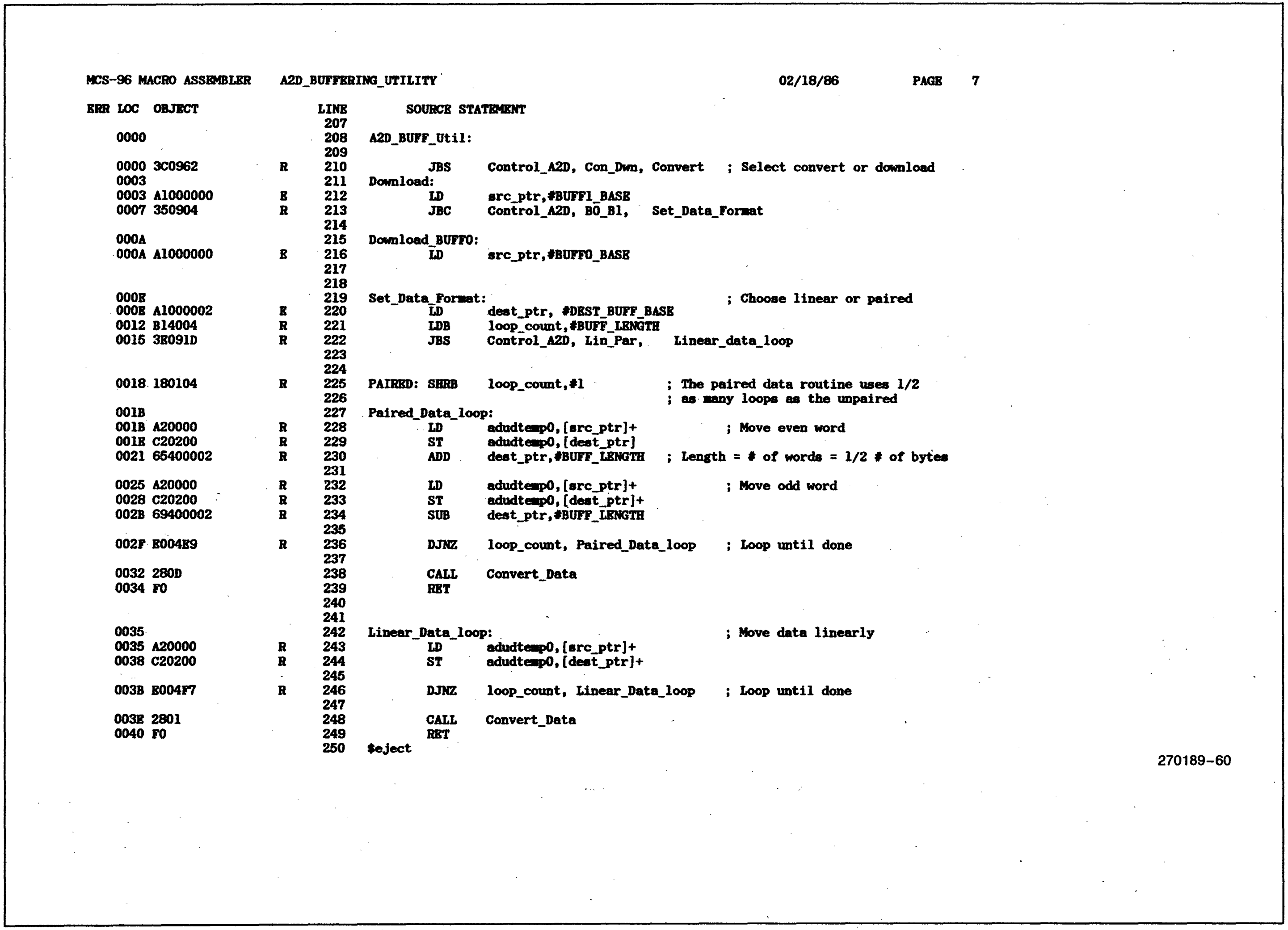
Listing 4—A to D Converter Routine (Continued)



```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                                   02/18/86          PAGE    6

ERR LOC  OBJECT                 LINE        SOURCE STATEMENT
                                 175
    000A                         176   Sample_Period:  DSW     1; the word that specifies the number of clock ticks
                                 177                            ; that elapse between each sample
                                 178
                                 179   PUBLIC  Control_A2D, Sample_Period
                                 180
                                 181
    0000                         182           OSEG
                                 183
    0000                         184   src_ptr:        DSW     1; some overlayable temp registers
      0000                       185           temp set src_ptr:WORD
    0002                         186   dest_ptr:       DSW     1
    0004                         187   loop_count:     DSW     1
                                 188
                                 189
    2002                         190           CSEG    at      2002h
                                 191
                                 192   PUBLIC  A2D_DONE_Vector
                                 193
    2002 AC00             R      194   DCW     A2D_DONE_Vector
                                 195
                                 196
    0000                         197           CSEG
                                 198
                                 199   PUBLIC  A2D_BUFF_Util
                                 200
                                 201   Load_HSO_Command MACRO  var     ; Macro to load HSO
                                 202
                                 203           LDB     hso_command,#var
                                 204           LD      hso_time,aductemp0
                                 205   ENDM
                                 206   $eject
```

270189-59

Listing 4—A to D Converter Routine (Continued)



```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                                   02/18/86        PAGE   7

ERR LOC  OBJECT              LINE        SOURCE STATEMENT
                              207
    0000                      208   A2D_BUFF_Util:
                              209
    0000 3C0962         R     210          JBS     Control_A2D, Con_Dwn, Convert   ; Select convert or download
    0003                      211   Download:
    0003 A1000000       E     212          LD      src_ptr,#BUFF1_BASE
    0007 350904         R     213          JBC     Control_A2D, B0_B1,   Set_Data_Format
                              214
    000A                      215   Download_BUFF0:
    000A A1000000       E     216          LD      src_ptr,#BUFF0_BASE
                              217
                              218
    000E                      219   Set_Data_Format:                           ; Choose linear or paired
    000E A1000002       E     220          LD      dest_ptr, #DEST_BUFF_BASE
    0012 B14004         R     221          LDB     loop_count,#BUFF_LENGTH
    0015 3E091D         R     222          JBS     Control_A2D, Lin_Par,    Linear_data_loop
                              223
                              224
    0018 180104         R     225   PAIRED: SHRB   loop_count,#1           ; The paired data routine uses 1/2
                              226                                          ; as many loops as the unpaired
    001B                      227   Paired_Data_loop:
    001B A20000         R     228          LD      adudtemp0,[src_ptr]+            ; Move even word
    001E C20200         R     229          ST      adudtemp0,[dest_ptr]
    0021 65400002       R     230          ADD     dest_ptr,#BUFF_LENGTH   ; Length = # of words = 1/2 # of bytes
                              231
    0025 A20000         R     232          LD      adudtemp0,[src_ptr]+            ; Move odd word
    0028 C20200         R     233          ST      adudtemp0,[dest_ptr]+
    002B 69400002       R     234          SUB     dest_ptr,#BUFF_LENGTH
                              235
    002F E004E9         R     236          DJNZ    loop_count, Paired_Data_loop    ; Loop until done
                              237
    0032 280D                 238          CALL    Convert_Data
    0034 F0                   239          RET
                              240
                              241
    0035                      242   Linear_Data_loop:                          ; Move data linearly
    0035 A20000         R     243          LD      adudtemp0,[src_ptr]+
    0038 C20200         R     244          ST      adudtemp0,[dest_ptr]+
                              245
    003B E004F7         R     246          DJNZ    loop_count, Linear_Data_loop    ; Loop until done
                              247
    003E 2801                 248          CALL    Convert_Data
    0040 F0                   249          RET
                              250   $eject
```

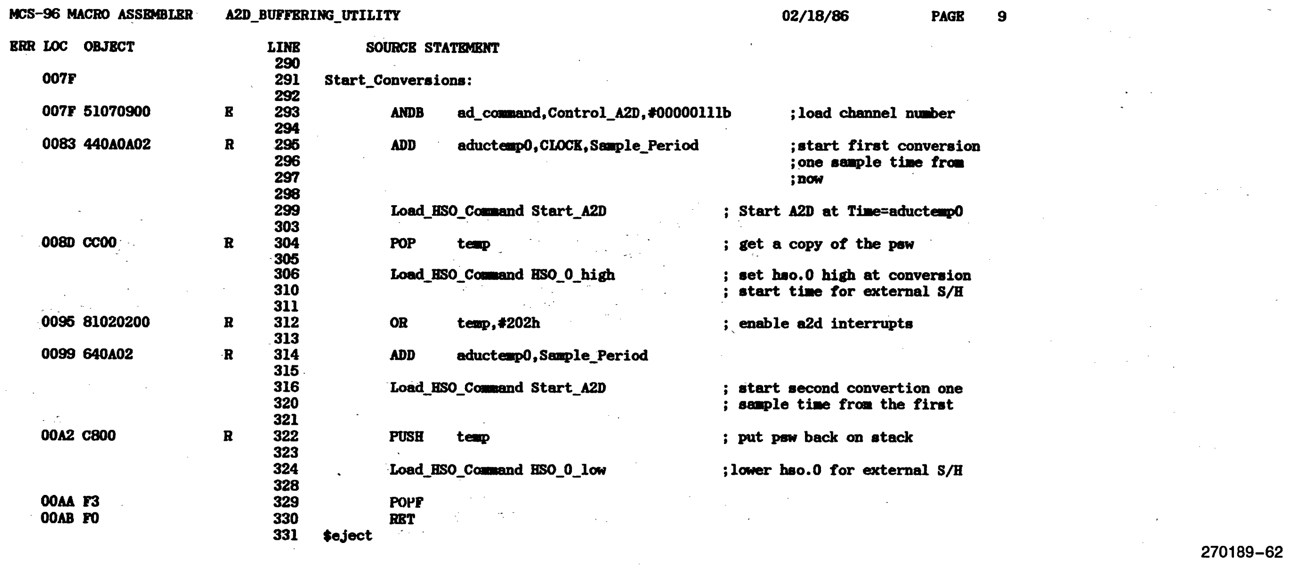
270189-60

Listing 4—A to D Converter Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                                02/18/86        PAGE    8

ERR LOC  OBJECT              LINE       SOURCE STATEMENT
                              251
    0041                      252   Convert_Data:                    ; Convert the data in the destination buffer
                              253
    0041 A1400004        R    254         LD      loop_count,#BUFF_LENGTH
    0045 A1000000        E    255         LD      src_ptr,#DEST_BUFF_BASE
                              256
    0049 A20000          R    257   Again: LD     adudtemp0,[src_ptr]
    004C 71C000          R    258         ANDB    adudtemp0,#11000000b
    004F 330909          R    259         JBC     Control_A2D, DForm, Unsigned_Result
                              260
    0052                      261   Signed_Result:
    0052 69E07F00        R    262         SUB     adudtemp0,#7fe0H
    0056 0A0100          R    263         SHRA    adudtemp0,#Shift_Count
    0059 2003                 264         BR      Replace_Sample
                              265
    005B                      266   Unsigned_Result:
    005B 080100          R    267         SHR     adudtemp0, #Shift_Count
                              268
    005E                      269   Replace_Sample:
    005E C20000          R    270         ST      adudtemp0,[src_ptr]+
    0061 E004E5          R    271         DJNZ    loop_count,Again         ; Loop until done
                              272
    0064 F0                   273         RET
                              274
                              275
    0065                      276   Convert:         ;; Prepare to Start Conversions
                              277
    0065 F2                   278         PUSHF
                              279
    0066 918009          R    280         ORB     Control_A2D, #Busy              ; set converter busy bit
                              281
    0069 B13F08          R    282         LDB     sample_count,#BUFF_LENGTH - 1
    006C A1000006        E    283         LD      top_of_buffer,#BUFF0_BASE
    0070 A1800004        E    284         LD      aductempl,#(BUFF0_BASE + 2*BUFF_LENGTH)
                              285
    0074 350908          R    286         JBC     Control_A2D, B0_B1, Start_Conversions
    0077 A1000006        E    287         LD      top_of_buffer,#BUFF1_BASE
    007B A1800004        E    288         LD      aductempl,#(BUFF1_BASE + 2*BUFF_LENGTH)
                              289   $eject
```

270189-61

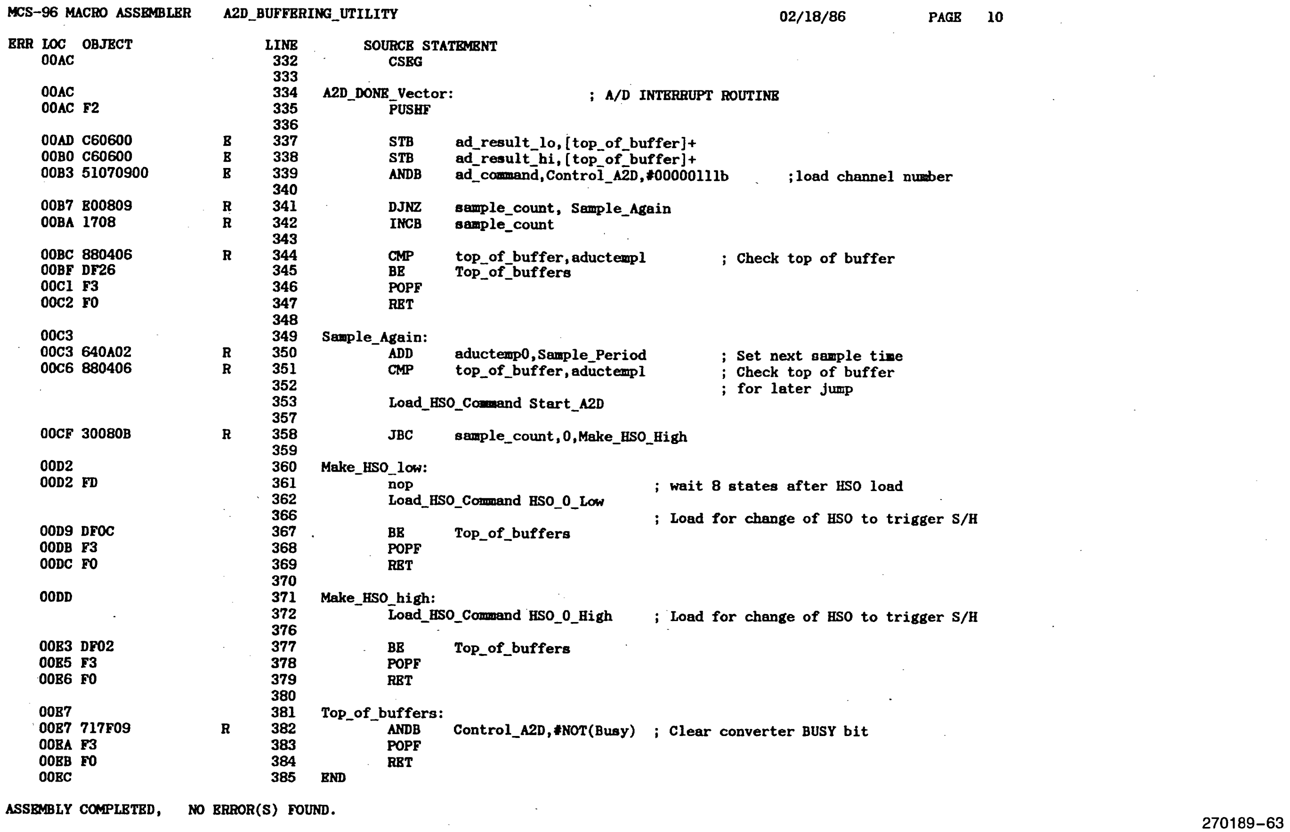
Listing 4—A to D Converter Routine (Continued)



```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                                        02/18/86        PAGE    9

ERR LOC  OBJECT               LINE        SOURCE STATEMENT
                               290
    007F                       291   Start_Conversions:
                               292
    007F 51070900        R     293         ANDB    ad_command,Control_A2D,#00000111b       ;load channel number
                               294
    0083 440A0A02        R     295         ADD     aductemp0,CLOCK,Sample_Period           ;start first conversion
                               296                                                         ;one sample time from
                               297                                                         ;now
                               298
                               299         Load_HSO_Command Start_A2D              ; Start A2D at Time=aductemp0
                               303
    008D CC00            R     304         POP     temp                            ; get a copy of the psw
                               305
                               306         Load_HSO_Command HSO_0_high             ; set hso.0 high at conversion
                               310                                                 ; start time for external S/H
                               311
    0095 81020200        R     312         OR      temp,#202h                      ; enable a2d interrupts
                               313
    0099 640A02          R     314         ADD     aductemp0,Sample_Period
                               315
                               316         Load_HSO_Command Start_A2D              ; start second convertion one
                               320                                                 ; sample time from the first
                               321
    00A2 C800            R     322         PUSH    temp                            ; put psw back on stack
                               323
                               324         Load_HSO_Command HSO_0_low              ;lower hso.0 for external S/H
                               328
    00AA F3                    329         POPF
    00AB F0                    330         RET
                               331   $eject
```

270189-62

Listing 4—A to D Converter Routine (Continued)



```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                                  02/18/86        PAGE   10

ERR LOC  OBJECT              LINE        SOURCE STATEMENT
    00AC                      332            CSEG
                              333
    00AC                      334    A2D_DONE_Vector:                      ; A/D INTERRUPT ROUTINE
    00AC F2                   335            PUSHF
                              336
    00AD C60600          E    337            STB      ad_result_lo,[top_of_buffer]+
    00B0 C60600          E    338            STB      ad_result_hi,[top_of_buffer]+
    00B3 51070900        E    339            ANDB     ad_command,Control_A2D,#00000111b        ;load channel number
                              340
    00B7 E00809          R    341            DJNZ     sample_count, Sample_Again
    00BA 1708            R    342            INCB     sample_count
                              343
    00BC 880406          R    344            CMP      top_of_buffer,aductempl           ; Check top of buffer
    00BF DF26                 345            BE       Top_of_buffers
    00C1 F3                   346            POPF
    00C2 F0                   347            RET
                              348
    00C3                      349    Sample_Again:
    00C3 640A02          R    350            ADD      aductemp0,Sample_Period           ; Set next sample time
    00C6 880406          R    351            CMP      top_of_buffer,aductempl           ; Check top of buffer
                              352                                                      ; for later jump
                              353            Load_HSO_Command Start_A2D
                              357
    00CF 30080B          R    358            JBC      sample_count,0,Make_HSO_High
                              359
    00D2                      360    Make_HSO_low:
    00D2 FD                   361            nop                                ; wait 8 states after HSO load
                              362            Load_HSO_Command HSO_0_Low
                              366                                               ; Load for change of HSO to trigger S/H
    00D9 DF0C                 367            BE       Top_of_buffers
    00DB F3                   368            POPF
    00DC F0                   369            RET
                              370
    00DD                      371    Make_HSO_high:
                              372            Load_HSO_Command HSO_0_High      ; Load for change of HSO to trigger S/H
                              376
    00E3 DF02                 377            BE       Top_of_buffers
    00E5 F3                   378            POPF
    00E6 F0                   379            RET
                              380
    00E7                      381    Top_of_buffers:
    00E7 717F09          R    382            ANDB     Control_A2D,#NOT(Busy)  ; Clear converter BUSY bit
    00EA F3                   383            POPF
    00EB F0                   384            RET
    00EC                      385    END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270189-63

Listing 4—A to D Converter Routine (Continued)

The listing contains a fairly complete description of what the program does. The block by block operations are shown below:

Lines 1-198 describe the program, declare the variables and set up equates. Several of these variables are declared as overlayable, so the user needs to be careful if using this module for other than the FFT program.

Lines 205-210 declare a macro which is used to load the HSO unit. This will be used repeatedly through the code.

Lines 212-253 determine whether a conversion or download has been requested. If a download has been requested, the data is downloaded to the destination array as either paired or linear data. Paired data has been described earlier.

Lines 255-278 contain a subroutine which converts the destination array to either signed or unsigned numbers. The numbers are also shifted right to provide the desired full-scale value as requested by SHIFT__COUNT.

Lines 279-334 initialize the conversion routine. HSO.0 is toggled with the start of each routine so that an external sample and hold can be used. The instructions in lines 308, 316, and 326 have been interweaved with the Load__HSO__Commands to provide the required 8 state delays between HSO loadings. If this was not done, NOPs would have been needed. It is easier to understand the code if these lines are thought of as being gathered at line 326.

Lines 337-353 are the actual A/D interrupt routine. The A/D results are placed BYTE by BYTE on the buffer, the A/D is reloaded, and then the number of samples taken is compared to the number needed. Note that the A/D command register needs to be reloaded even if the channel does not change. INCB on line 348 is used to insure that the DJNZ falls through on the next pass (if sample__count is not reset).

Lines 355-396 complete the routine. The HSO is set up to trigger the next conversion and provide the HSO.0 toggle for an external sample and hold. Once again, the time between consecutive loads of the HSO is 8 states minimum. Note that this section of code has been optimized for speed by reducing branches to an absolute minimum and duplicating code where needed.

This concludes the description of the A to D buffer module. In the FFT program, this module is run, then the FFT transform module, then the plot module. This allows variables to be overlaid, saving RAM space. The time cost for this is not bad, considering the printer is the limiting factor in these conversions. If more RAM was provided, and the FFT was run with its data in external RAM, this module could be run simultaneously with the other modules.

10.0 DATA PLOTTING MODULE

The plot module is relatively straight-forward, and is shown in Listing 5. After the declarations, which include overlayable registers, an initialization routine is listed. This separately called routine sets up the serial port on the 8096 to talk to the printer. In this case, the port has to be set for 300 baud.

A console out routine follows. This routine can also be called by any program, but it is used only by the plot routine in this example. The write to port 1 is used to trace the program flow. The character to be output is passed to this routine on the stack. This conforms to PLM-96 requirements.

Since all stack operations on the 8096 are 16-bits wide, a multiple character feature has been added to the console out routine. If the high byte it receives is non-zero, the ASCII character in that byte is printed after the character in the low byte. If the high byte has a value between 128 and 255, the character in the low byte is repeated the number of times indicated by the least significant 7 bits of the high byte.

The print decimal number routine is next. It is called with two words on the stack. The first word is the unsigned value to be printed. The second byte contains information on the number of places to be printed and zero and blank suppression. This routine is not overflow-proof. The user must declare a sufficient number of places to be printed for all possible numbers.

The DRAW__GRAPH routine provides the plot. It first sends a series of carriage return, line feeds (CRLFs) to clear the printer and provides a margin on the paper. Each row is started with the row number, 2 spaces, and a "+". Asterisks are then plotted until

Number of asterisks > FFT Value / PLOT__RES

Recall that PLOT__RES is a variable set by the main program. When the number of asterisks hits the desired value, the value of the line is printed. If the Decibel mode is selected, the line value is divided by 512 and printed in integer + decimal part form, followed by "dB". If the number of asterisks reaches PLOT__MAX, no value is printed. The next line is then started. A line with only a "!" is printed before the next plot line to provide a more aesthetic display on the printer. If a CRT was used, this extra line would probably not be wanted.

Listing 5—The Plot Module

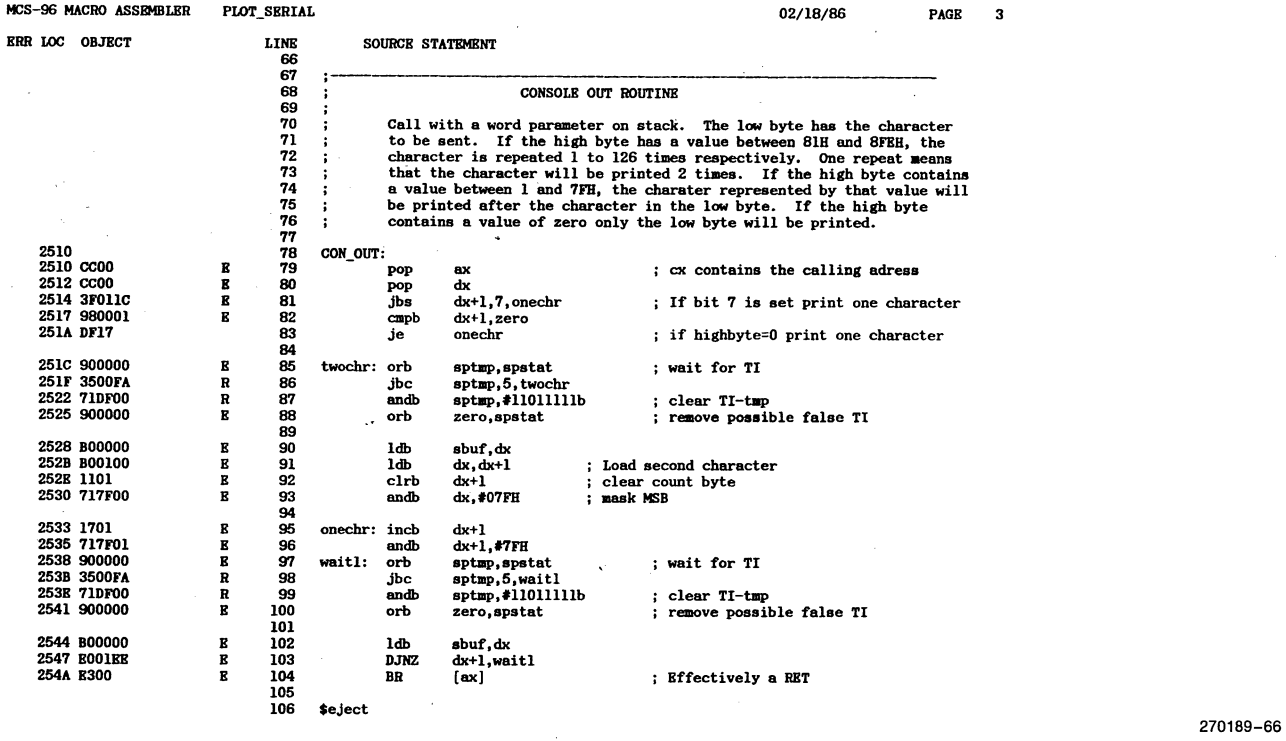
```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                          02/18/86        PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:PLOTSP.A96
OBJECT FILE: :F2:PLOTSP.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT              LINE       SOURCE STATEMENT
                                1   $pagelength(50)
                                2
                                3   PLOT_SERIAL  MODULE STACKSIZE (6)
                                4
                                5   ; Intel Corporation,  December 12, 1985
                                6   ; by Ira Horden, MCO Applications
                                7
                                8   ;     This program produces a plot on serially connected printer.  The
                                9   ; maginitude of each of the 32 input values is plotted horizontally, with one
                               10   ; "!" followed by a linefeed between each plot line.  Each plot line starts
                               11   ; with a "+" and the entire plot begins with 3 line feeds and ends with a form
                               12   ; feed.  The values to be plotted are 32 unsigned words based at the externally
                               13   ; defined pointer PLOT_IN.
                               14
                               15   ;     The routine INIT_OUTPUT must be run to set up the serial port when the
                               16   ; system is turned on.  CON_OUT can be used by a program to output to the
                               17   ; serial port.  DRAW_GRAPH is the routine that automatically plots the data.
                               18
                               19   ;     Sizing of the graph can be done using PLOT_RES, which determines how many
                               20   ; units are needed for each dot, and PLOT_MAX, which is the maximum value the
                               21   ; program will be passed.  Note that (PLOT_MAX/PLOT_RES) defines the maximum
                               22   ; number of columns the routine will print.
                               23   ;
                               24
    0000                       25   RSEG
                               26           EXTRN   iocl, baud_reg, spcon, spstat, sbuf, port1
                               27           EXTRN   zero, ax, bx, cx, dx, FFT_MODE
    0000                       28           sptmp:  dsb     1
                               29
    0024                       30   OSEG at 24H
    0024                       31           value:          dsl     1
    0028                       32           divisor:        dsl     1
    002C                       33           xptr:           dsw     1
    002E                       34           yptr:           dsw     1
    0030                       35           xval:           dsw     1
    0032                       36           log_val:        dsw     1
                               37
    0000                       38   DSEG
                               39           EXTRN   PLOT_IN
                               40
                               41   $eject
```

270189-64

```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                              02/18/86         PAGE   2

ERR LOC  OBJECT               LINE       SOURCE STATEMENT
                                42
    2500                        43  CSEG at 2500H          ;;;;            PROGRAM  MODULE BEGINS
                                44
                                45   PUBLIC INIT_OUTPUT, CON_OUT, DRAW_GRAPH
                                46   EXTRN  PLOT_RES, PLOT_RES_2, PLOT_MAX
                                47
    2500                        48  INIT_OUTPUT:                           ; INITIALIZE SERIAL PORT
                                49
    2500 B12000          E      50          ldb     iocl,#00100000B ; set p2.0 to txd
                                51
      0270                      52  baud_val        equ     624             ; 624=300 baud (at 12 MHz)
                                53
      0082                      54  Baud_high       equ     ((baud_val-1)/256) OR 80H       ; set for XTAL1 clock
      006F                      55  baud_low        equ     (baud_val-1) MOD 256
                                56
    2503 B16F00          E      57          ldb     baud_reg,#baud_low
    2506 B18200          E      58          ldb     baud_reg,#baud_high
                                59
    2509 B14900          E      60          ldb     spcon,#01001001b        ; enable reciver mode 1
    250C B12000          R      61          ldb     sptmp,#00100000B        ; set TI-tmp
                                62
    250F F0                     63          RET
                                64
                                65  $eject
```

270189-65

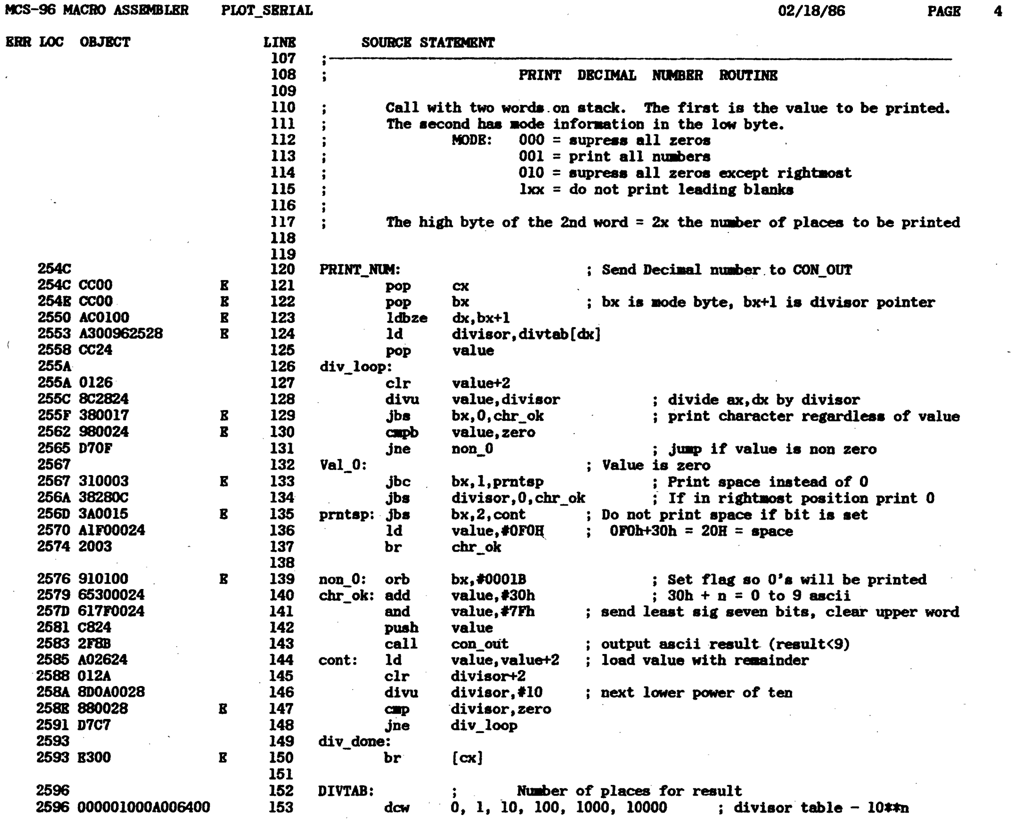
Listing 5—The Plot Module (Continued)



```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                   02/18/86         PAGE    3

ERR LOC  OBJECT                 LINE       SOURCE STATEMENT
                                  66
                                  67  ;--------------------------------------------------------------------
                                  68  ;                        CONSOLE OUT ROUTINE
                                  69  ;
                                  70  ;       Call with a word parameter on stack.  The low byte has the character
                                  71  ;       to be sent.  If the high byte has a value between 81H and 8FEH, the
                                  72  ;       character is repeated 1 to 126 times respectively.  One repeat means
                                  73  ;       that the character will be printed 2 times.  If the high byte contains
                                  74  ;       a value between 1 and 7FH, the charater represented by that value will
                                  75  ;       be printed after the character in the low byte.  If the high byte
                                  76  ;       contains a value of zero only the low byte will be printed.
                                  77
    2510                          78  CON_OUT:
    2510 CC00               E     79          pop     ax                      ; cx contains the calling adress
    2512 CC00               E     80          pop     dx
    2514 3F011C             E     81          jbs     dx+1,7,onechr           ; If bit 7 is set print one character
    2517 980001             E     82          cmpb    dx+1,zero
    251A DF17                     83          je      onechr                  ; if highbyte=0 print one character
                                  84
    251C 900000             E     85  twochr: orb     sptmp,spstat            ; wait for TI
    251F 3500FA             R     86          jbc     sptmp,5,twochr
    2522 71DF00             R     87          andb    sptmp,#11011111b        ; clear TI-tmp
    2525 900000             E     88          orb     zero,spstat             ; remove possible false TI
                                  89
    2528 B00000             E     90          ldb     sbuf,dx
    252B B00100             E     91          ldb     dx,dx+1         ; Load second character
    252E 1101               E     92          clrb    dx+1            ; clear count byte
    2530 717F00             E     93          andb    dx,#07FH        ; mask MSB
                                  94
    2533 1701               E     95  onechr: incb    dx+1
    2535 717F01             E     96          andb    dx+1,#7FH
    2538 900000             E     97  waitl:  orb     sptmp,spstat            ; wait for TI
    253B 3500FA             R     98          jbc     sptmp,5,waitl
    253E 71DF00             R     99          andb    sptmp,#11011111b        ; clear TI-tmp
    2541 900000             E    100          orb     zero,spstat             ; remove possible false TI
                                 101
    2544 B00000             E    102          ldb     sbuf,dx
    2547 E001EE             E    103          DJNZ    dx+1,waitl
    254A E300               E    104          BR      [ax]                    ; Effectively a RET
                                 105
                                 106  $eject
```

270189-66

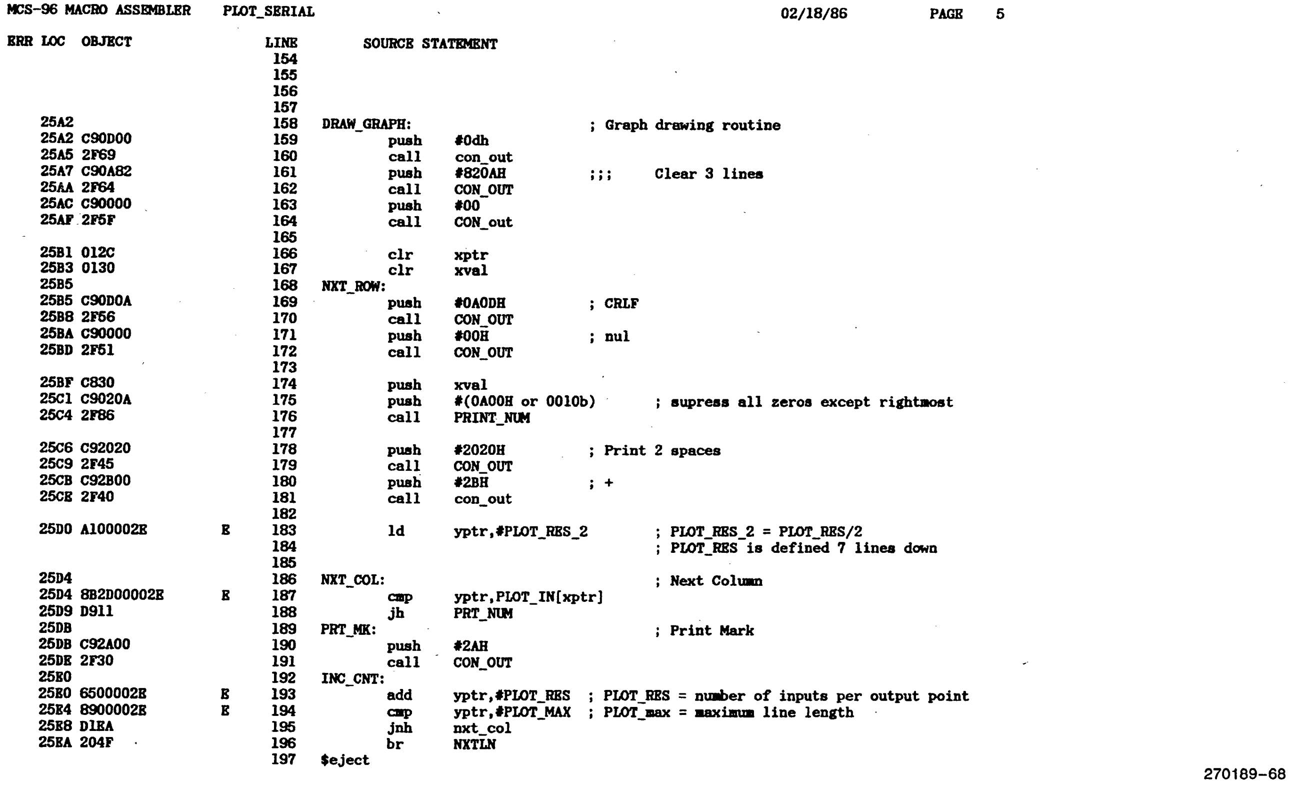
Listing 5—The Plot Module (Continued)



```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                   02/18/86          PAGE    4

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                               107   ;------------------------------------------------------------------------
                               108   ;                         PRINT  DECIMAL  NUMBER  ROUTINE
                               109
                               110   ;       Call with two words on stack.  The first is the value to be printed.
                               111   ;       The second has mode information in the low byte.
                               112   ;               MODE:   000 = supress all zeros
                               113   ;                       001 = print all numbers
                               114   ;                       010 = supress all zeros except rightmost
                               115   ;                       1xx = do not print leading blanks
                               116   ;
                               117   ;       The high byte of the 2nd word = 2x the number of places to be printed
                               118
                               119
    254C                       120   PRINT_NUM:                           ; Send Decimal number to CON_OUT
    254C CC00           E      121           pop     cx
    254E CC00           E      122           pop     bx                   ; bx is mode byte, bx+1 is divisor pointer
    2550 AC0100         E      123           ldbze   dx,bx+1
    2553 A300962528     E      124           ld      divisor,divtab[dx]
    2558 CC24                  125           pop     value
    255A                       126   div_loop:
    255A 0126                  127           clr     value+2
    255C 8C2824                128           divu    value,divisor                ; divide ax,dx by divisor
    255F 380017         E      129           jbs     bx,0,chr_ok                  ; print character regardless of value
    2562 980024         E      130           cmpb    value,zero
    2565 D70F                  131           jne     non_0                        ; jump if value is non zero
    2567                       132   Val_0:                               ; Value is zero
    2567 310003         E      133           jbc     bx,1,prntsp                  ; Print space instead of 0
    256A 38280C                134           jbs     divisor,0,chr_ok             ; If in rightmost position print 0
    256D 3A0015         E      135   prntsp: jbs     bx,2,cont            ; Do not print space if bit is set
    2570 A1F00024              136           ld      value,#0F0H          ;  0F0h+30h = 20H = space
    2574 2003                  137           br      chr_ok
                               138
    2576 910100         E      139   non_0:  orb     bx,#0001B                    ; Set flag so 0's will be printed
    2579 65300024              140   chr_ok: add     value,#30h                   ; 30h + n = 0 to 9 ascii
    257D 617F0024              141           and     value,#7Fh           ; send least sig seven bits, clear upper word
    2581 C824                  142           push    value
    2583 2F8B                  143           call    con_out              ; output ascii result (result<9)
    2585 A02624                144   cont:   ld      value,value+2        ; load value with remainder
    2588 012A                  145           clr     divisor+2
    258A 8D0A0028              146           divu    divisor,#10          ; next lower power of ten
    258E 880028         E      147           cmp     divisor,zero
    2591 D7C7                  148           jne     div_loop
    2593                       149   div_done:
    2593 E300           E      150           br      [cx]
                               151
    2596                       152   DIVTAB:         ;       Number of places for result
    2596 000001000A006400      153           dcw     0, 1, 10, 100, 1000, 10000      ; divisor table - 10**n
```

270189-67

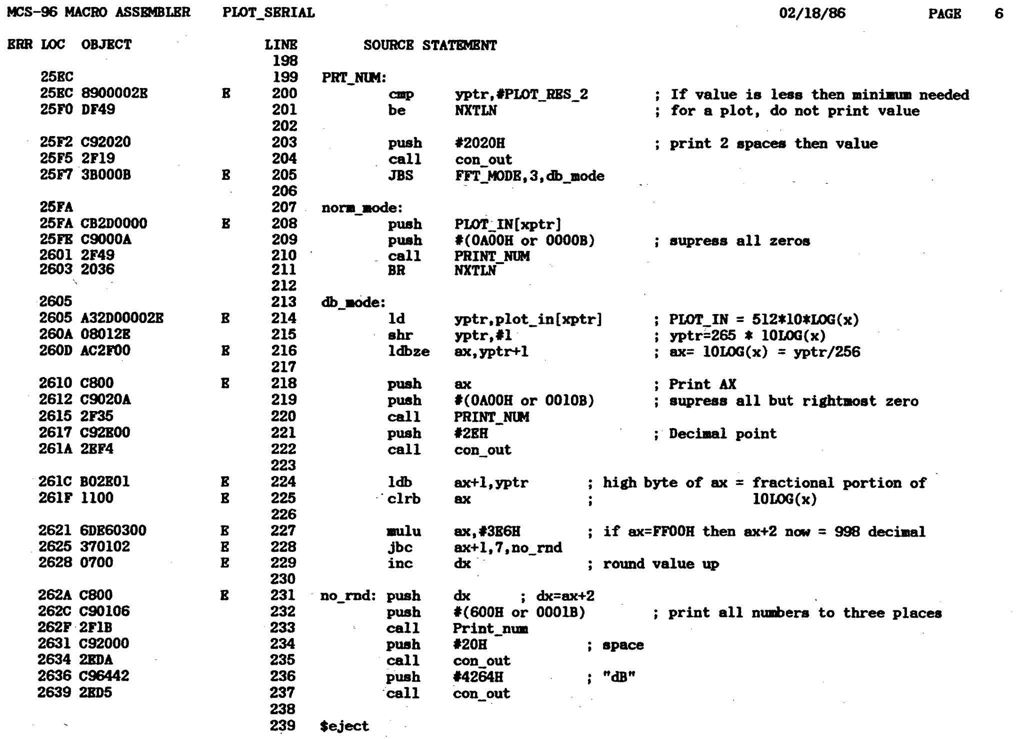
Listing 5—The Plot Module (Continued)



```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                              02/18/86         PAGE    5

ERR LOC  OBJECT                 LINE       SOURCE STATEMENT
                                 154
                                 155
                                 156
                                 157
    25A2                         158   DRAW_GRAPH:                          ; Graph drawing routine
    25A2 C90D00                  159           push     #0dh
    25A5 2F69                    160           call     con_out
    25A7 C90A82                  161           push     #820AH              ;;;     Clear 3 lines
    25AA 2F64                    162           call     CON_OUT
    25AC C90000                  163           push     #00
    25AF 2F5F                    164           call     CON_out
                                 165
    25B1 012C                    166           clr      xptr
    25B3 0130                    167           clr      xval
    25B5                         168   NXT_ROW:
    25B5 C90D0A                  169           push     #0A0DH              ; CRLF
    25B8 2F56                    170           call     CON_OUT
    25BA C90000                  171           push     #00H                ; nul
    25BD 2F51                    172           call     CON_OUT
                                 173
    25BF C830                    174           push     xval
    25C1 C9020A                  175           push     #(0A00H or 0010b)        ; supress all zeros except rightmost
    25C4 2F86                    176           call     PRINT_NUM
                                 177
    25C6 C92020                  178           push     #2020H              ; Print 2 spaces
    25C9 2F45                    179           call     CON_OUT
    25CB C92B00                  180           push     #2BH                ; +
    25CE 2F40                    181           call     con_out
                                 182
    25D0 A100002E        E       183           ld       yptr,#PLOT_RES_2         ; PLOT_RES_2 = PLOT_RES/2
                                 184                                            ; PLOT_RES is defined 7 lines down
                                 185
    25D4                         186   NXT_COL:                                 ; Next Column
    25D4 8B2D00002E      E       187           cmp      yptr,PLOT_IN[xptr]
    25D9 D911                    188           jh       PRT_NUM
    25DB                         189   PRT_MK:                                  ; Print Mark
    25DB C92A00                  190           push     #2AH
    25DE 2F30                    191           call     CON_OUT
    25E0                         192   INC_CNT:
    25E0 6500002E        E       193           add      yptr,#PLOT_RES   ; PLOT_RES = number of inputs per output point
    25E4 8900002E        E       194           cmp      yptr,#PLOT_MAX   ; PLOT_max = maximum line length
    25E8 D1EA                    195           jnh      nxt_col
    25EA 204F                    196           br       NXTLN
                                 197   $eject
```

270189-68

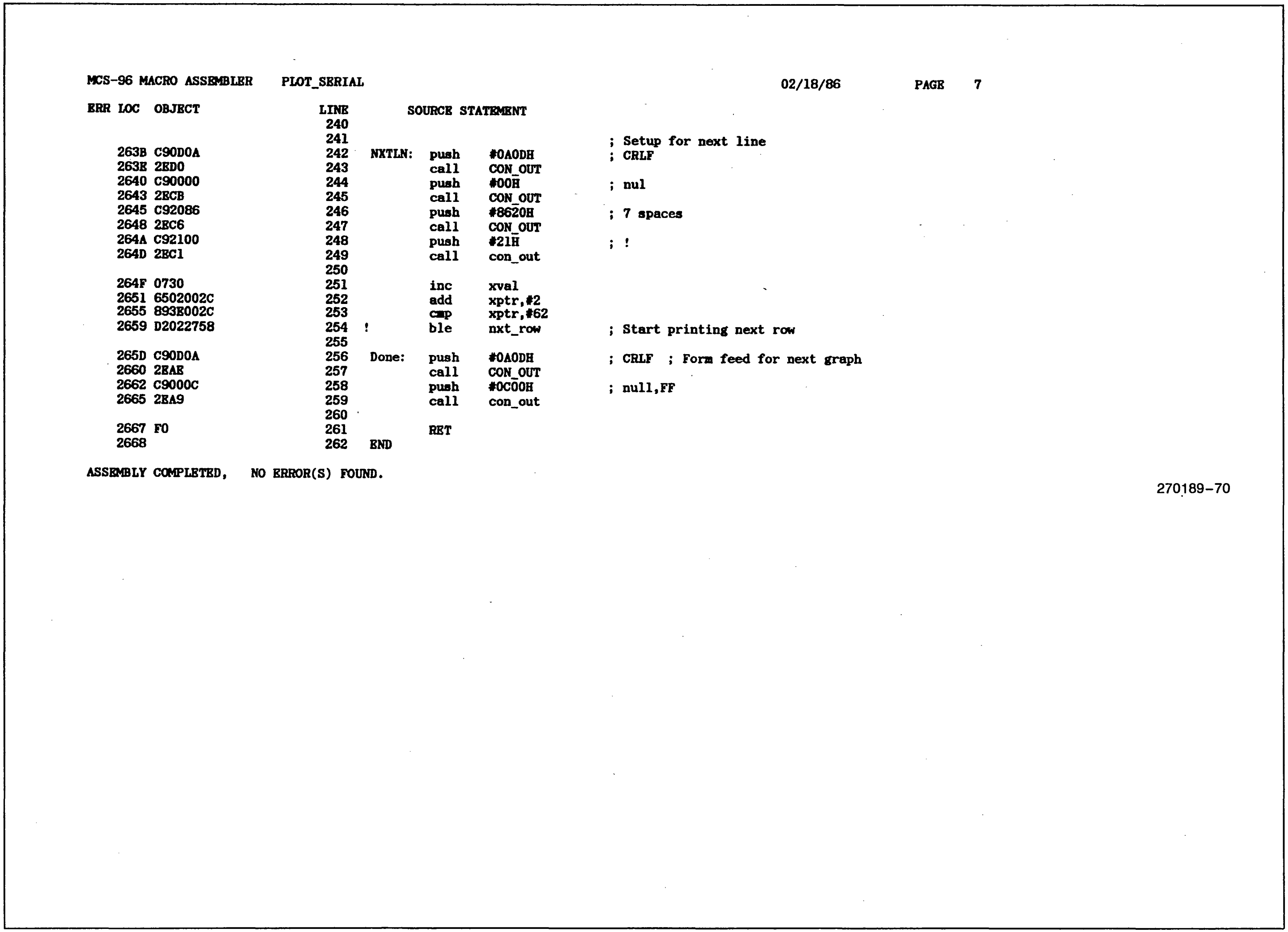
Listing 5—The Plot Module (Continued)



```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                      02/18/86        PAGE    6

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                198
    25EC                        199   PRT_NUM:
    25EC 8900002E          E    200           cmp     yptr,#PLOT_RES_2        ; If value is less then minimum needed
    25F0 DF49                   201           be      NXTLN                   ; for a plot, do not print value
                                202
    25F2 C92020                 203           push    #2020H                  ; print 2 spaces then value
    25F5 2F19                   204           call    con_out
    25F7 3B000B            E    205           JBS     FFT_MODE,3,db_mode
                                206
    25FA                        207   norm_mode:
    25FA CB2D0000          E    208           push    PLOT_IN[xptr]
    25FE C9000A                 209           push    #(0A00H or 0000B)       ; supress all zeros
    2601 2F49                   210           call    PRINT_NUM
    2603 2036                   211           BR      NXTLN
                                212
    2605                        213   db_mode:
    2605 A32D00002E        E    214           ld      yptr,plot_in[xptr]      ; PLOT_IN = 512*10*LOG(x)
    260A 08012E                 215           shr     yptr,#1                 ; yptr=265 * 10LOG(x)
    260D AC2F00            E    216           ldbze   ax,yptr+1               ; ax= 10LOG(x) = yptr/256
                                217
    2610 C800              E    218           push    ax                      ; Print AX
    2612 C9020A                 219           push    #(0A00H or 0010B)       ; supress all but rightmost zero
    2615 2F35                   220           call    PRINT_NUM
    2617 C92E00                 221           push    #2EH                    ; Decimal point
    261A 2EF4                   222           call    con_out
                                223
    261C B02E01            E    224           ldb     ax+1,yptr       ; high byte of ax = fractional portion of
    261F 1100              E    225           clrb    ax              ;                        10LOG(x)
                                226
    2621 6DE60300          E    227           mulu    ax,#3E6H        ; if ax=FF00H then ax+2 now = 998 decimal
    2625 370102            E    228           jbc     ax+1,7,no_rnd
    2628 0700              E    229           inc     dx              ; round value up
                                230
    262A C800              E    231   no_rnd: push    dx      ; dx=ax+2
    262C C90106                 232           push    #(600H or 0001B)        ; print all numbers to three places
    262F 2F1B                   233           call    Print_num
    2631 C92000                 234           push    #20H            ; space
    2634 2EDA                   235           call    con_out
    2636 C96442                 236           push    #4264H          ; "dB"
    2639 2ED5                   237           call    con_out
                                238
                                239   $eject
```

270189-69

Listing 5—The Plot Module (Continued)



```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                         02/18/86          PAGE    7

ERR LOC  OBJECT                 LINE        SOURCE STATEMENT
                                240
                                241                                      ; Setup for next line
    263B C90D0A                 242  NXTLN:  push     #0A0DH             ; CRLF
    263E 2ED0                   243          call     CON_OUT
    2640 C90000                 244          push     #00H               ; nul
    2643 2ECB                   245          call     CON_OUT
    2645 C92086                 246          push     #8620H             ; 7 spaces
    2648 2EC6                   247          call     CON_OUT
    264A C92100                 248          push     #21H               ; !
    264D 2EC1                   249          call     con_out
                                250
    264F 0730                   251          inc      xval
    2651 6502002C               252          add      xptr,#2
    2655 893E002C               253          cmp      xptr,#62
    2659 D2022758               254  !       ble      nxt_row            ; Start printing next row
                                255
    265D C90D0A                 256  Done:   push     #0A0DH             ; CRLF  ; Form feed for next graph
    2660 2EAE                   257          call     CON_OUT
    2662 C9000C                 258          push     #0C00H             ; null,FF
    2665 2EA9                   259          call     con_out
                                260
    2667 F0                     261          RET
    2668                        262  END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270189-70

Listing 5—The Plot Module (Continued)

At the end of the plot, a form feed is given to set the printer up for the next graph. Our printer would frequently miss the character after a CRLF. To solve this problem, a null (ASCII 0) is sent after every CRLF to make sure the printer is ready for the next line. This has been found to be a problem with many devices running at close to their maximum capacity, and the nulls work well to solve it.

With the plot completed, the program begins to run again by taking another set of A to D samples.

11.0 USING THE FFT PROGRAM

The program can be used with either real or tabled data. If real data is used, the signal is applied to analog channel 1. The program as written performs A/D samples at 100 microsecond intervals, collecting the 64 samples in 6.4 milliseconds. This sets the sampling window frequency at 156 Hz. If tabled data is used, 64 words of data should be placed in the location pointed to by DATA0 in the TABLE__LOAD routine of the Main Module.

Program control is specified by FFT__MODE which is loaded in the main module. Also within the main module are settings which control the A to D buffer routine and the Plot routine. The intention was to have only one module to change and recompile to vary parameters in the entire program.

The program modules are set up to run one-at-a-time so that the code would be easy to understand. Additionally, the Plot routine takes so long relative to the other sections, that it doesn't pay to try to overlap code sections. If this code were to be converted to run a process instead of print a graph, it might be worthwhile to run the FFT and the A/D routines at the same time.

If the goal of a modified program is to have the highest frequency sampling possible, it might be desirable to streamline the A/D section and run it without interruption. When the A to D routine was complete the FFT routine could be started. The reasoning behind this is that at the fastest A/D speeds the processor will be almost completely tied up processing the A/D information and storing it away. Using an interrupt based A/D routine would slow things down.

A set of programs which will perform a FFT has been presented in this application note. These programs are available from the INSITE users library as program CA-26. More importantly, dozens of programing examples have been made available, making it easier to get started with the 8096. Examples of how to use the hardware on the 8096 have already appeared in AP-248, "Using The 8096". These two applications notes form a good base for the understanding of MCS-96 microcontroller based design.

12.0 APPENDIX A - MATRICES

Matrices are a convenient way to express groups of equations. Consider the complex discrete Fourier Transform in equation 9, with N = 4.

$$Y_n = \sum_{k=0}^{3} X(k)\, W^{nk} \qquad n = 0, 1, 2, 3$$

This can be expanded to

$$\begin{aligned}
Y(0) &= X(0)\,W^0 + X(1)\,W^0 + X(2)\,W^0 + X(3)\,W^0\\
Y(1) &= X(0)\,W^0 + X(1)\,W^1 + X(2)\,W^2 + X(3)\,W^3\\
Y(2) &= X(0)\,W^0 + X(1)\,W^2 + X(2)\,W^4 + X(3)\,W^6\\
Y(3) &= X(0)\,W^0 + X(1)\,W^3 + X(2)\,W^6 + X(3)\,W^9
\end{aligned}$$

In matrix notation, this is shown as

$$\begin{bmatrix} Y(0)\\ Y(1)\\ Y(2)\\ Y(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0\\ W^0 & W^1 & W^2 & W^3\\ W^0 & W^2 & W^4 & W^6\\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} X(0)\\ X(1)\\ X(2)\\ X(3) \end{bmatrix}$$

The first step to simplifying this is to reduce the center matrix. Recalling that

$$W^N = W^{N\ \mathrm{MOD}\ N} \quad \text{and} \quad W^0 = 1$$

The matrix can be reduced to have less non-trivial multiplications.

$$\begin{bmatrix} Y(0)\\ Y(1)\\ Y(2)\\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1\\ 1 & W^1 & W^2 & W^3\\ 1 & W^2 & W^0 & W^2\\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} X(0)\\ X(1)\\ X(2)\\ X(3) \end{bmatrix}$$

The square matrix can be factored into

$$\begin{bmatrix} Y(0)\\ Y(2)\\ Y(1)\\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0\\ 1 & W^2 & 0 & 0\\ 0 & 0 & 1 & W^1\\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0\\ 0 & 1 & 0 & W^0\\ 1 & 0 & W^2 & 0\\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} X(0)\\ X(1)\\ X(2)\\ X(3) \end{bmatrix}$$

For this equation to work, the Y(1) and Y(2) terms need to be swapped, as shown above. This procedure is a Bit Reversal, as described in the text.

Multiplying the two rightmost matrices results in

$$\begin{array}{ll}
X(0) + X(2)\,W^0 & \\
X(1) + X(3)\,W^0 & \text{requiring 4 complex multiplications}\\
X(0) + X(2)\,W^3 & \qquad\text{\& 4 complex additions}\\
X(1) + X(3)\,W^2 &
\end{array}$$

Noting that $W^0 = -W^2$, 2 of the complex multiplications can be eliminated, with the following results

$$\begin{array}{ll}
X(0) + X(2)\,W^0 & \\
X(1) + X(3)\,W^0 & \text{requiring 2 complex multiplications}\\
X(0) - X(2)\,W^0 & \qquad\text{and 4 complex additions}\\
X(1) - X(3)\,W^0 &
\end{array}$$

Since $W^1 = -W^3$, a similar result occurs when this vector is multiplied by the remaining square matrix. The resulting equations are:

$$\begin{aligned}
Y(0) &= (X(0) + X(2)\,W^0) + W^0\,(X(0) + X(3)\,W^0)\\
Y(2) &= (X(0) + X(2)\,W^0) - W^0\,(X(1) + X(3)\,W^0)\\
Y(1) &= (X(0) - X(2)\,W^0) + W^1\,(X(1) - X(3)\,W^0)\\
Y(3) &= (X(0) - X(2)\,W^0) - W^1\,(X(1) - X(3)\,W^0)
\end{aligned}$$

The number of complex multiplications required is 4, as compared with 16 for the unfactored matrix.

In general, the FFT requires

$$\frac{N * \text{EXPONENT}}{2} \text{ complex multiplications}$$

and

$$N * \text{EXPONENT complex additions}$$

where

$$\text{EXPONENT} = \text{Log}_2\, N$$

A standard Fourier Transform requires

$$N^2 \text{ complex multiplications}$$

and

$$N(N-1) \text{ complex additions}$$

13.0 APPENDIX B - PLOTS

The following plots are examples of output from the FFT program. These plots were generated using tabled data, but very similar plots have also been made using the analog input module. Typically, a plot made using the analog input module will not show quite as much power at each frequency and will show a positive value for the DC component. This is because it is difficult to get exactly a full-scale analog input with no DC offset.

Plot 1 is a Magnitude plot of a square wave of period NT.

Plot 2 is the same data plotted in dB. Note how the dB plot enhances the difference in the small signal values at the high frequencies.

Plot 3 shows the windowed version of this data. Note that the widening of the bins due to windowing shows energy in the even harmonics that is not actually present. For data of this type a different window other than Hanning would normally be used. Many window types are available, the selection of which can be determined by the type of data to be plotted.[3]

Plot 4 shows a sine wave of period NT/7 or frequency 7/NT.

Plot 5 shows the same input with windowing. Note the signal shown in bins 6 and 8.

Plot 6 shows a sine wave of period NT/7.5. Note the noise caused by the discontinuity as discussed earlier.

Plot 7 uses windowing on the data used for plot 6. Note the cleaner appearance.

Plot 8 shows a sine wave input of magnitude 0.707 and period NT/7.5.

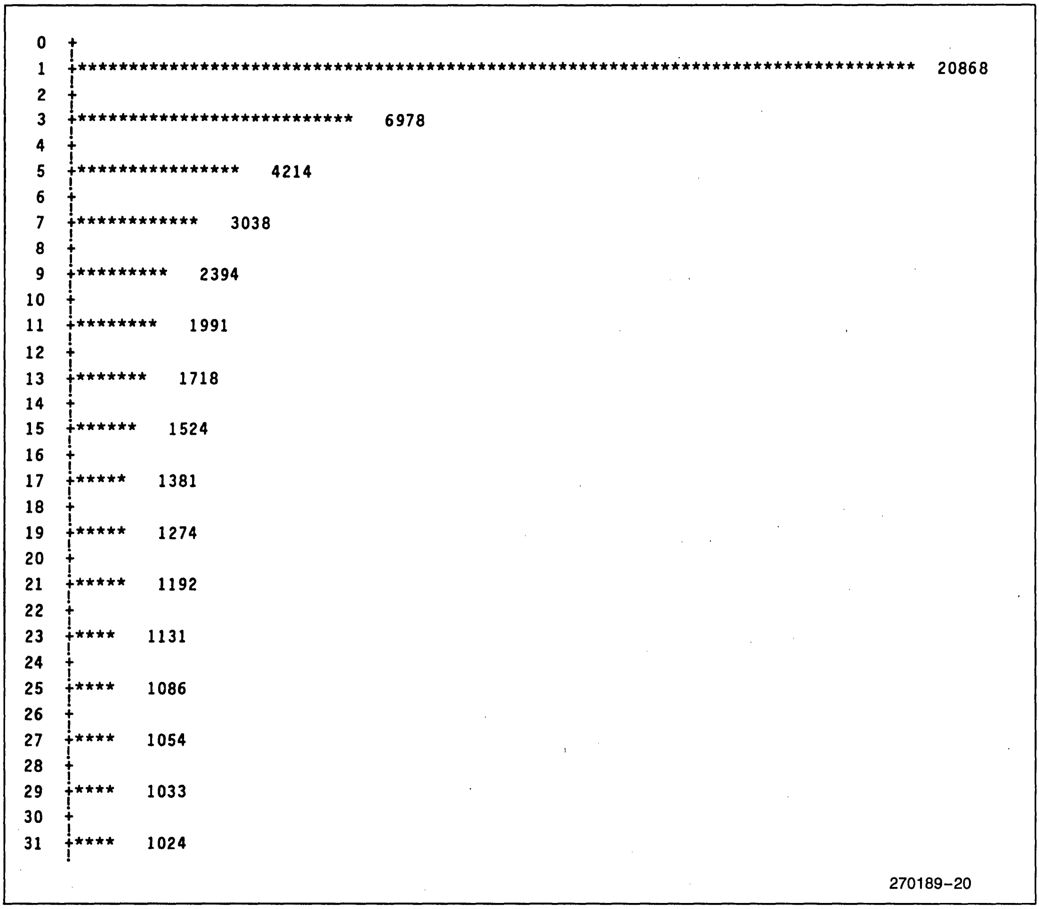
Plot 9 shows same input with windowing.

Plot 10 shows a sine wave of magnitude 0.707/16 and period NT/11.

Plot 11 shows the same input with windowing. Note that there is no power shown in bins 10 and 12. This is because at 6 dB down from 3 dB they are nearly equal to zero.

Plot 12 uses the sum of the signals for plots 8 and 10 as inputs. Note that the component at period NT/11 is almost hidden.

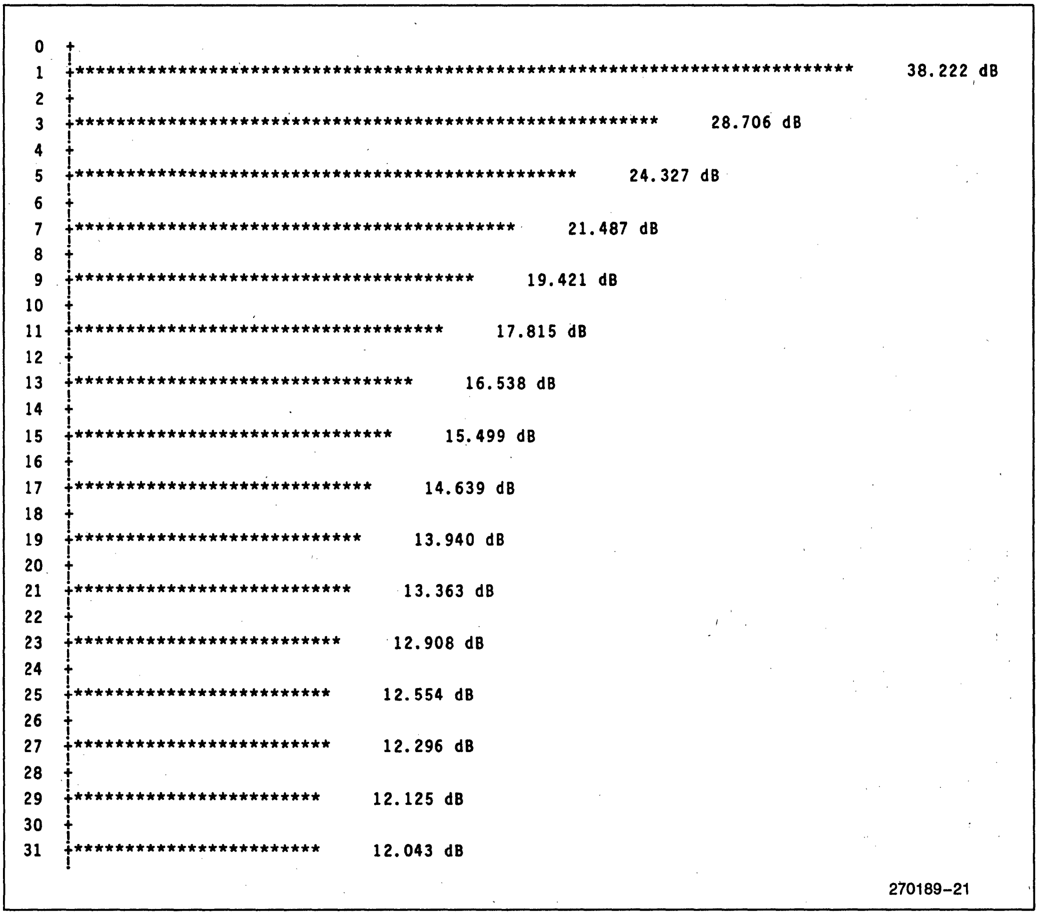
Plot 13 uses the same signal as plot 12 but applies windowing. Now the period component at NT/11 can easily be seen. The Hanning window works well in this case to separate the signal from the leakage. If the signals were closer together the Hanning window may not have worked and another window may have been needed.



```
 0  +
    !
 1  +****************************************************************************************   20868
    !
 2  +
    !
 3  +***************************   6978
    !
 4  +
    !
 5  +****************   4214
    !
 6  +
    !
 7  +************   3038
    !
 8  +
    !
 9  +*********   2394
    !
10  +
    !
11  +********   1991
    !
12  +
    !
13  +*******   1718
    !
14  +
    !
15  +******   1524
    !
16  +
    !
17  +*****   1381
    !
18  +
    !
19  +*****   1274
    !
20  +
    !
21  +*****   1192
    !
22  +
    !
23  +****   1131
    !
24  +
    !
25  +****   1086
    !
26  +
    !
27  +****   1054
    !
28  +
    !
29  +****   1033
    !
30  +
    !
31  +****   1024
    !
```

270189–20

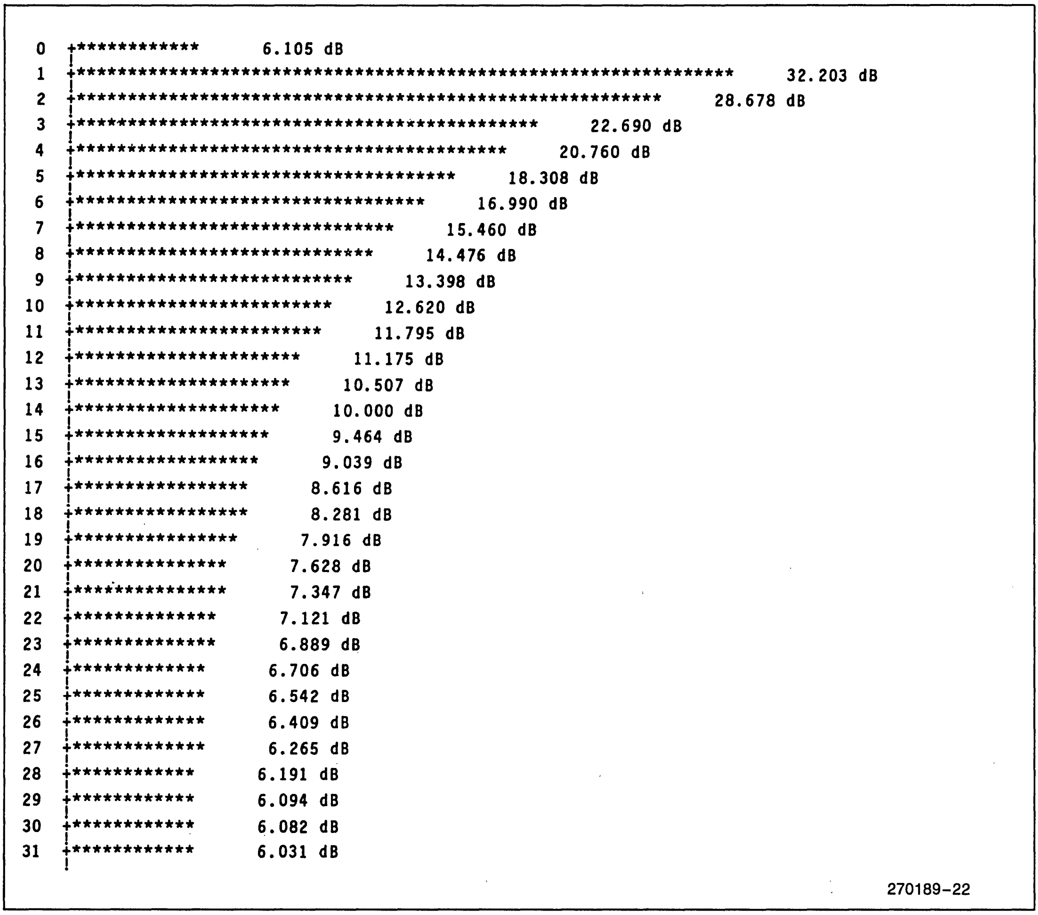
Plot 1—Magnitude Plot of Squarewave



```
 0 +
 1 +****************************************************************************     38.222 dB
 2 +
 3 +*********************************************************     28.706 dB
 4 +
 5 +*************************************************     24.327 dB
 6 +
 7 +*******************************************     21.487 dB
 8 +
 9 +***************************************     19.421 dB
10 +
11 +************************************     17.815 dB
12 +
13 +*********************************     16.538 dB
14 +
15 +*******************************     15.499 dB
16 +
17 +*****************************     14.639 dB
18 +
19 +****************************     13.940 dB
20 +
21 +***************************     13.363 dB
22 +
23 +**************************     12.908 dB
24 +
25 +*************************     12.554 dB
26 +
27 +*************************     12.296 dB
28 +
29 +************************     12.125 dB
30 +
31 +************************     12.043 dB
   !
```

270189-21

Plot 2—Decibel Plot of Squarewave



```
 0  +*************      6.105 dB
 1  +*****************************************************************     32.203 dB
 2  +**********************************************************     28.678 dB
 3  +**********************************************     22.690 dB
 4  +*******************************************     20.760 dB
 5  +**************************************     18.308 dB
 6  +***********************************     16.990 dB
 7  +********************************     15.460 dB
 8  +******************************     14.476 dB
 9  +****************************     13.398 dB
10  +**************************     12.620 dB
11  +*************************     11.795 dB
12  +***********************     11.175 dB
13  +**********************     10.507 dB
14  +*********************     10.000 dB
15  +********************     9.464 dB
16  +*******************     9.039 dB
17  +******************     8.616 dB
18  +******************     8.281 dB
19  +*****************     7.916 dB
20  +****************     7.628 dB
21  +****************     7.347 dB
22  +***************     7.121 dB
23  +***************     6.889 dB
24  +**************     6.706 dB
25  +**************     6.542 dB
26  +**************     6.409 dB
27  +**************     6.265 dB
28  +*************     6.191 dB
29  +*************     6.094 dB
30  +*************     6.082 dB
31  +*************     6.031 dB
```

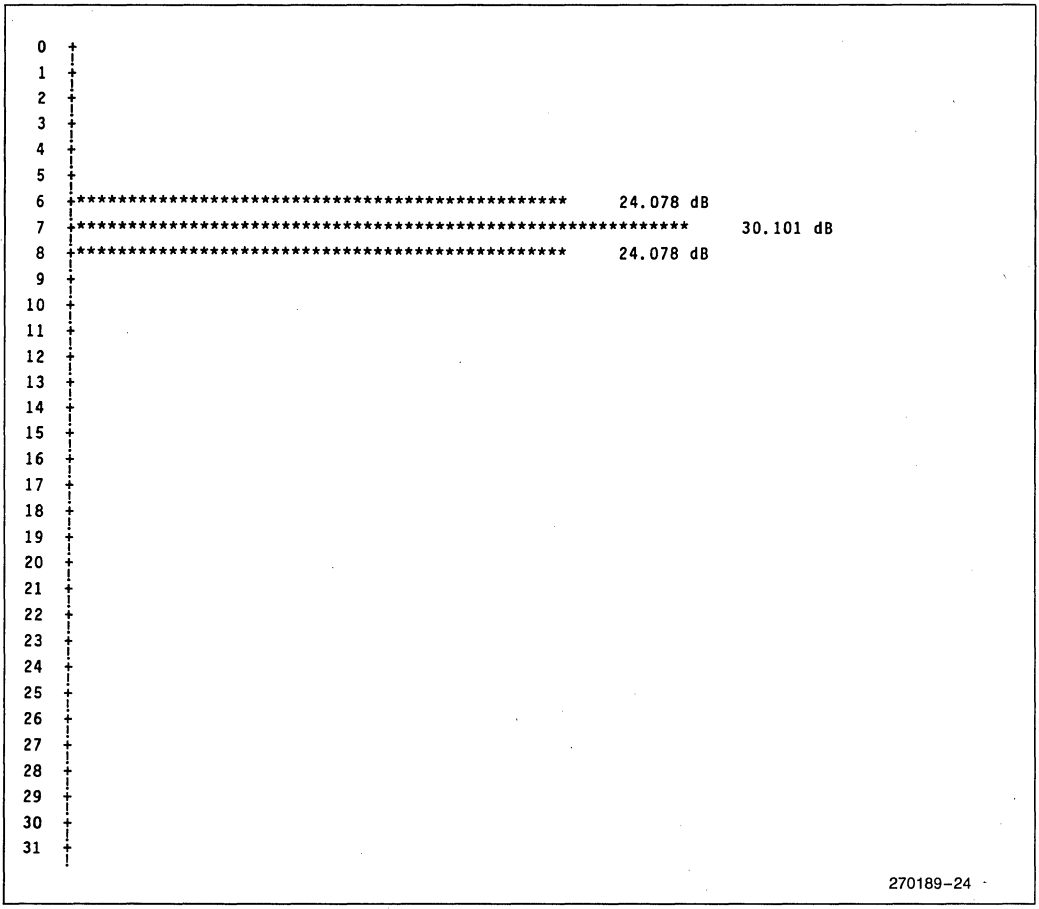
270189-22

Plot 3—Plot of Squarewave with Window

```
 0  +
 1  +
 2  +
 3  +
 4  +
 5  +
 6  +
 7  +*********************************************************************************      36.121 dB
 8  +
 9  +
10  +
11  +
12  +
13  +
14  +
15  +
16  +
17  +
18  +
19  +
20  +
21  +
22  +
23  +
24  +
25  +
26  +
27  +
28  +
29  +
30  +
31  +
    |
```

270189-23

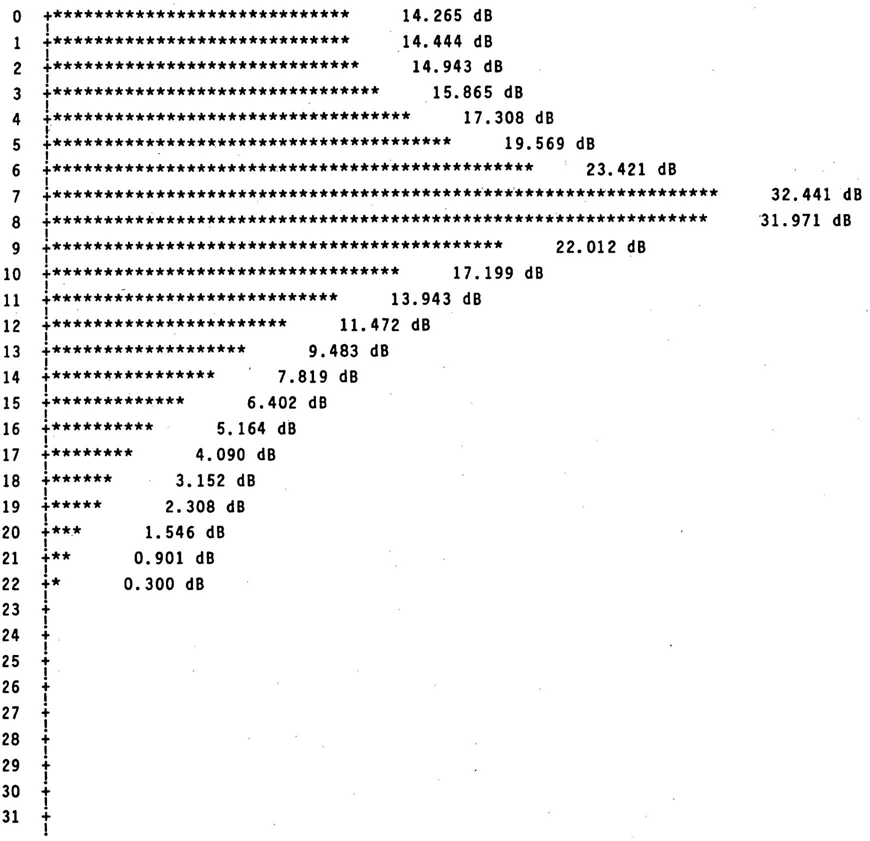
Plot 4—Sin (7.0X) without Window



```
 0  +
    !
 1  +
    !
 2  +
    !
 3  +
    !
 4  +
    !
 5  +
    !
 6  +**************************************************     24.078 dB
    !
 7  +***************************************************************     30.101 dB
    !
 8  +**************************************************     24.078 dB
    !
 9  +
    !
10  +
    !
11  +
    !
12  +
    !
13  +
    !
14  +
    !
15  +
    !
16  +
    !
17  +
    !
18  +
    !
19  +
    !
20  +
    !
21  +
    !
22  +
    !
23  +
    !
24  +
    !
25  +
    !
26  +
    !
27  +
    !
28  +
    !
29  +
    !
30  +
    !
31  +
    !
```

270189-24

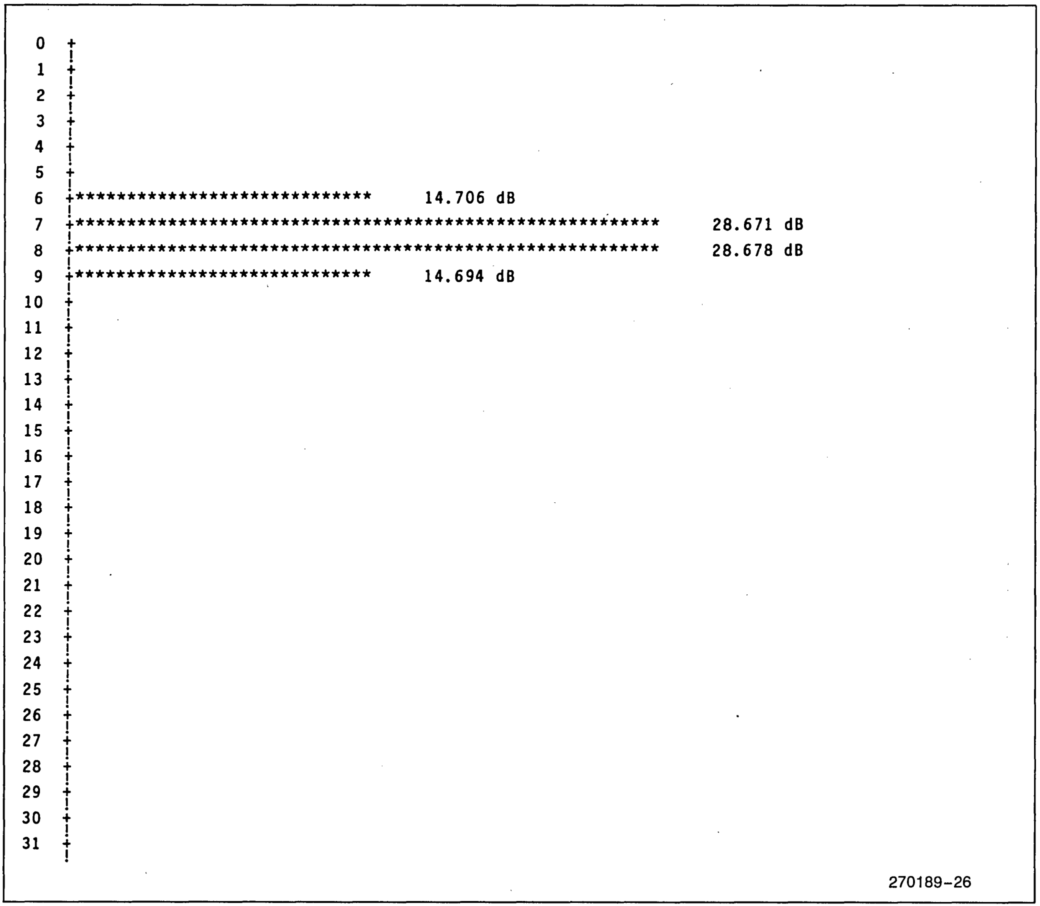
Plot 5—Sin (7.0X) with Window



```
 0  +*****************************     14.265 dB
    |
 1  +*****************************     14.444 dB
    |
 2  +******************************     14.943 dB
    |
 3  +********************************     15.865 dB
    |
 4  +***********************************     17.308 dB
    |
 5  +***************************************     19.569 dB
    |
 6  +***********************************************     23.421 dB
    |
 7  +*****************************************************************     32.441 dB
    |
 8  +****************************************************************     31.971 dB
    |
 9  +********************************************     22.012 dB
    |
10  +**********************************     17.199 dB
    |
11  +****************************     13.943 dB
    |
12  +***********************     11.472 dB
    |
13  +*******************     9.483 dB
    |
14  +****************     7.819 dB
    |
15  +*************     6.402 dB
    |
16  +**********     5.164 dB
    |
17  +********     4.090 dB
    |
18  +******     3.152 dB
    |
19  +*****     2.308 dB
    |
20  +***     1.546 dB
    |
21  +**     0.901 dB
    |
22  +*     0.300 dB
    |
23  +
    |
24  +
    |
25  +
    |
26  +
    |
27  +
    |
28  +
    |
29  +
    |
30  +
    |
31  +
    |
```

270189-25

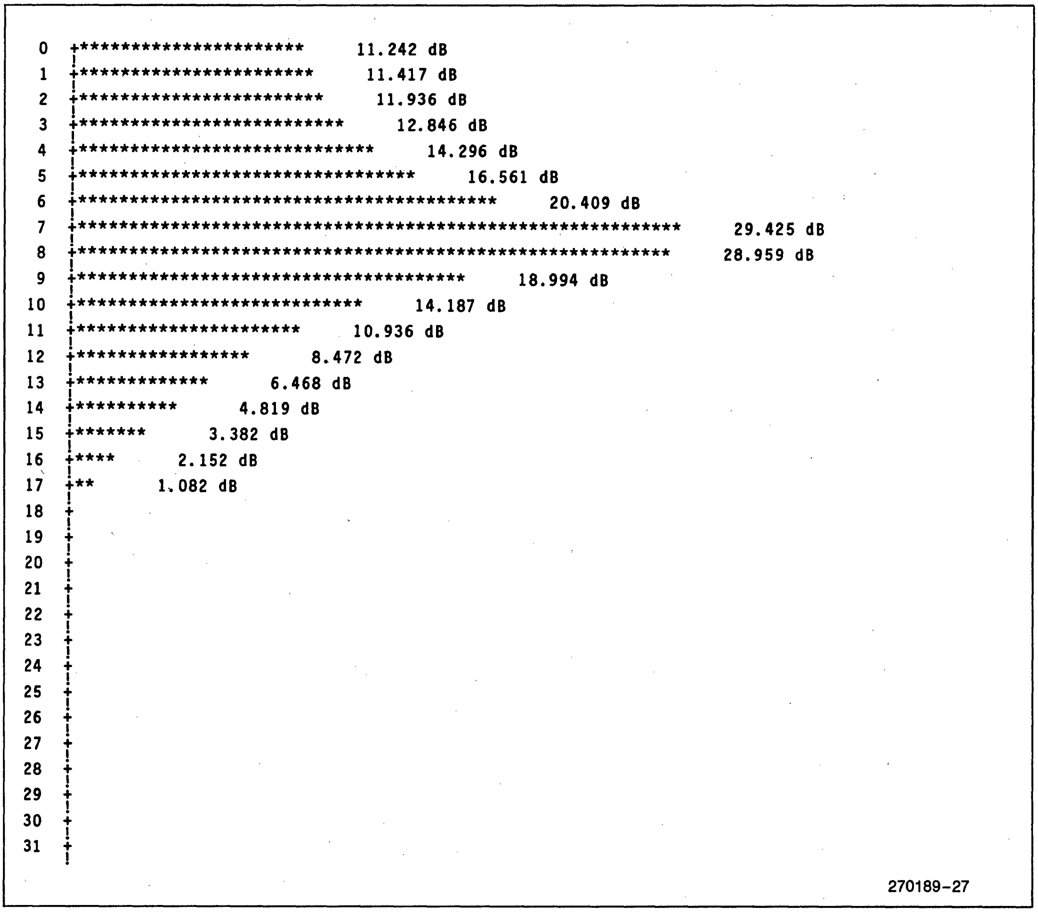
Plot 6—Sin (7.5X) without Window



```
 0  +
 1  +
 2  +
 3  +
 4  +
 5  +
 6  +*****************************      14.706 dB
 7  +*********************************************************      28.671 dB
 8  +*********************************************************      28.678 dB
 9  +*****************************      14.694 dB
10  +
11  +
12  +
13  +
14  +
15  +
16  +
17  +
18  +
19  +
20  +
21  +
22  +
23  +
24  +
25  +
26  +
27  +
28  +
29  +
30  +
31  +
```

270189-26

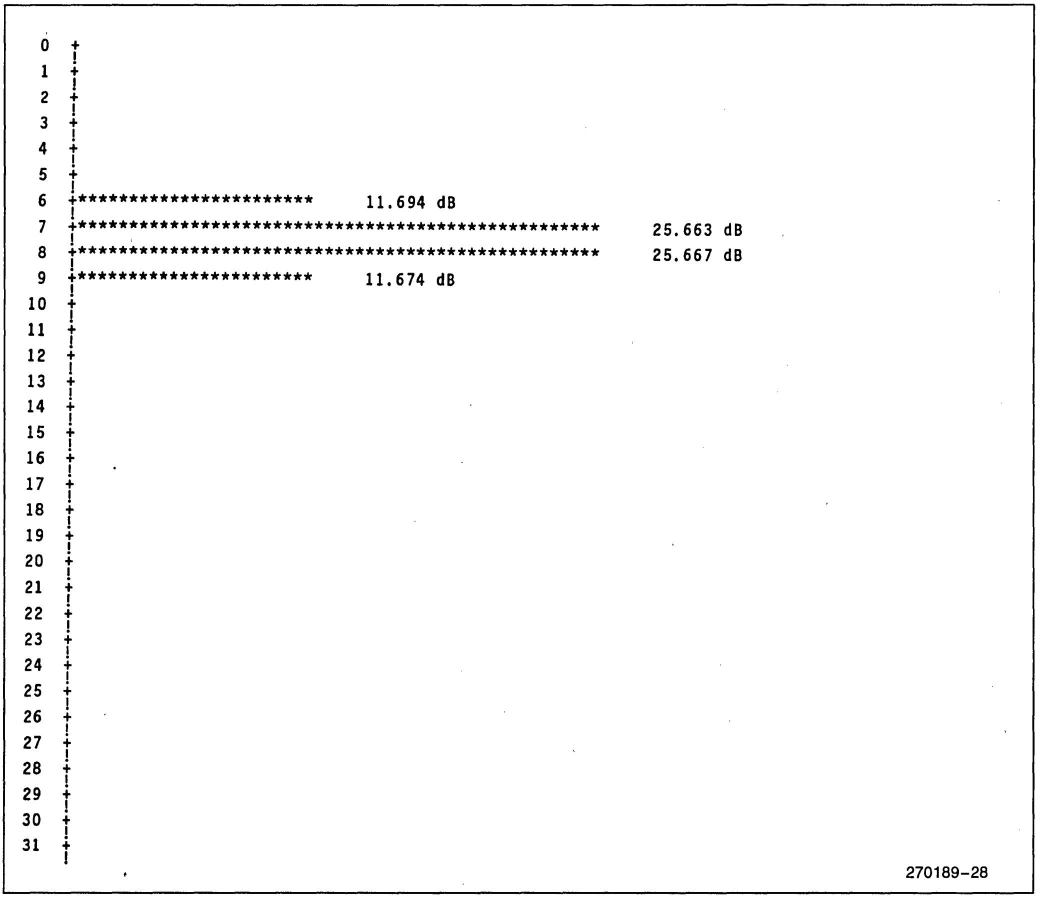
Plot 7—Sin (7.5X) with Window



```
 0  +**********************      11.242 dB
    |
 1  +***********************      11.417 dB
    |
 2  +************************      11.936 dB
    |
 3  +**************************      12.846 dB
    |
 4  +*****************************      14.296 dB
    |
 5  +*********************************      16.561 dB
    |
 6  +*****************************************      20.409 dB
    |
 7  +***********************************************************      29.425 dB
    |
 8  +**********************************************************      28.959 dB
    |
 9  +**************************************      18.994 dB
    |
10  +****************************      14.187 dB
    |
11  +**********************      10.936 dB
    |
12  +*****************      8.472 dB
    |
13  +*************      6.468 dB
    |
14  +**********      4.819 dB
    |
15  +*******      3.382 dB
    |
16  +****      2.152 dB
    |
17  +**      1.082 dB
    |
18  +
    |
19  +
    |
20  +
    |
21  +
    |
22  +
    |
23  +
    |
24  +
    |
25  +
    |
26  +
    |
27  +
    |
28  +
    |
29  +
    |
30  +
    |
31  +
    |
```

270189-27

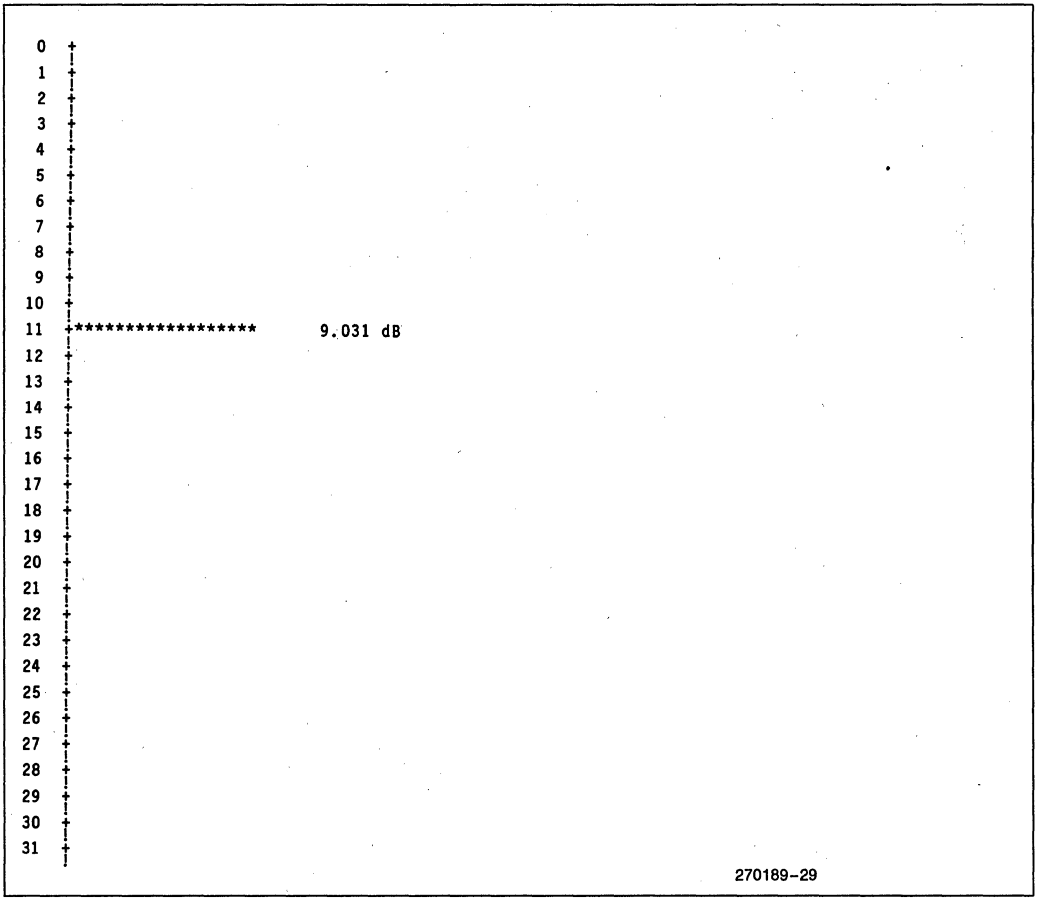
Plot 8—0.707 * Sin (7.5X) without Window



```
 0  +
 1  +
 2  +
 3  +
 4  +
 5  +
 6  +***********************     11.694 dB
 7  +***************************************************     25.663 dB
 8  +***************************************************     25.667 dB
 9  +***********************     11.674 dB
10  +
11  +
12  +
13  +
14  +
15  +
16  +
17  +
18  +
19  +
20  +
21  +
22  +
23  +
24  +
25  +
26  +
27  +
28  +
29  +
30  +
31  +
```

270189–28

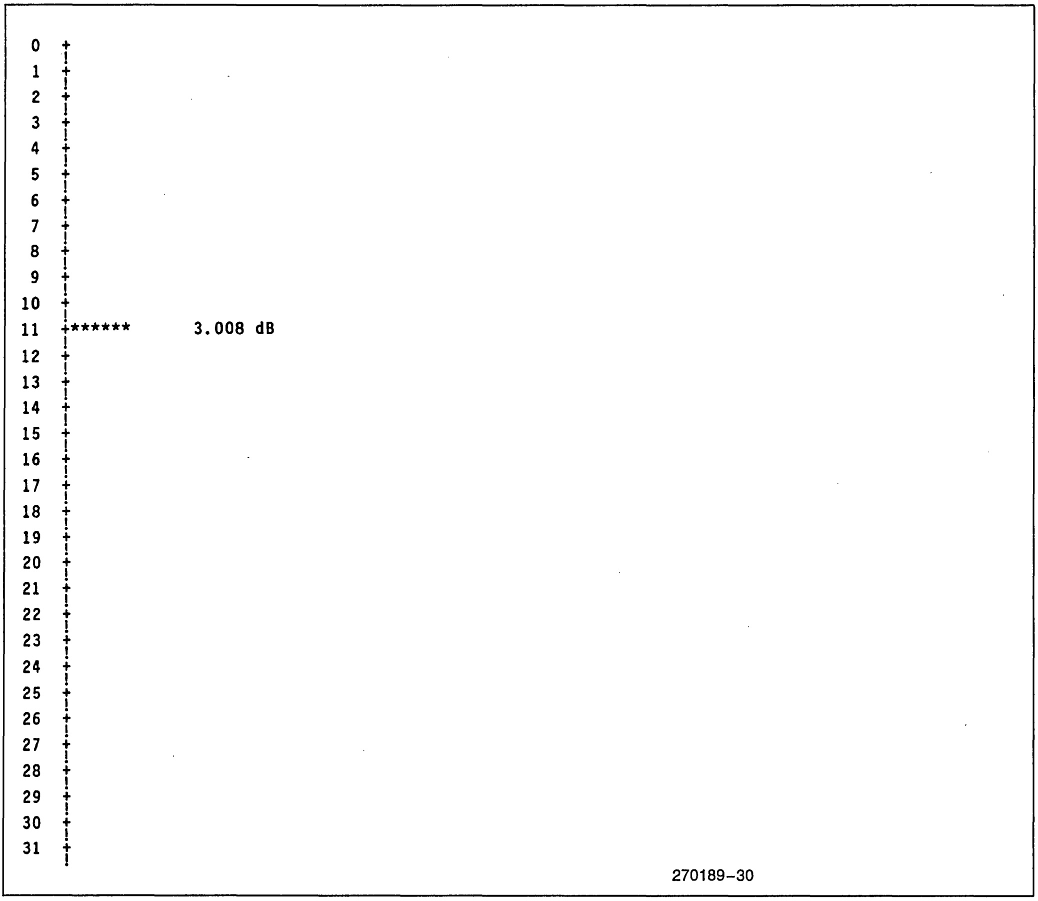
Plot 9—0.707 * Sin (7.5X) with Window



```
 0  +
    !
 1  +
    !
 2  +
    !
 3  +
    !
 4  +
    !
 5  +
    !
 6  +
    !
 7  +
    !
 8  +
    !
 9  +
    !
10  +
    !
11  +******************      9.031 dB
    !
12  +
    !
13  +
    !
14  +
    !
15  +
    !
16  +
    !
17  +
    !
18  +
    !
19  +
    !
20  +
    !
21  +
    !
22  +
    !
23  +
    !
24  +
    !
25  +
    !
26  +
    !
27  +
    !
28  +
    !
29  +
    !
30  +
    !
31  +
    !
```

270189-29

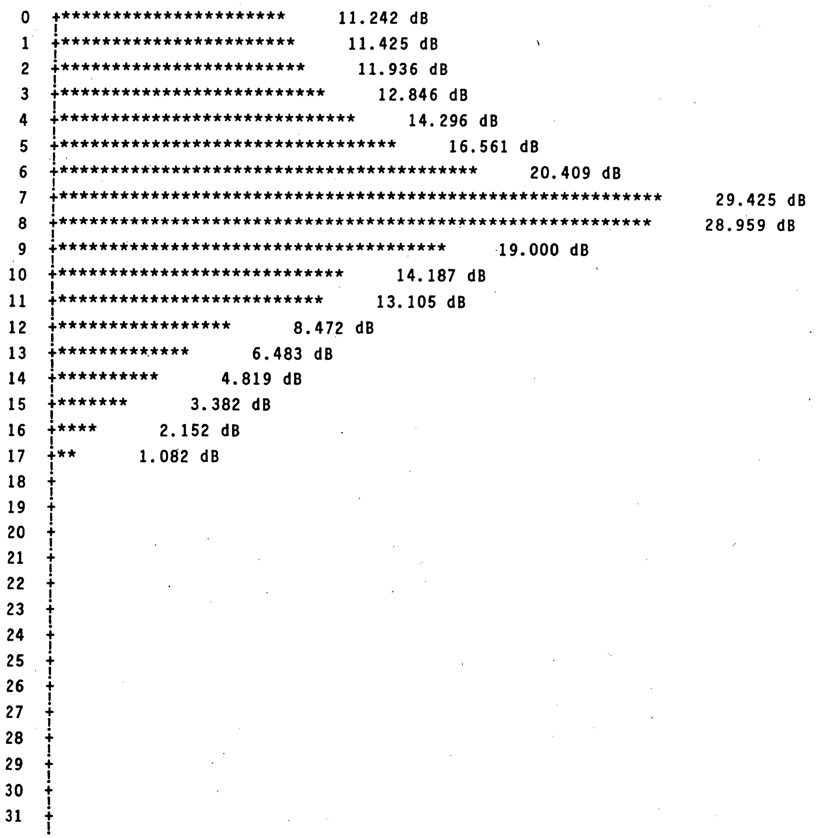
Plot 10—0.707/16 * Sin (11X) without Window



```
 0  +
    :
 1  +
    :
 2  +
    :
 3  +
    :
 4  +
    :
 5  +
    :
 6  +
    :
 7  +
    :
 8  +
    :
 9  +
    :
10  +
    :
11  +******       3.008 dB
    :
12  +
    :
13  +
    :
14  +
    :
15  +
    :
16  +
    :
17  +
    :
18  +
    :
19  +
    :
20  +
    :
21  +
    :
22  +
    :
23  +
    :
24  +
    :
25  +
    :
26  +
    :
27  +
    :
28  +
    :
29  +
    :
30  +
    :
31  +
    :
```

270189–30

Plot 11—0.707/16 * Sin (11X) with Window



```
 0  +**********************     11.242 dB
 1  +***********************     11.425 dB
 2  +************************     11.936 dB
 3  +**************************     12.846 dB
 4  +*****************************     14.296 dB
 5  +*********************************     16.561 dB
 6  +*****************************************     20.409 dB
 7  +***********************************************************     29.425 dB
 8  +**********************************************************     28.959 dB
 9  +**************************************     19.000 dB
10  +****************************     14.187 dB
11  +**************************     13.105 dB
12  +*****************     8.472 dB
13  +*************     6.483 dB
14  +**********     4.819 dB
15  +*******     3.382 dB
16  +****     2.152 dB
17  +**     1.082 dB
18  +
19  +
20  +
21  +
22  +
23  +
24  +
25  +
26  +
27  +
28  +
29  +
30  +
31  +
```

270189-31

Plot 12—0.707 (Sin (7.5X) + 1/16 Sin (11X)) without Window

```
 0  +
 1  +
 2  +
 3  +
 4  +
 5  +
 6  +************************      11.702 dB
 7  +*****************************************************      25.663 dB
 8  +*****************************************************      25.667 dB
 9  +************************      11.674 dB
10  +
11  +******       3.074 dB
12  +
13  +
14  +
15  +
16  +
17  +
18  +
19  +
20  +
21  +
22  +
23  +
24  +
25  +
26  +
27  +
28  +
29  +
30  +
31  +
    |
```

270189–32

Plot 13—0.707 (Sin (7.5X) + 1⁄16 Sin (11X)) with Window

BIBLIOGRAPHY

1. Boyet, Howard and Katz, Ron, The 16-Bit 8096: Programming, Interfacing, Applications. 1985, Microprocessor Training Inc., New York, NY.
2. Brigham, E. Oran, The Fast Fourier Transform. 1974, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
3. Harris, Fredric J., On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform. Proceedings of the IEEE, Vol. 66, No. 1, January 1978.
4. Weaver, H. Joseph, Applications of discrete and continuous Fourier analysis. 1983, John Wiley and Sons, New York.

INTEL PUBLICATIONS

1. 1986 Microcontroller Handbook, Order Number 210918-004
2. Using the 8096, AP-248, Order Number 270061-001
3. MCS-96 Macro Assembler User's Guide, Order Number 122048-001
4. MCS-96 Utilities User's Guide, Order Number 122049-001

8096 SOFTWARE DEVELOPMENT PACKAGES

- **Choice of Hosts**
- **MCS®-96 Software Support Package**
- **C-96/196 Software Package**
- **Supports All Members of the MCS-96 Family**
- **PL/M-96 Software Package**

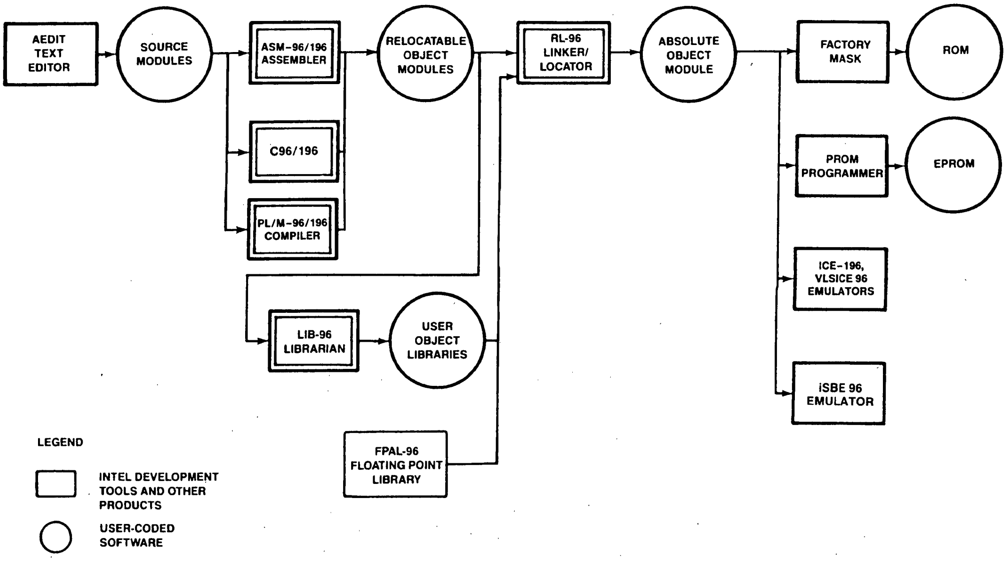
8096 ASSEMBLER PACKAGE

- **Symbolic relocatable assembly language programming for the 8096 microcontroller family**
- **System Utilities for Program Linking and Relocation**
- **Extends Intellec® Microcomputer Development System to support 8096 program development**
- **Encourages modular program design for maintainability and reliability**

The 8096 Software Support Package provides development system support for the 8096 family of 16-bit single chip microcomputers. The support package includes a macro assembler and system utilities.

The assembler produces relocatable object modules from 8096 macro assembly language instructions. The object modules then are linked and located to absolute memory locations.

The assembler and utilities run on PC DOS 3.0 IBM* PC XT/AT Systems.



230613-1

Figure 1. 8096 Software Development Process

*IBM is a registered trademark of International Business Machines Corporation.

October 1987
Order Number: 230613-006

8096 MACRO ASSEMBLER

- Gives Symbolic Access to Powerful 8096 Hardware Features
- Object Files are Linkable and Locatable
- Symbolic Assembler Supports Macro Capabilities, Cross Reference, Symbol Table and Conditional Assembly

ASM-96 is the macro assembler for the MCS family of microcontrollers, including the 80C196. ASM-96 translates symbolic assembly language mnemonics into relocatable object code. Since the object modules are linkable and locatable, ASM-96 encourages modular programming practices.

The macro facility in ASM-96 allows programmers to save development and maintenance time since common code sequences only have to be done once. The assembler also provides conditional assembly capabilities.

ASM-96 supports symbolic access to the many features of the 8096 architecture. An "include" file is provided with all of the 8096 hardware registers defined. Alternatively, the user can define any subset of the 8096 hardware register set.

Math routines are supported with mnemonics for 16 × 16-bit multiply or 32/16-bit divide instructions.

The assembler runs on a PC-DOS 3.0 IBM PC XT/AT.

RL96 LINKER AND RELOCATOR PROGRAM

- Links Modules Generated by ASM-96, C-96, and PL/M-96
- Locates the Linked Object Module to Absolute Memory Locations
- Encourages Modular Programming for Faster Program Development
- Automated Selection of Required Modules from Libraries to Satisfy Symbolic References

RL96 is a utility that performs two functions useful in MCS-96 software development:

— The link function which combines a number of MCS-96 object modules into a single program.

— The locate functions which assigns an absolute address to all relocatable addresses in the MCS-96 object module.

RL96 resolves all external symbol references between modules and will select object modules from library files if necessary.

RL96 creates two files:

— The program or absolute object module file that can be executed by the targeted member of the MCS-96 family.

— The listing file that shows the results of link/locate, including a memory map symbol table and an optional cross reference listing.

The relocator allows programmers to concentrate on software functionally and not worry about the absolute addresses of the object code. RL96 promotes modular programming. The application can be broken down into separate modules that are easier to design, test and maintain. Standard modules can be developed and used in different applications thus saving software development time.

FPAL96 FLOATING POINT ARITHMETIC LIBRARY

- **Implements IEEE Floating Point Arithmetic**
- **Basic Arithmetic Operations +, −, ×, /, Mod Plus Square Root**
- **Supports Single Precision 32 Bit Floating Point Variables**
- **Includes an Error Handler Library**

FPAL96 is a library of single precision 32-bit floating point arithmetic functions. All math adheres to the proposed IEEE floating point standard for accuracy and reliability. An error handler to handle exceptions (for example, divide by zero) is included.

The following functions are included:

ADD	NEGATE
SUBTRACT	ABSOLUTE
MULTIPLY	SQUARE ROOT
DIVIDE	INTEGER
COMPARE	REMAINDER

LIB 96

The LIB 96 utility creates and maintains libraries of software object modules. The customer can develop standard modules and place them in libraries. Application programs can then call these modules using predefined interfaces.

LIB 96 uses the following set of commands:

—CREATE:	Creates an empty library file.
—ADD:	Adds object modules to a library file.
—DELETE:	Deletes object modules from a library file.
—LIST:	Lists the modules in the library file.
—EXIT:	Terminates LIB 96

When using object libraries, RL96 will include only those object modules that are required to satisfy external references, thus saving memory space.

ORDERING INFORMATION

Order Code	Operating Environment
D86ASM96	96 Assembler for PC DOS 3.0 Systems

Documentation Package:

MCS-96 Macro Assembler User's Guide
MCS-96 Utilities User's Guide
MCS-96 Assembler and Utilities Pocket Reference Card
8096 Floating Point Arithmetic Library

SUPPORT:

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

PL/M-96 SOFTWARE PACKAGE

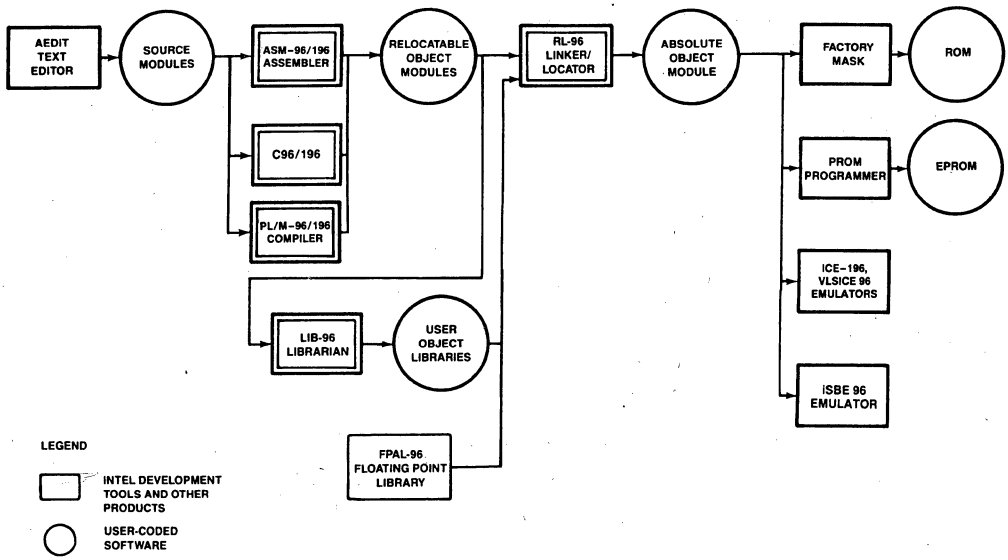
- **Choice of Hosts**
- **Block Structured Language Design Encourages Module Programming**
- **Provides Access to 8096 on Chip Resources**
- **Produces Relocatable Object Code which is Linkable to Object Modules Generated by Other 8096 Translators**
- **Resident on 8086 Intel Microcomputer Development Systems for Higher Performance**
- **Includes a Linking and Relocating Utility and the Library Manager**
- **IEEE Floating Point Library included for Numeric Support**
- **Compatible with PL/M-86 Assuring Design Portability**

PL/M-96 is a structured, high-level programming language useful for developing software for the Intel 8096 family of microcontrollers, including the 80C196. PL/M-96 was designed to support the software requirements of advanced 16 bit microcontrollers. Access to the on chip resources of the 8096 has been provided in PL/M-96.

PL/M-96 is compatible with PL/M-86. Programmers familiar with PL/M will find they can program in PL/M-96 with little relearning effort.

The PL/M-96 compiler translates PL/M-96 high level language statements into 8096 machine instructions. By programming in PL/M an engineer can be more productive in the initial software development cycle of the project. PL/M can also reduce future maintenance and support cost because PL/M programs are easier to understand. PL/M-96 was designed to complement Intel's ASM-96.

PL/M-96 is available for PC DOS 3.0 based IBM PC XT/AT Systems.



230613-1

Figure 2. PL/M-96 Software Package

PL/M-96 COMPILER

FEATURES

Major features of the PL/M-96 compiler and programming language include:

Structured Programming

Programs written in PL/M-96 are developed as a collection of procedures, modules and blocks. Structured programs are easier to understand, maintain and debug. PL/M-96 programs can be made more reliable by clearly defining the scope of user variables (for example, local variables in a procedure). REENTRANT procedures are also supported by PL/M-96.

Language Compatibility

PL/M-96 object modules are compatible with all other object modules generated by Intel MCS-96 translators. Programmers may choose to link ASM-96 and PL/M-96 object modules together.

PL/M-96 object modules were designed to work with other Intel support tools for the MCS-96. The DEBUG compiler control provides these tools with symbolic information.

Data Types Supported

PL/M-96 supports seven data types for programmer flexibility in various logical, arithmetic and addressing functions. The seven data types include:

—BYTE:	8-bit unsigned number
—WORD:	16-bit unsigned number
—DWORD:	32-bit unsigned number
—SHORTINT:	8-bit signed number
—INTEGER:	16-bit signed number
—LONGINT:	32-bit signed number
—REAL:	32-bit floating point number

Another powerful feature are BASED variables. BASED variables allow the user to map more than one variable to the same memory location. This is especially useful for passing parameters, relative and absolute addressing, and memory allocation.

Data Structures Supported

Two data structuring facilities are supported by PL/M-96. The user can organize data into logical groups. This adds flexibility in referencing data.

- — Array: Indexed list of same type data elements
- — Structure: Named collection of same or different type data elements
- — Combinations of Both: Arrays of structures or structures of arrays

Interrupt Handling

Interrupts are supported in PL/M-96 by defining a procedure with the INTERRUPT attribute. The compiler will generate code to save and restore the program status word when handling hardware interrupts of the MCS-96.

Compiler Controls

Compile time options increase the flexibility of the PL/M-96 compiler. These controls include:

- — Optimization
- — Conditional compilation
- — The inclusion of common PL/M-96 source files from disk
- — Cross reference of symbols
- — Optional assembly language code in the listing file

Code Optimizations

The PL/M-96 compilers has four levels of optimization for reducing program size.

— Combination of constant expressions; "Strength reductions" (e.g.: a shift left rather than multiply by two)

— Machine code optimizations; elimination of superfluous branches; reuse of duplicate code, removal of unreachable code

— Overlaying of on chip RAM variables

— Optimization of based variable operations

— Use of short jumps where possible

Built in Functions

An extensive list of built in functions has been supplied as part of the PL/M-96 language. Besides TYPE CONVERSION functions, there are built in functions for STRING manipulations. Functions are provided for interrogating the MCS-96 hardware flags such as CARRY and OVERFLOW.

Error Checking

If the PL/M-96 compiler detects a programming or compilation error, a fully detailed error message is provided by the compiler. If a syntax or program error is detected, the compiler will skip the code generation and optimization passes. This powerful PL/M-96 feature can yield a two times increase in throughput when a user is in the initial program development cycle.

BENEFITS

PLM-96 is designed to be an efficient, cost-effective solution to the special requirements of MCS-96 Microcontroller Software Development, as illustrated by the following benefits of PL/M use:

Low Learning Effort

PL/M-96 is easy to learn and to use, even for the novice programmer.

Earlier Project Completion

Critical projects are completed much earlier than otherwise possible because PL/M-96, a structured high-level language, increases programmer productivity.

Lower Development Cost

Increases in programmer productivity translate immediately into lower software development costs because less programming resources are required for a given programmed function.

Increased Reliability

PL/M-96 is designed to aid in the development of reliable software (PL/M programs are simple statements of the program algorithm). This substantially reduces the risk of costly correction of errors in systems that have already reached full production status. The more simply the program is stated, the more likely it is to perform its intended function.

Easier Enhancements and Maintainance

Programs written in PL/M tend to be self-documenting, thus easier to read and understand. This means it is easier to enhance and maintain PL/M programs as the system capabilities expand and future products are developed.

ORDERING INFORMATION

Order Code	Operating Environment
D86PLM96	PL/M-96 Compiler for PC DOS 3.0 based Systems

Documentation Package

PL/M-96 User's Guide
MCS-96 Utilities User's Guide
MCS-96 Assembler and Utilities Pocket Reference Card
8096 Floating Point Arithmetic Library

SUPPORT

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

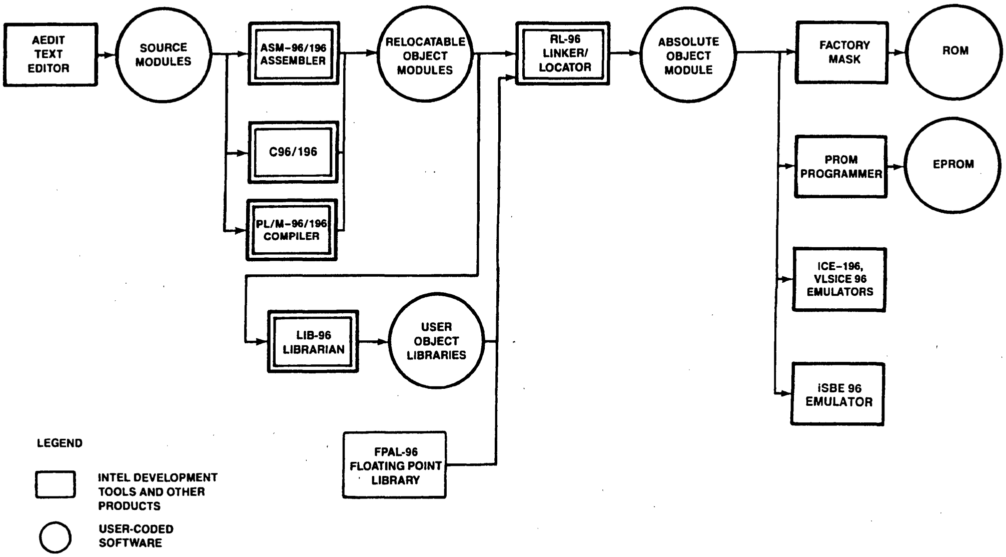
C 96 SOFTWARE PACKAGE

- **Implements the Full Programming Capabilities of the C Language**
- **Complies with Draft ANSI Standard**
- **Produces Relocatable Object Code which is Linkable to Object Modules Generated by Other MCS®-96 Translators**
- **Produces High-Density Code That Rivals Assembly in Efficiency**
- **Fully Linkable with the PL/M-96 and ASM-96 Programming Languages**
- **IEEE Floating Point Library (FPAL96) Included for Numeric Support**
- **Supports All of the Standard C Language I/O Library (STDIO)**
- **Includes a Linking and Relocating Utility, an Object-To-Hexadecimal Convertor, and a Library Manager**
- **Supports the 80C196 Architecture**

Intel's C 96 is a general purpose, structured programming language designed to support applications for the 16-bit family of MCS-96 microcontrollers including the 80C196. C 96 implements the C language as described in the Kernighan and Ritchie book, *The C Programming Language* (Prentice-Hall) Software Series, 1978). The latest enhancements to the C programming language as defined by the draft proposed ANSI C standard (e.g., structure assignments, and the void and enum data types) are supported.

The C 96 compiler translates C 96 language statements into MCS-96 machine instructions. The compiler generates code in Intel's relocatable Object Module Format (OMF) without using an intermediate assembly file. The OMF files can then be debugged using either the iSBE-96 emulator, the VLSiCE-96 emulator, or the ICE-196

C 96 is available for the IBM PC AT and the PC XT



230613-1

Figure 3. 8096 Software Development Process

C 96 COMPILER

COMPILER DESCRIPTION

Major features of the C 96 compiler include the preprocessor, the parser, and the code generator and optimizer. The code is output in Intel relocatable Object Module Format (OMF). The compiled code can then be debugged with either the iSBE-96 emulator or the VLSiCE-96 emulator.

The preprocessor interprets statements in the source code and performs such actions as macro expansion, file inclusion, and conditional compilation (for example, the #if directive, which specifies optional inclusion or exclusion of code).

The parser performs syntactic and semantic error checking on the code. The code generator converts the parser's output into efficient binary code. The optimizer streamlines the code and generates Intel relocatable OMF code, without creating an intermediate assembly file.

The compiler's DEBUG/NODEBUG control option specifies whether or not the object module should contain debug information. The debug information can be used to debug the compiled program using either the iSBE-96 emulator or the VLSiCE-96 emulator.

COMPILER FEATURES

Some of the features of the C 96 compiler are:

- Declarations
- Expressions and operators
- Statements
- Run-time library (STDIO)
- Compiler invocation
- Output conventions

Each of these features is discussed in the following sections.

DECLARATIONS

Declarations are used to specify the attributes of a set of identifiers. The scope of a declaration can encompass the entire source file or be local to a function body or block.

The storage class specifier defines the location and scope. The storage classes are as follows:

• auto	active block
• extern	external data definition
• static	active data segment or register segment
• typedef	a type definition (not storage allocation) that defines another name or a synonym

The storage class can be further defined with one of the following storage class modifiers:

• const	code segment
• register	machine register
• volatile	I/O port (modifies the extern storage class only)

Identifiers are defined by their type. The types fall into one of the following categories:

• basic	characters, integers, floating point numbers
• derived	arrays, structures, unions, enumerations, functions, and pointers
• void	empty set

The type is further defined by the following type specifiers:

- char, short, int, long, signed, unsigned, float, double, struct, union, enum, and typedef

EXPRESSIONS AND OPERATORS

All of the C language expressions and operators are supported by Intel's C 96 compiler. Table 1 is a summary of the C operators, arranged in order of precedence (from top to bottom). Operator precedence within an expression is evaluated in the order of associativity shown in Table 1.

STATEMENTS

A statement is a program element that specifies an action to be performed. The C language supports the following types of statements:

• Simple	any valid expression
• Compound	an optional list of variable declarations followed by a list of statements

- Selection an if or switch statement which is optionally included dependent on specified conditions
- Iteration a do, while or for statement which executes repeatedly until the controlling value is zero
- Branching a break, continue, goto, or return statement which changes the program control flow

Table 1. Precedence and Associativity

Class	Operator	Associativity
primary	[ ] ( ) . →	left to right
unary	++ −− & * + − ~ sizeof far	right to left
binary mult.	* / %	left to right
binary add	+ −	left to right
binary shift	<< >>	left to right
binary relat.	< > <= >=	left to right
binary equal.	== =	left to right
bitwise AND	&	left to right
bitwise XOR	∧	left to right
bitwise OR	|	left to right
logical AND	&&	left to right
logical OR	||	left to right
conditional	? :	right to left
assignment	= *= /= %= += −=	right to left
	<<=+ >>= &= ∧= |=	right to left
comma	,	left to right

RUN-TIME LIBRARY (STDIO)

Intel's C 96 compiler supports the standard C language I/O library functions (STDIO). The include files listed in Table 2 are included with the C 96 compiler.

Table 2. C 96 Include Files

Name	Description
ctype.h	Used to declare and map characters.
errno.h	Used for error checking.
setjump.h	Used to bypass a normal call/return.
stdio.h	Used for standard I/O functions.
string.h	Used to manipulate strings.
time.h	Used to manipulate the time and date.

Character and arithmetic conversion functions are also included (atof, atoi, atol, cstr, tolower, toupper, and udistr).

COMPILER INVOCATION

Intel's C 96 compiler is invoked with the following general syntax:

c96 *pathname* [*controls*]

The following invocation controls are some of the options supported by the C 96 compiler.

- Object file controls—DEBUG/NODEBUG, OBJECT, OPTIMIZE (0 through 3), REGISTERS, REGOVERLAY/NOREGOVERLAY, TYPE/NOTYPE
- Listing controls (selection and content)—CODE/NOCODE, COND/NOCOND, LIST/NOLIST, LISTINCLUDE/NOLISTINCLUDE, PREPRINT/NOPREPRINT, SYMBOLS/NOSYMBOLS, XREF/NOXREF
- Listing format controls—PAGING/NOPAGING, PAGELENGTH, PAGEWIDTH
- Source inclusion control—INCLUDE

The REENTRANT/NOREENTRANT extension has been added to the C 96 compiler invocation controls to enhance the compiler's use of the MCS-96 architecture. This extension enables the compiler to fully use the large register set of the MCS-96 family of microprocessors. When porting to programs in other environments, these keywords should be either removed or defined as null.

Output Conventions

The C 96 compiler produces a listing file and an object file. The listing file contains a formatted list of the source code and a list of compiler error messages. The compiler produces the object file in Intel's relocatable OMF code directly, without creating an intermediate assembly file.

BENEFITS

There are many benefits to the C 96 compiler, as explained in the following sections.

PROGRAM DEBUGGING

With the DEBUG control the C 96 compiler produces extensive debug information, including symbols. The debug information can be used to debug the program code with either the VLSiCE-96 emulator or the iSBE-96 emulator. This serves to enhance programmer productivity.

FASTER COMPILATION

The C 96 compiler creates Intel object module format (OMF) directly, without creating an intermediate assembly file. This increases the compiler's execution speed.

PORTABLE CODE

Code portability has been designed into the C 96 compiler. The C 96 code is fully linkable with both the PL/M-96 and the ASM-96 programming languages.

Because the compiler supports the standard C library and produces Intel OMF code, programs developed on a variety of machines can be transported to the MCS-96. In addition, because C 96 conforms to accepted C language standards, programmers can quickly begin programming the MCS-96.

FULL MANIPULATION OF THE 8096 MICROCONTROLLER

The C 96 compiler has been highly optimized for the MCS-96 architecture. The REENTRANT/NOREENTRANT control has been added so that the compiler can identify non-reentrant procedures. This is extremely useful because it enables the programmer to have full access to the large MCS-96 register set.

With the C 96 compiler, the programmer can declare register variables that are not local to any procedure. Due to the large register set of the MCS-96 architecture, the compiler can dedicate registers to such variables.

SOFTWARE SUPPORT

Intel's Software Support Service provides maintenance on software packages with software support contracts which include subscription services, information phone support, and updates. Consulting services can be arranged for on-site assistance at the customer's location for both short-term and long-term needs. For more information, contact your local Intel Sales Office.

ORDERING INFORMATION

Part Number	Description
D86C96	C 96 Software Package
	PL/M-96 packages also include the RL96 Linker and Relocator, the FPAL96 Floating Point Library, and the LIB96 librarian utility.

Operating Environment

IBM PC AT

IBM PC XT

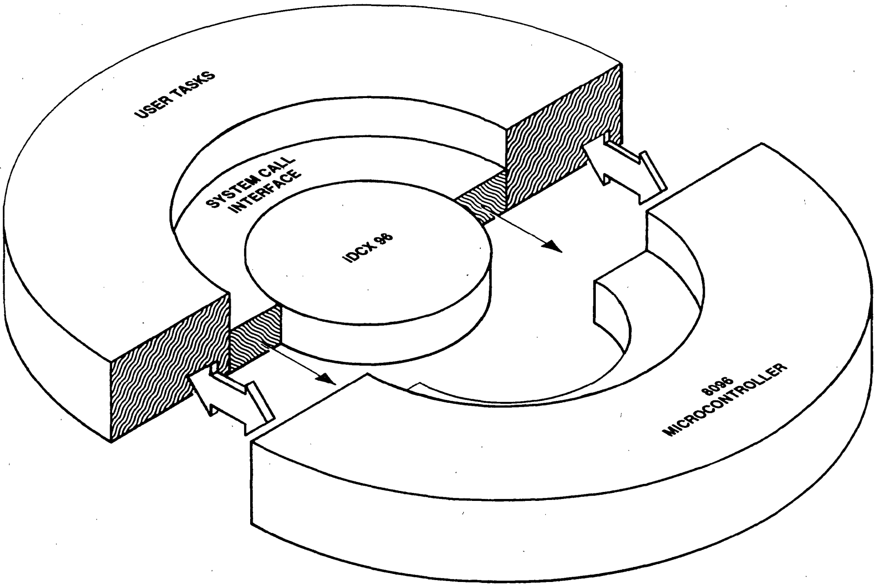
intel®

iDCX 96
DISTRIBUTED CONTROL EXECUTIVE

- **High Performance, Real-time, Multitasking Executive**
- **Full Support of MSC®-96 Microcontroller Family**
- **Configurable for User Customization**
- **Integral Task Management, Timing, Interrupt and Message Passing Services**
- **Reliable, Compact 2.9K bytes**
- **Simple User Interface**

The iDCX 96 Distributed Control Executive is compact, configurable, easy-to-use software for developing and implementing applications built on the high performance 16-bit family of 8096 microcontrollers (MCS-96). As a real-time, multitasking nucleus, the iDCX 96 Executive enhances the users ability to efficiently design MCS-96 microcontroller applications requiring handling of multiple asynchronous events, and real-time response.

In addition to the features integrated into most microcontrollers (CPU, RAM, ROM, and I/O) the MCS-96 family provides analog to digital conversion, pulse width modulation, and high-speed I/O facilities. Some examples of applications well-suited to the feature set and performance of the 8096 microcontrollers are: motor control, medical instrumentation, automotive transmission control, and machine control. Using the iDCX 96 Distributed Control Executive in these environments will significantly reduce application development time and expense. The iDCX 96 Executive performs equally well in stand-alone applications as well as distributed applications.



280148-1

Figure 1. iDCX 96 Distributed Control Executive System

December 1986
Order Number: 280148-002

ARCHITECTURE

Real-time and Multitasking

Real-time control systems must be responsive to the external environment and typically involve the execution of more than one function (task or set of tasks) in response to different external stimuli. Control of manufacturing process is an example. These processes can require the monitoring of multiple temperatures and pressures; control of heaters, fans, and motors all responding to many seemingly random inputs. The iDCX 96 Distributed Control Executive fully supports applications requiring response to inputs as they occur ie., in real-time. Multiple tasks in control applications require real-time response. The iDCX 96 Executive helps the user implement these multitasking time-critical applications.

Some of the executive's facilities specifically tailored for developing and implementing standalone and distributed control systems are: task managment, timing and interrupt handling, and message passing. When integrated with communications software, the iDCX 96 Executive provides message passing to tasks on different microcontrollers. Response to the environment is guaranteed due to the event-driven nature of the executive. Interrupts, timers, or messages can initiate tasks for proper system response.

Task Management

A task can be thought of as a block of code that performs a specific activity. This activity is one that can occur in parallel with other activities in the system. A task starts at a single point and executes indefinitely, usually in a loop. The iDCX 96 Executive's multitasking facility allows the user to partition system applications code into manageable activities or tasks. Each task competes for processor resources. The executive provides all synchronization, control, and scheduling to ensure each task gets the processor time it requires. A priority mechanism used by the executive determines when a task accesses the processor. Up to 16 tasks can be managed by the executive.

All tasks in an iDCX 96 Executive application are in one of three states as shown in Figure 2. For example, when an RQ WAIT system call is made, the calling task becomes ASLEEP until one of the events upon which it is waiting occurs. These events can be messages, timeouts, time intervals, or interrupts. When an event occurs the task becomes READY or RUNNING.

Also, the executive allows for PREEMPTION of a task currently using processor resources so that emergencies will be responded to immediately. For example, suppose a conveyor in a manufacturing

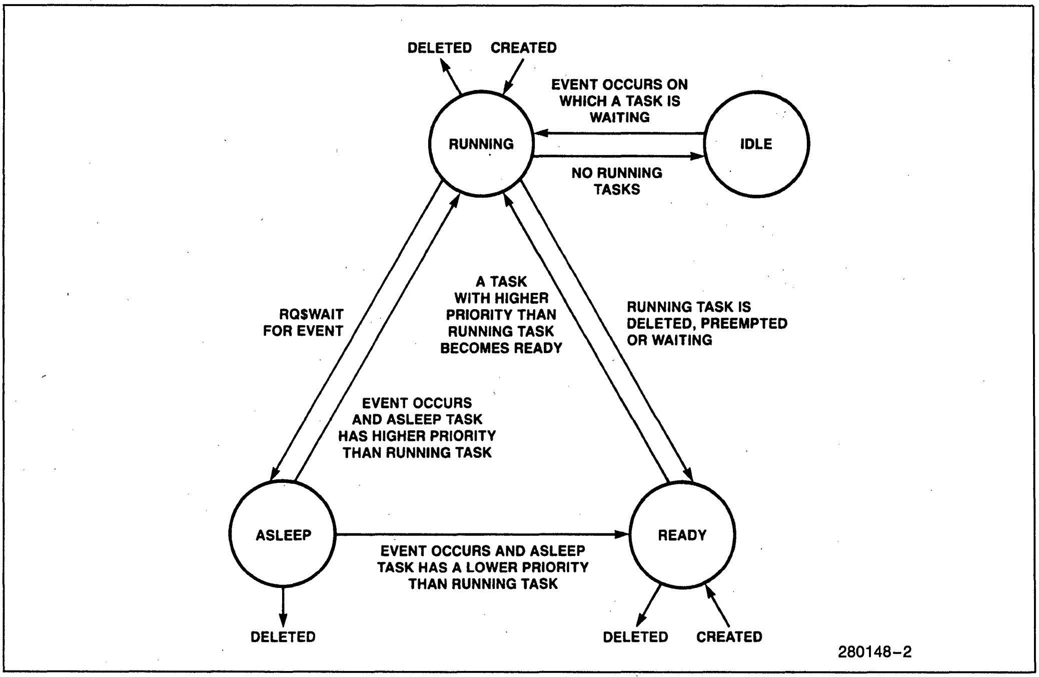

DELETED
CREATED
EVENT OCCURS ON WHICH A TASK IS WAITING
RUNNING
IDLE
NO RUNNING TASKS
A TASK WITH HIGHER PRIORITY THAN RUNNING TASK BECOMES READY
RQ$WAIT FOR EVENT
RUNNING TASK IS DELETED, PREEMPTED OR WAITING
EVENT OCCURS AND ASLEEP TASK HAS HIGHER PRIORITY THAN RUNNING TASK
ASLEEP
READY
EVENT OCCURS AND ASLEEP TASK HAS A LOWER PRIORITY THAN RUNNING TASK
DELETED
DELETED
CREATED
280148-2

Figure 2. iDCX 96 Executive Task States

system suddenly developed a fault and began running out of the normal range. The other parts of the system cannot compensate, and an alarm is triggered. Immediate response is a must to minimize losses. The executive's task prioritization scheme, task state definitions, and preemption facility reflect the asynchronous nature of events in real-time systems as well as the need to respond to the most critical events first.

Interrupt Handling

Interrupts signal the occurrence of an external event and are typically asynchronous with respect to the processor. In real-time control system interrupt handling plays a major factor in the responsiveness and performance of the system. The iDCX 96 Distributed Control Executive provides the following interrupt handling services and features:

- Interrupt source assignment to a task at system configuration.
- Ability to disable all or some interrupts using the RQ DISABLEINTERRUPT system call.
- Ability to enable disabled interrupts using the RQ ENABLEINTERRUPT system call.
- Synchronization of events using the RQ WAIT system call.
- Configuring a custom interrupt handler into the system.

In keeping with the executive's preemptive priority-based scheduling scheme for an interrupt to occur its associated task must have a higher priority than the present running task. The executive will mask all interrupts of lower priority.

The eight interrupt sources provided by the 8096 architecture are shown in Table 1. The iDCX 96 Executive architecture provides interrupt handlers for each source but allows users to substitute custom interrupt handlers if desired.

Table 1. 8096 Hardware Interrupt Sources

Source
EXTINT
Serial Port
HSI.0
High Speed Outputs
HSI Data Available
A/D Conversion Complete
Timer Overflow
Software Interrupt

Timer Management

The iDCX 96 Executive supplies timing management facilities for synchronizing timed control loops and determining how long tasks wait on an event. In multitasking environments tasks compete for timing resources. The executive eliminates contention for this resource by reserving one of the 8096 on-chip timers for software timing services. A software clock is maintained from this on-chip timer, and is used for system timing functions. Tasks request interval timing or timeout timing services via the iDCX 96 Executive appropriate system calls.

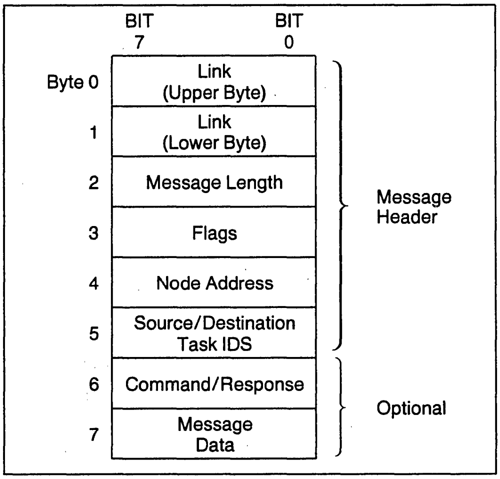
Message Passing

The iDCX 96 Distributed Control Executive facilitates intertask communication that allows tasks to:

- communicate with other tasks via messages
- wait indefinitely on a message event
- synchronize task operations throughout a system
- manage system resources

These services greatly simplify design of multitasking, real-time control applications by providing an extremely flexible method of communication. Because tasks in an iDCX 96 Executive system exchange messages via message queues the communicating tasks are independent of one another. Tasks can store messages not yet received and put messages in a buffer that have not yet been sent. The user simply invokes the relevant system calls when required (RQ ALLOCATE, RQ DEALLOCATE, RQ SENDMESSAGE, RQ WAIT).

The format of iDCX 96 messages follows the standard BITBUS™ Interconnect message format. Figure 3 shows the iDCX 96 Executive message format.



	BIT 7 ... BIT 0	
Byte 0	Link (Upper Byte)	Message Header
1	Link (Lower Byte)	
2	Message Length	
3	Flags	
4	Node Address	
5	Source/Destination Task IDS	
6	Command/Response	Optional
7	Message Data	

Figure 3. iDCX 96 Message Format

By implementing communications software, users can incorporate iDCX 96 Executive systems into a BITBUS Interconnect environment. Thus the executive supports communications in standalone and distributed control systems. Although users need to provide some communications software to implement communication between different microcontrollers, the support already provided in the executive gives users a head start in applications development.

HIGH PERFORMANCE AND EASE OF USE

To meet the dual requirements of high performance and ease of use, two interfaces are provided for each system call: a PL/M 96 interface and a register interface. The PL/M 96 interface provides a higher degree of ease of use thus speeding development time. For extremely demanding applications the register interface provides greater run-time speed and can be used with either PL/M 96 or ASM 96.

The iDCX 96 Executive's capabilities are invoked through a set of system calls. Table 2 includes a listing of these interfaces and their functions. All the system calls with the exception of RQ GET FUNCTION IDS have already been referenced in this document as part of the interrupt handling, message passing, and timing support facilities. The RQ GET FUNCTION IDS call allows the user to reference tasks by function rather than task number. This constant identifier facility remains valid even if functions are moved to different physical locations (e.g., another processor in a distributed system).

The iDCX 96 Distributed Control Executive executes a variety of services in about half the time the iDCX 51 Executive (formally iRMX™ 51 Executive) can. (The iDCX 96 Executive is a functional port of the iDCX 51 Executive to the MCS-96 family of microcontrollers.) Table 3 shows ADVANCE performance information for the iDCX 96 Executive.

Table 3. iDCX 96 Executive Performance

Function	iDCX 51 Time (μs)	iDCX 96 Time* (μs)
Interrupt Latency w/Context Switch	130	70
Interrupt Latency from Idle Stage	46	42
Interrupt Latency w/Custom Handler	N/A	16
RQALLOCATE	18	16
RQSEND = > Non-Waiting Task	98	46
RQSEND = > >Priority Waiting Task	172	90
RQSEND = > <Priority Waiting Task	137	66
RQWAIT on No Events	27	24

*Advance Information

Table 2. Functional Listing of System Calls

Task Management Calls	
RQCREATETASK RQDELETETASK RQGETFUNCTIONIDS	Create and schedule a new task. Delete the specified task from the system. Obtain the function IDs of tasks currently in the system.
Intertask Communication Calls	
RQALLOCATE RQDEALLOCATE RQSENDMESSAGE RQWAIT	Obtain a message buffer from the system buffer pool. Return a message buffer to the system buffer pool. Send a message to the specified task. Wait for interrupt, message, or interval.
Interrupt Management Calls	
RQDISABLEINTERRUPT RQENABLEINTERRUPT RQWAIT	Temporarily disable multiple interrupts. Reenable one or more interrupts previously disabled by RQDISABLEINTERRUPT. Wait for interrupt, message, or interval.
Time Management Calls	
RQSETINTERVAL RQWAIT	Establish a time interval. Wait for interrupt, message, or interval.

CONFIGURABLE

Aside from the interrupt handler variables noted previously, other system variables are made available to the user for system customization. Most of these variables must be defined during initial system configuration. Task-specific attributes like task priority, interrupt vectors, and function ID are assigned via the Initial Task Descriptor structure at configuration time. Table 4 shows the configuration constants accessible to the user. These configuration constants give the iDCX 96 Executive added flexibility to satisfy the users needs. Table 5 shows other USER AVAILABLE variables. Run-time variables reflect the condition of the running system. Development-time diagnostic variables also reflect conditions of the running environment, but are usually helpful during application development.

Also, the executive allows for adding additional tasks to an already configured system or changing initial configuration constants via an Initial Data Descriptor (IDD). The IDD structure lets the user redefine existing configuration constants without reconfiguring the entire system. Constants that may be redefined are the system: clock unit, clock priority, buffer pool address, buffer pool size, and buffer size.

Table 4. Configuration Constants

Constant Name	Description
RQMAXTASKS	The maximum number of tasks that can exist in the system at any given time.
RQMAXPRIORITY	The highest priority level that can be assigned to a task or to the system clock.
RQCLOCKPRIORITY	The priority level of the system clock.
RQCLOCKTICK	The number of time cycles in the system clock basic time unit (a 'tick").
RQSTACKPOOLADR	The starting address of the system stack pool.
RQSTACKPOOLLEN	The length, in bytes, of the system stack pool.
RQSYSPOOLADR	The starting address of the system buffer pool.
RQSYSPOOLLEN	The length, in bytes, of the system buffer pool.
RQSYSBUFSIZE	The size, in bytes, of each buffer in the system buffer pool.
RQFIRSTITD	The absolute address of the first ITD in the ITD/IDD chain.
RQDIAGNOSTICS	An entry point in which user-written power-up diagnostic code is added.

Table 5. System Variables Available to the User

Variable	Size	Access	Description
General Run-Time Variables			
RQTASKID	WORD	Read Only	Contains the ID of the running task
RQCLOCKUNIT	WORD	Read/Write	Specifies the unit of time for the system clock
RQBUFSIZE	WORD	Read Only	Specifies the size of the buffers in the system buffer pool
Development-Time Diagnostic Variables			
RQPRIORITY	WORD	Read Only	Contains the priority of the running task, or zero if the system is idle
RQINITSTATUS	WORD	Read Only	Specifies the system status at the end of the system initialization (low byte), and the ID of the last task initialized (high byte)
RQRUNSTATUS	BYTE	Read Only	Specifies certain occurences and conditions which exist during runtime
RQSTACKOVERFLOW	WORD	Read Only	Specifies which tasks, if any, may have stack overflow conditions

RELIABLE AND COMPACT

Real-time control applications require reliability. The iDCX 96 Distributed Control Executive requires 2.9K bytes of code space, 75 bytes of on-chip register RAM, and a minimum of 56 bytes of data RAM. This streamlined executive increases performance and reliability by providing a range of services in a minimal amount of code. The compact nature of the executive, in addition to its architecture, allows for incorporating it into PROM or the memory of the 8096 microcontroller further reducing component count of the total system.

The iDCX 96 Executive is completely tested and verified by Intel's stringent software evaluation process. Thus the user realizes higher system reliability with reduced effort by incorporating fully functional and tested software. Using the iDCX 96 Executive allows the software development team to focus on the application-specific parts of a project.

The modular nature of the executive also enhances reliability by allowing user tasks to be refined independently. In this way, errors can be isolated more easily and corrected in each specific module. Using the iDCX 96 Executive for MCS-96 microcontroller application development reduces risk and development time.

OPERATING ENVIRONMENT

The iDCX 96 Executive will operate on any of the MCS-96 Family of microcontrollers. Tables 6 and 7 show the product family and a summary of the MCS-96 Family features and benefits.

Table 6. MCS®-96 Family of Products

Options		68 Pin	48 Pin
Digital I/O	ROMless	8096	8094
	ROM	8396	8394
Analog and Digital I/O	ROMless	8097	8095
	ROM	8397	8395

The 48 pin version is available in DIP (dual inline) package.
The 68 pin version comes in two packages, the plastic Flatpack and the Pin Grid Array.

Table 7. MCS®-96 Features and Benefits Summary

Features	Benefits
16-Bit CPU	Efficient machine with higher throughput.
8K Bytes ROM	Large program space for more complex, larger programs.
	Large on-board register file.
Hardware MUL/DIV	Provides good math capability 16 by 16 multiply or 32 by 16 divide in 6.5 μs @ 12 MHz.
6 Addressing Modes	Provides greater flexibility of programming and data manipulation.
High Speed I/O Unit 4 dedicated I/O lines 4 programmable I/O lines	Can measure and generate pulses with high resolution (2 μs @ 12 MHz).
10-Bit A/D Converter	Reads the external analog inputs.
Full Duplex Serial Port	Provides asynchronous serial link to other processors or systems.
Up to 40 I/O Ports	Provides TTL compatible digital data I/O including system expansion with standard 8- or 16-bit peripherals.
Programmable 8 Source Priority Interrupt System	Respond to asyncchronous events.
Pulse Width Modulated Output	Provides a programmable pulse train with variable duty cycle. Also used to generate analog output.
Watchdog Timer	Provides ability to recover from software malfunction or hardware upset.
48 Pin (DIP) & 68 Pin (Flatpack, Pin Grid Array) Versions	Offers a variety of package types to choose from to better fit a specific application need for number of I/O's and package size.

DEVELOPMENT ENVIRONMENT

Intel provides a complete development environment for the MCS-96 Family of microcontrollers. The iDCX 96 Executive is only one of many of the software develoment products available. Figure 4 shows the iDCX 96 Executive development environment. The executive is compatible with the following software development utilities available from Intel:

- 8096 Macro Assembler (ASM 96)
- PL/M 96 Complier
- RL 96 Linker and Relocator Program
- LIB 96
- FPAL 96 Floating Point Arithmetic Library

Hardware development tools available for MCS-96 microcontrollers

- iSBE-96, Single Board Emulator for the MCS-96 Family of Microcontrollers
- VLSiCE-96 In-Circuit Emulator

Table 8 shows the possible MCS-96 Family development environments: host systems, operating systems, available software utilities, and hardware debug tools.

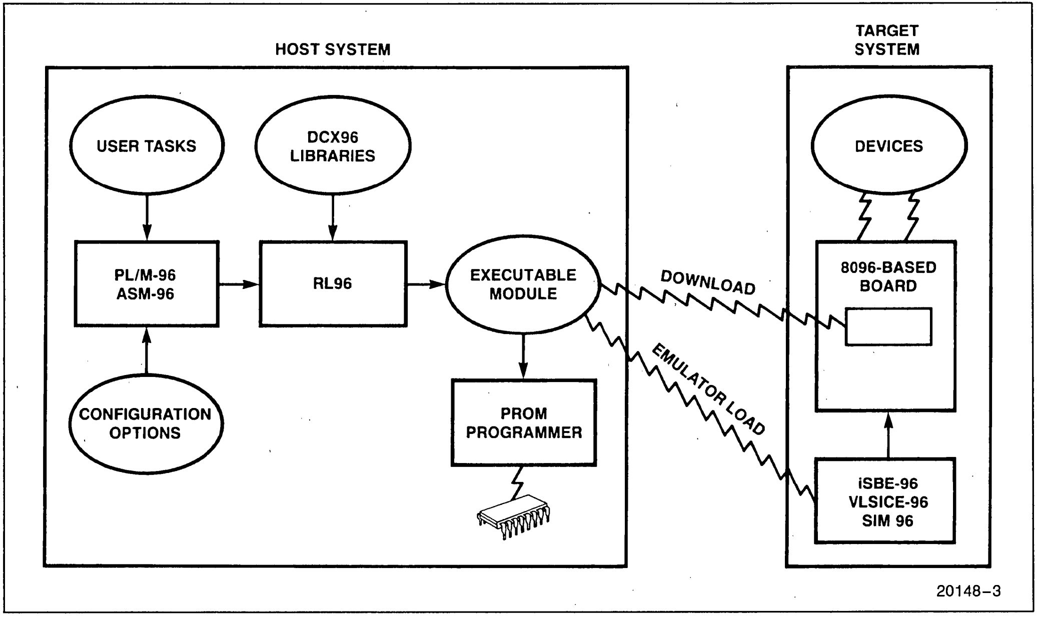


Figure 4. iDCX 96 Development Environment

Table 8. MCS®-96 Family Development Environments

Development Utilities	Host Systems		
Software	**Intellec® Series III/IV Systems**	**iPDS™ System**	**IBM** -PC System**
MCS® 96 Software Support Package (ASM96)	X		X
PL/M 96 Software Package	X		X
iDCX 96 Executive	X	X	X
XASM96, COMM96, ATOP 96*		X	
Hardware			
iSBE-96, Single Board Emulator	X	X	X
VLSiCE-96, In-Circuit Emulator	X		X

*Products of U.S. Software, Portland, OR.
**IBM is a registered trademark of International Business Machines.

SPECIFICATIONS

Hardware

MCS-96 Family of Microcontrollers

8094	8394
8095	8395
8096	8396
8097	8397

DEVELOPMENT ENVIRONMENT

Software

MCS-96 Software Support Package
PL/M-96 Software Package

iPDS System Host:

*XASM96 Assembles MCS-96 programs on the iPDS™
*COM96 iPDS host communication software. Use with XASM96
*ATOP96 Performs host communications and assembly/disassembly of iSBE-96 instructions. Use with XASM96.

*Products of U.S. Software
5470 N.W. Innisbrook, Portland, OR 97229
Phone: 503-645-5043
Telex: 4993875

Hardware

SYSTEMS

Intellec Microcomputer Development System, Series III/IV
iPDS Intel Personal Development System
IBM Personal Computer

DEBUG TOOLS

SBE-96 Single Board Emulator for MCS-96 Family of Microcontrollers
VLSiCE 96 In-Circuit Emulator

Reference Manual (Supplied)

148107-001 iDCX 96 Distributed Control Executive User's Guide

ORDERING INFORMATION

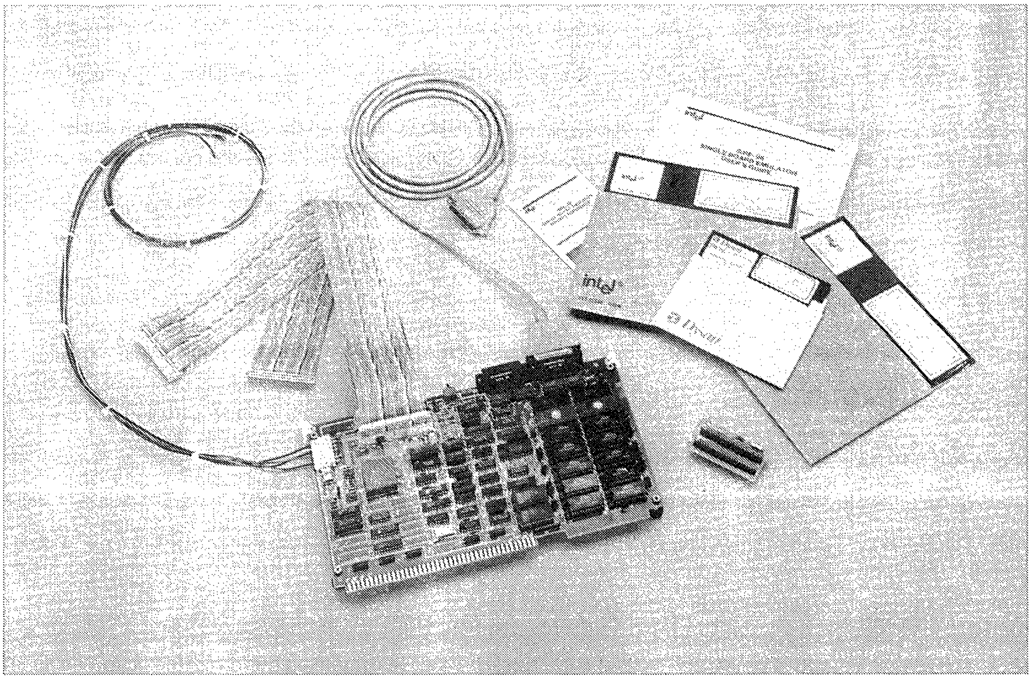
Part Number	Description
iDCX96SU	Executive for the MCS-96 Family of Microcontrollers Single User License, Development Only Media Supplied: B, E, F, J and I
iDCX96BY	Executive for the MCS-96 Family of Microcontrollers OEM License, Derivative Products Media Supplied: B, E, F, J and I

iSBE-96 DEVELOPMENT KIT SINGLE BOARD EMULATOR AND ASSEMBLER FOR THE MCS®-96 FAMILY OF MICROCONTROLLERS

- **Hosts**
  - **—Intellec® Series III/IV Development Systems**
  - **—IBM* PC AT, PC XT, and Compatibles (3.0)**
- **Eight Software Execution Breakpoints That Can Selectively Be Turned On and Off**
- **12 MHz Emulation Speed**
- **Single Line Assembler/Disassembler**
- **MCS®-96 Software Support Package**
- **Configurable Serial I/O**
- **17.75 of On-Board User Memory**
- **Optionally Expandable to 64K of On-Board User Memory**

The iSBE-96 emulator supports the execution and debugging of programs for the MCS-96 family of microcontrollers at speeds up to 12 MHz. The MCS-96 family configurations are shown in Table 1. The iSBE-96 emulator consists of an 8097 microcontroller, a serial port and cables, and an EPROM-based monitor that controls emulator operation and the user interface.

The iSBE-96 emulator is a combination of hardware and software that permits programs written for the MCS-96 family of microcontrollers to be run and debugged in the emulator's artificial environment or in the user's prototype system. As a result, development time can be reduced by the early integration of hardware and software.



231015–1

April 1987
Order Number: 231015-006

FUNCTIONAL DESCRIPTION

Integrated Hardware and Software Development

The iSBE-96 emualtor allows hardware and software development to proceed simultaneously. This approach is more time- and cost-effective than the alternate method: independent hardware and software development followed by system integration. With the iSBE-96 emulator, prototype hardware can be added to the system as it is designed; software and hardware integration occurs while the product is being developed. The emulator aids in the recognition of hardware and software problems.

Emulation is the controlled execution of the prototype software in the prototype hardware or in an artificial hardware environment that duplicates the microcontroller of the prototype system. The iSBE-96 emulator permits reading and writing of system memory, and control of program execution. The emulator also allows interactive debugging of the prototype software and can externally control program execution while operating in the prototype system. When the prototype system memory is not yet available, the iSBE-96 emulator's on-board memory permits software debugging.

Table 1. The Configurations of the MCS®-96 Family of Microcontrollers

		68 Pin	48 Pin
Digital I/O	ROMLESS	8096	
	ROM	8396	
	EPROM	8796	
Analog and Digital I/O	ROMLESS	8097	8095
	ROM	8397	8395
	EPROM	8797	8795

iSBE-96 Software

The iSBE-96 emulator software is available for use with the following host systems:

- Intellec Series III and Series IV development systems
- IBM PC/AT and PC/XT computer systems

The iSBE-96 emulator software is also available from U S Software* for use on the Intel Personal Development System (iPDS™) and the Intellec Series II development system.

***NOTE:**
U S Software is a registered trademark of United States Software Corporation.

The iSBE-96 emulator is supplied with a driver routine that communicates with the monitor software on the iSBE-96 emulator board through serial channel 1 or 2 (com1/com2). The driver interrupts the 8097 using the non-maskable interrupt (NMI) line for incoming keyboard input. The commands associated with the driver and the monitor are described in the following sections.

iSBE-96 Driver

iSBE-96 emulator is shipped with driver software for use on the Series III/IV development systems and the IBM PC AT/XT running PC DOS, version 3.0 or greater. The driver software provides a few easy-to-use commands. These commands are described in Table 2. ASM/DASM available on DOS version only.

Table 2. iSBE-96 Driver Commands

Driver Command	Function
ASM	Loads memory with MCS-96 assembly mnemonics.
DASM	Displays memory as MCS-96 assembly mnemonics.
EXIT	Exits the driver and returns to the host operating system.
<CONTROL> C	Same as for the EXIT command.
HELP	Displays the syntax of all commands.
INCLUDE	Specifies a command file.
<CONTROL> I	Turns the command file on and off.
<TAB>	Same as <CONTROL> I (turns the command file on and off).
LIST	Specifies a list file.
<CONTROL> L	Turns list file on and off.
<CONTROL> S	Stops scrolling of the screen display.
<CONTROL> Q	Resumes scrolling of the screen display.
<CONTROL> X	Deletes the line being entered.
<ESCAPE>	Aborts the command executing.

iSBE-96 MONITOR

The iSBE-96 monitor performs the following functions:

- Loads and saves user programs.
- Independently emulates user programs.

Table 3. ISBD Monitor Commands

Monitor Command	Function
BAUD	Sets up the baud rate.
BR	Permits display and setting of up to eight software breakpoints.
BYTE	Permits display and changing of a single byte or range of bytes of memory or a single byte of the 8097 internal registers.
CHANGE	Permits display and changing of a series of memory words or bytes.
<CONTROL> S	Stops scrolling of the screen display.
<CONTROL> Q	Resumes scrolling of the screen display.
<CONTROL> X	Deletes the line being entered.
<ESCAPE>	Aborts the command executing.
GO	Begins emulation and continues until an enabled breakpoint is reached or the escape key is pressed.
LOAD	Loads programs and data from disks.
MAP	Permits mapping of several preprogrammed memory maps; also permits configurable serial I/O and selective servicing of the watchdog timer.
PC	Displays and changes the program counter.
PSW	Displays and changes the program status word.
RESET CHIP	Resets the 8096 to power-up conditions.
SAVE	Saves programs and data to disks.
SP	Displays and changes the stack pointer.
STEP	Provides single-step emulation with selective display formats.
VERSION	Displays the monitor version number.
WORD	Permits display and changing of a single word or range of words of memory or a single word of the 8097 internal registers.

- Examines and changes memory contents.
- Examines registers.
- Maps the file capabilities of the serial ports (DS/DT).
- Maps different memory configurations.

The monitor commands are described in Table 3.

Integrating Hardware and Software

When the prototype system hardware is developed, the iSBE-96 emulator interfaces to the prototype through two 50-pin ribbon cables. The emulator can then execute code from the iSBE-96 on-board RAM (or from user-provided memory) and exercise the prototype system hardware.

BLOCK DIAGRAM

Figure 1 is a block diagram showing the iSBE-96 emulator. The following sections describe each block.

The Processor

The 68-pin processor of the iSBE-96 emulator is used only in the 8097 external-access mode. An 8097BH will be supported in 16-bit bus mode only.

An adapter board is provided for the 68-pin PGA version of the 8096 and 8097 microcontrollers. When debugging a 68-pin package, the two 50-pin ribbon cables plug into the 68-pin adaptor board which is plugged into the 68-pin socket on the prototype system.

When debugging a 48-pin package, the two 50-pin cables plug into the 48-pin adaptor board, which is then plugged into a 48-pin socket in the prototype system. A 68-pin PLCC Adaptor may be ordered.

iSBE-96 Emulator I/O

The iSBE-96 emulator's memory-mapped I/O devices are used to manage the system. These I/O devices are mapped into memory between locations 01F00H and 01FFFH.

Included as part of the I/O are two serial ports. One is configured as data set (DS) and the other as data terminal (DT). When operating with an Intellec® development system, the data set port is used as the system console and the link for exchanging files.

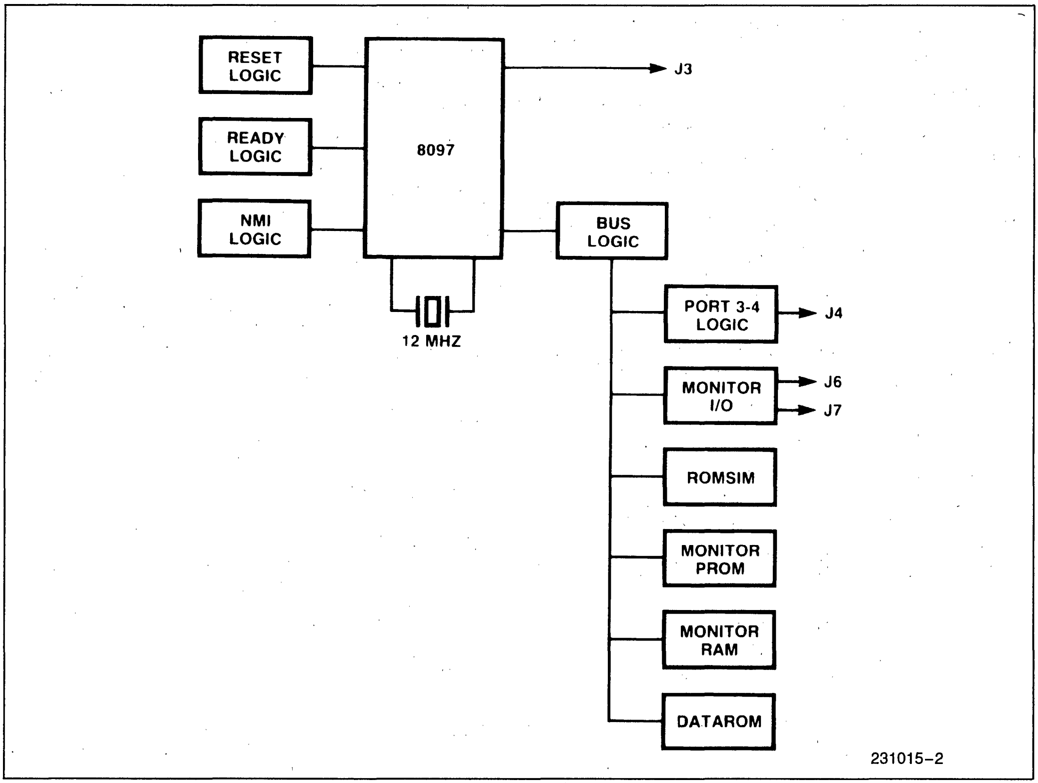


Figure 1. Block Diagram for the iSBE-96 Single Board Emulator

The serial ports are serviced under control of the NMI interrupt. The NMI interrupt has highest priority on the microcontroller and interrupts the user program when characters are entered from the keyboard. When in emulation, the monitor will still service inputs from the keyboard and execute certain monitor commands. Monitor activity is not transparent to the user.

Simulated ROM (ROMSIM)

There are eight 28-pin JEDEC byte-wide sockets with 2K-by-8 static RAMS present on the board. The partition on the user's prototype system that will be ROM is simulated by RAM on the iSBE-96 emulator board. This RAM facilitates easy program development, allowing users to correct and test problems in their programs.

ROMSIM can be expanded by replacing the iSBE-96 RAMs with 8K-by-8 static RAMs.

Port 3-4 Logic

The port 3-4 logic has two functions: to provide bus expansion and to provide I/O ports. The port 3-4 logic is controlled by a software switch available with the MAP command.

The iSBE-96 emulator reconstructs ports 3 and 4 of the 8395, 8396, and 8397 microcontrollers when the logic is defined by the MAP command as port 3-4. This port function should be selected when one of these four microcontrollers is intended as the target microcontroller.

When the BUS switch of the MAP command is specified, the iSBE-96 address/data expansion bus is available to the prototype system.

THE iSBE-96 EMULATOR MEMORY MAP

The target system should be designed with a memory map that is compatible with one of the iSBE-96

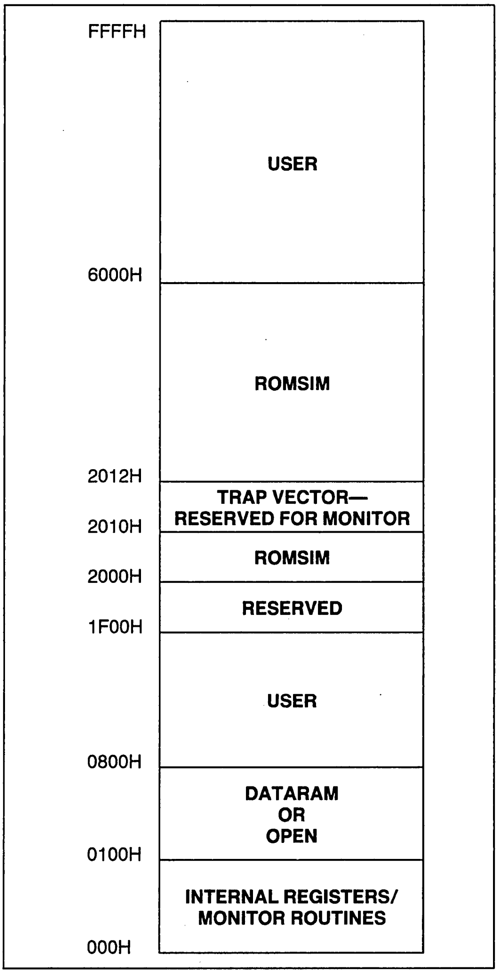


Figure 2. iSBE-96 Emulator Default Mapping

memory maps. Figure 2 shows the default address mapping. The following sections describe the areas of memory.

Internal Registers/Monitor Routines

Normally locations 000H through 0FFH contain the internal register space of the 8097. However, instruction fetches from these locations access external memory. This memory space contains the monitor's non-maskable interrupt service routine and utility routines.

For the monitor to access the user memory, the address and data is passed to the interrupt or utility routines. The routines then modify the mode register to enable user memory, disable all of the monitor's memory (except page zero), and perform the appropriate operation. After an operation is complete, the service and utility routines restore the mode register to its previous state and return to the main monitor code. The NMI service routine is used to handle the keyboard input on the serial port.

DATARAM

Locations 100H to 7FFH are mapped as the DATARAM space. This RAM is for general purpose use and is optionally enabled by using the MAP command. When the DATARAM buffer is not enabled, any access to this partition results in an access to user prototype system memory.

User Area

Locations 800H to 1EFFH are a user area. If an access is made to this partition, it is directed to the user's prototype system. Any memory mapped as I/O in the user system should be placed in this partition. With 8K-by-8 static RAMs, this area is located and available on the iSBE-96 board.

Reserved Area

Locations 1F00H to 1FFFH are reserved by the monitor for on-board I/O devices.

ROMSIM

Because some of the MCS-96 family of microcontrollers are ROMLESS parts, a user program can be loaded for execution into the on-board RAMS of the iSBE-96 emulator. Locations 2000H to 5FFFH are mapped to this RAM space; the space is called ROMSIM.

Trap Vector

Locations 2000H to 2010H are the interrupt vector locations. Vector address location 2010H is used by the iSBE-96 monitor for breakpoints.

User Area

The partition 6000H to 0FFFFH is mapped to the user prototype area. During emulation any access to this partition is directed to the user's prototype system.

EXPANDING ON-BOARD MEMORY

On-board memory can be expanded to a full 64K bytes by replacing the supplied 2K-by-8 static RAMs with 8K-by-8 static RAMs or PROMs. The user may also replace on-board ROMSIM memory with 2K-by-8 PROMs or even locate all 64K bytes of memory on the prototype system.

DESIGN CONSIDERATIONS

Designers should note the following considerations for designing with the iSBE-96 emulator:

- The iSBE-96 software uses 6 bytes of user stack space.
- Analog signal accuracy is impaired when driven over the emulator cable (up to ±50 mV loss of A/D conversion accuracy).
- The iSBE-96 emulator has some ac/dc parametric differences from the 8097 chip.
- The NMI vector is used for console service (Intel reserved interrupt).
- Keyboard activity during emulation affects real-time emulation because a 50 to 100 microsecond interrupt service routine is executed for every keystroke.
- The only hardware reset available for the iSBE-96 emulator is the system reset momentary switch (switch 1 on the emulator board).
- User system memory should be configured to the iSBE-96 memory map (see Figure 2).
- The iSBE-96 emulator does not support a user system crystal as shipped.
- The iSBE-96 driver software provided by Intel is not compatible with the Intellec Model 800 or Series II Development Systems.
- The IBM PC/AT and PC/XT have been evaluated and accepted by Intel as compatible hosts for its development systems. Intel has not evaluated any ohter PC DOS machines (3.0). However, Intel knows of no reason why these PC DOS machines would not be compatible hosts for its development systems.

SPECIFICATIONS

Equipment Supplied

Standard MULTIBUS®-size board assembly

EPROM-based monitor

Auxiliary power cable

RS-232 serial cables

Two standard, 18 in., 50-pin ribbon cables for connection to the user's prototype system

Adapter board for the 48-pin DIP and 68-pin PGA versions of the MCS-96 microcontroller

MCS-96 software support package

One 8 in. single-density software disk for the Series III

One 8 in. double-density software disk for the Series III

One 5¼ in. software disk for the Series IV

One 5¼ in. software disk for the IBM PC AT/XT

Documentation

ISBE-96 User's Guide (Order number 164116)
iSBE-96 Pocket Reference (Order number 164157)
Developing MCS-96 Applications Using iSBE-96 (Order Number 280249-001, AP-273)

Emulation Clock

12 MHz supplied crystal

Physcial Characteristics

Width: 6.75 in. (17.15 cm)
Length: 12 in. (30.48 cm)
Height: 0.75 in. (1.91 cm)

DC Electrical Requirements

Voltage	Current
+5V ± 5%	3.5a max
+12V ± 5%	0.06a max
−12V ± 5%	0.05a max

Environmental Characteristics

Operating Temperature:	10°C to 40°C
Operating Humidity:	10% to 85% relative humidity, without condensation

IBM PC XT/AT Host Requirements

- PC DOS, version 3.0 or greater
- External power supply
- Serial channel Com1/Com2

ORDERING INFORMATION

Intel 3065 Bowers Ave.
Santa Clara, CA 95051

Part Number	Description
SBE96SKIT	iSBE-96 single board emulator for use with the Series III/IV development systems. The kit contains the following parts: • iSBE-96 single board emulator • MCS-96 software support package for the Series III/IV development systems • iSBE-96 Series III/IV upgrade kit (cables and software needed to run on Intel Hosts)
SBE96DKIT	iSBE-96 single board emulator for use with the IBM PC/AT and PC/XT computer systems. The kit contains the following parts: • iSBE-96 single board emulator • MCS-96 software support package for PC DOS • iSBE-96 DOS upgrade kit (cables and software needed to run on the IBM PC/AT or PC/XT)
SBE96DU	iSBE-96 DOS upgrade kit for those customers who wish to upgrade their Series III/IV kit to run on the IBM PC AT or PC XT.
SBE96SU	iSBE-96 Series III/IV upgrade kit for those customers who wish to upgrade their DOS kit to run on Intel Hosts).
TASBEE	68-pin PLCC Adaptor Board.

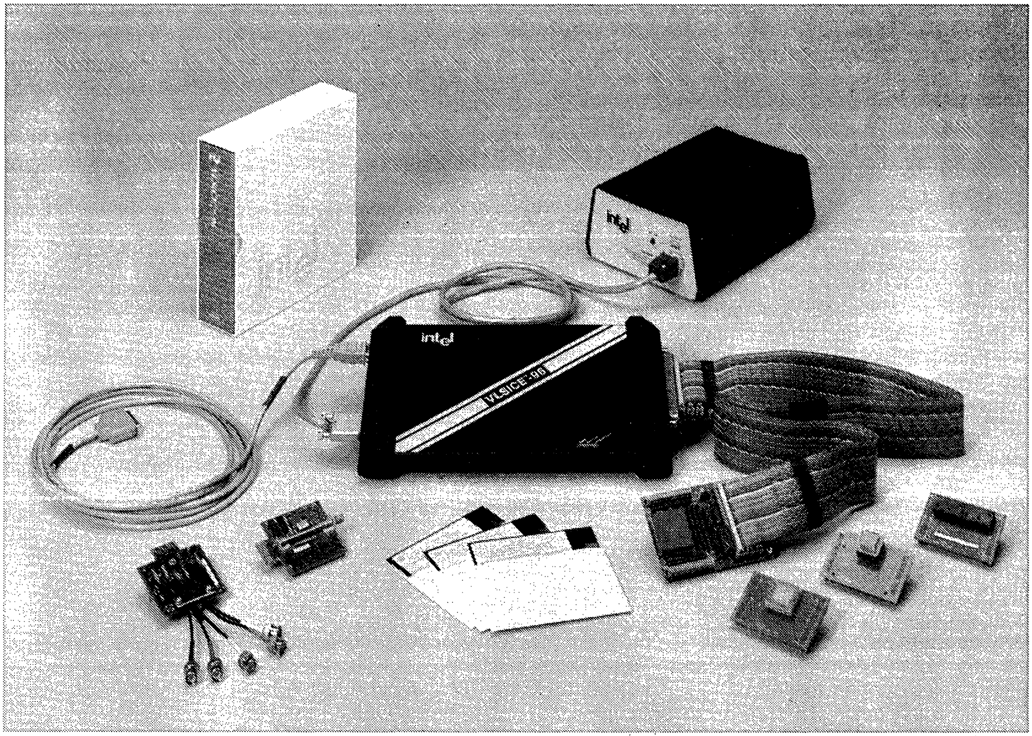
U S Software
5470 N. W. Innisbrook
Portland, OR 97229
Phone: 503-645-5043
International Telex 4993875

Part Number	Description
XASM96	Performs assembly of MCS®-96 programs written on the iPDS.
ATOP96	iPDS and Series II software for iSBE-96 host communications. Performs host communications and assembly/disassembly of iSBE-96 instructions. The XASM Host Cross Assembler software must be ordered with this software.

VLSiCE-96
IN-CIRCUIT EMULATOR FOR THE 8X9X FAMILY OF MICROCONTROLLERS

- Precise Real-Time Emulation of the 8X9X Family of Components
- 64K of Mappable Memory for Early Software Debug and (EP)ROM Simulation
- A 4K-Entry Trace Buffer for Storing Real-Time Execution History, Including Both Code and Data Flows
- Fastbreaks and Dynamic Trace
- Symbolic Debugging Allows Accesses to Memory Locations and Program Variables (Including Dynamic Variables) Using Program-Defined Names from the User's Assembler or Compiler Source Code
- Shadow Registers Allow Reading Many 8096 Write-Only and Writing Many Read-Only Registers
- Break and Trace are Qualified on Execution Addresses, Data Addresses, and Values (Both External and Internal RAM), Opcodes, Selected PSW Flags, and 2 External Sync Lines
- Equipped with the Integrated Command Directory (ICD™) Which Provides
  - — An On-Line Help File
  - — A Dynamic Syntax Menu
  - — Dynamic Command-Entry
  - — Error Checking
  - — On-Line Editor
- Serially Linked to Intel Series III/IV Hosts or IBM* PC-XT and AT

The VLSiCE-96 In-Circuit Emulator is a debugging and test tool used for development of the hardware and software of a target system based on the 8X9X family of microcontrollers (8095, 8096, 8097, 8395, 8396, 8397, 8795, 8796, 8797, 8098, 8398, 8798) including BH components.



280140–1

*IBM is a trademark of International Business Machines.

November 1987
Order Number: 280140-004

INTRODUCTION

The VLSiCE-96 emulator allows hardware and software development of a design project to proceed simultaneously. With the VLSiCE-96 emulator, prototype hardware can be added to the system as it is designed and software can be developed prior to the completion of the hardware prototype. Thus, software and hardware can be integrated while the product is being developed.

The VLSiCE-96 emulator assists four stages of development:

- Software development
- Hardware development
- System integration
- System test

Software Development

The VLSiCE-96 emulator can be operated without being connected to a prototype or before any of the prototype hardware is available. In this stand-alone mode, the VLSiCE-96 emulator can be used to facilitate application program development.

Hardware Development

Because the VLSiCE-96 emulator precisely matches the component's electrical and timing characteristics as well as full bus access, it is a valuable tool for hardware development and debug.

System Integration

Integration of software and hardware begins when the microcontroller socket is connected to any functional element of the target system. As each section of the user's hardware is completed, it is added to the prototype. Thus, each section of the hardware and software can be system tested with the VLSiCE-96 emulator in real-time operation as it becomes available.

System Test

When the prototype is complete, it is tested with the final version of the system software. The VLSiCE-96 emulator can then be used to verify or debug the target system as a completed unit.

By providing support for the ROMLESS, ROM, and EPROM versions of the microcontroller, the VLSiCE-96 emulator has the ability to debug a prototype or production product at any stage in its development without introducing extraneous hardware or software test tools.

PHYSICAL DESCRIPTION

The VLSiCE-96 emulator consists of the following components (see Figure 1):

- Software (includes the VLSiCE-96 emulator software, diagnostic software, and tutorial)
- 68-pin PGA adaptor
  68-pin PLCC adaptor (optional)
  48-pin DIP adaptor (optional)
- Controller pod
- User cable assembly (consisting of the user cable and processor module)
- Serial cable (host-specific)
- Crystal power accessory (CPA)
- Multi-synchronous accessory (MSA) (optional)
- Power supply and V_{CC} booster module
- AC and DC power cables

VLSiCE-96 software fully supports all mnemonics, object file formats, and symbolic references generated by Intel's ASM-96, PL/M-96, and C-96.

The on-line tutorial is written in VLSiCE-96 command language. Thus, the user is able to interact with and use the VLSiCE-96 emulator while executing the tutorial.

The controller pod contains 64K of ICE memory, a 4K-entry trace buffer, and circuitry that provides communication between the host and the emulator.

The processor module contains a special version of the Intel 8096 microcontroller, called the emulation processor. This chip performs real-time and single-step execution of a program's object code for execution and debugging purposes in place of the target system microcontroller.

The crystal power accessory (CPA) is a small detachable board that connects to the back of the controller pod and is used to run the VLSiCE-96 emulator in the stand-alone mode. It is also used when running customer confidence tests. In the stand-alone mode, the user plug on the user cable is connected through the 68-pin PGA adaptor to the CPA. The CPA supplies clock and power. Stand-alone mode is used to test and debug software prior to the availability of hardware.

The optional multi-synchronous accessory can be used to connect the VLSiCE-96 emulator with up to 20 multi-ICE compatible emulators together for synchronous GO and BREAK emulation, and TRACE.

It can also be used with other debug equipment such as logic analyzers and oscilloscopes for synchronous GO, BREAK and TRACE.

The serial cable connects the host system to the controller pod. The serial cable has electrical specifications similiar to the RS-232C standard.

The power supply connects to the controller pod via the V_{CC} booster module and the DC power cable. There are several voltage options available for the power supply depending on switch settings on the back of the power supply.

A comprehensive set of documentation is included with the VLSiCE-96 emulator.

Figure 1 shows a drawing of the VLSiCE-96 emulator.

VLSiCE-96 EMULATOR FEATURES

The VLSiCE-96 emulator assists hardware and software design engineers in developing, debugging and testing designs incorporating the 8X9X family of microcontrollers. The following are some of the VLSiCE-96 features:

Emulation

Emulation is the controlled execution of the prototype software in the prototype hardware or in an artificial hardware environment that duplicates the microcontroller of the target system. With the VLSiCE-96 emulator, emulation is a transparent process that happens in real-time without sacrificing microcontroller resources. The execution of prototype software is facilitated through the VLSiCE-96 command language.

Memory Mapping

There are 64 Kbytes of zero-waitstate, high-speed mappable memory available. This memory space can be mapped to either the target system or to the on-board VLSiCE-96 memory space in 1 Kbyte blocks on 1 Kbyte boundaries. Mapping memory to the VLSiCE-96 emulator allows software development to proceed before prototype hardware is available. Memory mapping also gives the VLSiCE-96 emulator the capability to simulate the 8 Kbytes of (EP)ROM on those versions of the chip for code verification and validation.

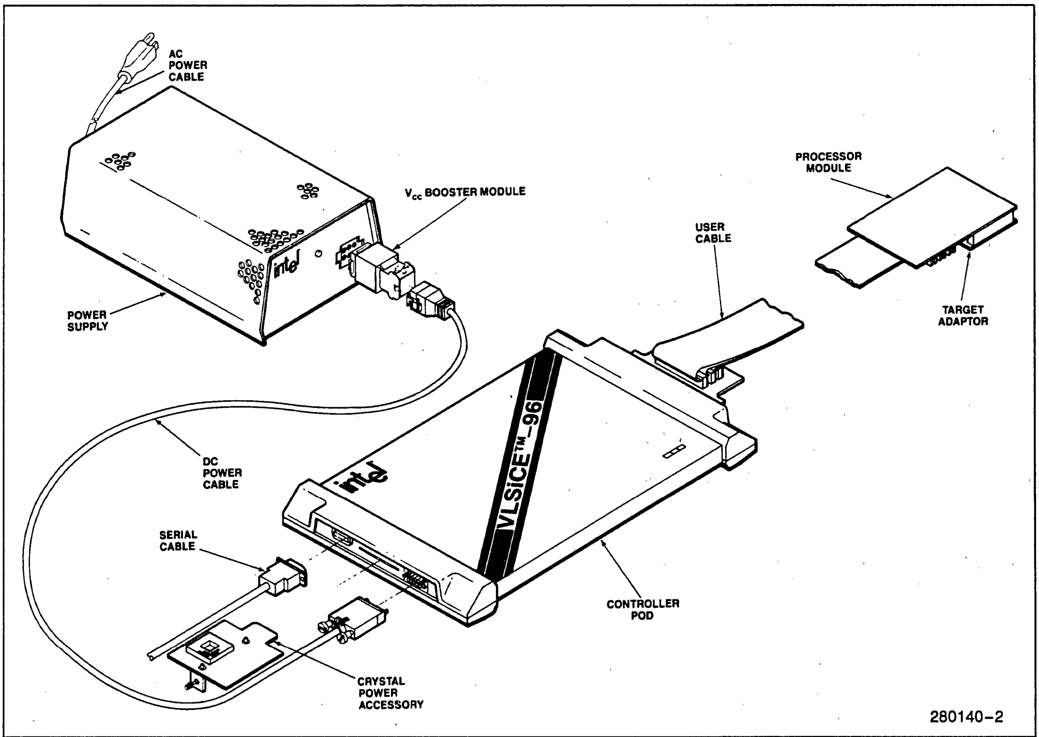


Figure 1. The VLSiCE™-96 Emulator

Memory Examination and Modification

The memory space for the 8X9X component and its target hardware is accessible through the emulator. The VLSiCE-96 software allows the component's special function registers to be accessed mnemonically (e.g. AD_RESULT, INT_MASK). A significant benefit to the VLSiCE-96 is its ability to read many of the write-only registers (e.g. IOC0, PWM_CONTROL) and to write many of the read-only registers (e.g. AD_RESULT, SBUFRX).

Data can be displayed or modified in several bases: hex, decimal, and binary, and in standard formats including: ASCII, real and integer. Program code can be disassembled and displayed as assembler mnemonics. It also can be modified with standard assembler statements.

Memory locations can also be examined or modified by their symbolic references. A symbolic reference is a procedure name, line number, or label in the user program that corresponds to a memory location.

Some typical symbolic functions include:

- Changing or inspecting the value of a program variable by using its symbolic name, rather than the address of the memory lacation.
- Defining break and trace events using symbolic references.
- Referencing static variables, dynamic (stack-resident) variables, based variables, and record structures combining primitive data types. The primitive data types are ADDRESS, BOOLEAN, BYTE, CHAR (character), WORD, DWORD (double word), INTEGER, LONGINT, SHORTINT, and REAL.

The VLSiCE emulator maintains a virtual symbol table for program symbols making it possible for the table to exist without fitting entirely into host RAM memory. The size of the virtual symbol table is constrained only by the capacity of the disk.

Breakpoint Specifications

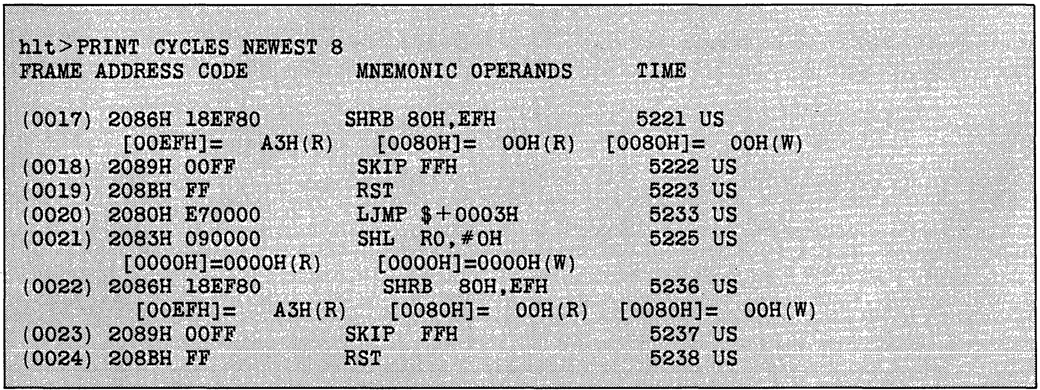
Breakpoints allow halting of a user program in order to examine the effect of the program's execution on the target system. Breakpoints can be defined as execution addresses, data addresses and data values (both external and **internal RAM**), opcodes, selected bits of the PSW flag, and as 2 external inputs (SYNC0IN AND SYNC1IN). These breaks can be arranged to occur over a range of addresses and with up to 8 levels of arming and disarming. After a break the user program can resume execution from where it left off.

Trace Specifications

Tracing can be triggered with the same conditions set for breaking. The trace buffer is displayed as disassembled instructions, data fetches and stores, and with the timetag showing the relative time at which the program executed each instruction. Figure 2 shows a trace display as a result of the PRINT command.

Normally, the VLSiCE-96 emulator traces program activity while the user program executes. With a trace specification, tracing can be specified to occur only when specific conditions are met during execution. The trace buffer collects data for up to 4 Kbyte entries of information during emulation.

The trace buffer can be examined during halt mode or if non-stop emulation is desired; the trace can be examined while emulation continues. If this second option is selected, trace collection stops while the trace buffer is uploaded to the host.



```
hlt>PRINT CYCLES NEWEST 8
FRAME ADDRESS CODE           MNEMONIC OPERANDS      TIME

(0017) 2086H 18EF80         SHRB 80H,EFH           5221 US
         [00EFH]=   A3H(R)   [0080H]=  00H(R)  [0080H]=  00H(W)
(0018) 2089H 00FF            SKIP FFH                5222 US
(0019) 208BH FF              RST                     5223 US
(0020) 2080H E70000          LJMP $+0003H            5233 US
(0021) 2083H 090000          SHL  R0,#0H             5225 US
         [0000H]=0000H(R)    [0000H]=0000H(W)
(0022) 2086H 18EF80            SHRB  80H,EFH         5236 US
          [00EFH]=   A3H(R)   [0080H]=  00H(R)  [0080H]=  00H(W)
(0023) 2089H 00FF           SKIP  FFH                5237 US
(0024) 208BH FF             RST                      5238 US
```

Figure 2. The Trace Buffer Display

Arming and Triggering

The VLSiCE-96 command language allows specification of complex events with up to 8 states, each with several conditions. For example, a specification can be made that causes a break to occur when a variable is written only within a certain procedure. The execution of the procedure is the arm condition and the variable modification is the break condition. The arm condition is an optional part of a break/trace sequence in the VLSiCE emulator. A set of arm conditions can be used to ensure that a break is not possible until all required qualifying conditions are satisfied.

Procedures

Debugging procedures (PROCS) are a user-named group of VLSiCE commands that are executed sequentially. Procs can simulate missing hardware or software, collect debug information, and make troubleshooting decisions. They can be copied to text files on disk, then included from the file into the command sequence in later test sessions.

Procedures can also serve as programmable diagnostics, implementing new emulator commands for special purpose.

FASTBREAKS

Fastbreaks make it possible to examine and modify memory without halting emulation. The commands that can be executed are simple one-access functions, such as, WORD 1FH or IOS0. When enabled, fastbreaks occur whenever a memory access is made.

Breakpoints and tracepoints can be re-specified during emulation with fastbreaks enabled.

While emulation does not halt during fastbreaks, a delay in code execution occurs when a fastbreak is requested. In most cases, this latency in code execution is less than 150 μs.

Interrupts During Interrogation (IDI) Mode

The VLSiCE-96 software can service and record interrupts even though emulation has been halted (interrogation mode). In the special mode designated as IDI mode, hardware interrupts can be serviced while the emulator is being interrogated. Use of this mode is determined by the setting of a VLSiCE-96 pseudo-variable (IDI_PC). After breaking from emulation or fastbreaks mode, whenever an interrupt occurs, the processor jumps to the appropriate vector and executes the interrupt service routine.

The setting of another VLSiCE-96 pseudo-variable (INT_REC_EN) allows the recording of interrupts but not the servicing of interrupts, during halt mode. If the pseudo-variable is set to TRUE, all interrupts are recorded in the INT_PENDING register, and serviced when the emulator re-enters emulation.

Dynamic Tracing

The trace buffer can be accessed in two ways, dynamically during emulation and statically after emulation halts. While dynamically tracing, any form of the PRINT command can be entered and the specified portion of the trace buffer is displayed. This allows real-time display of processor activity. Displaying the trace buffer during emulation stops collection of trace and some trace information can be lost, but emulation is unaffected.

On-Line Syntax Guide

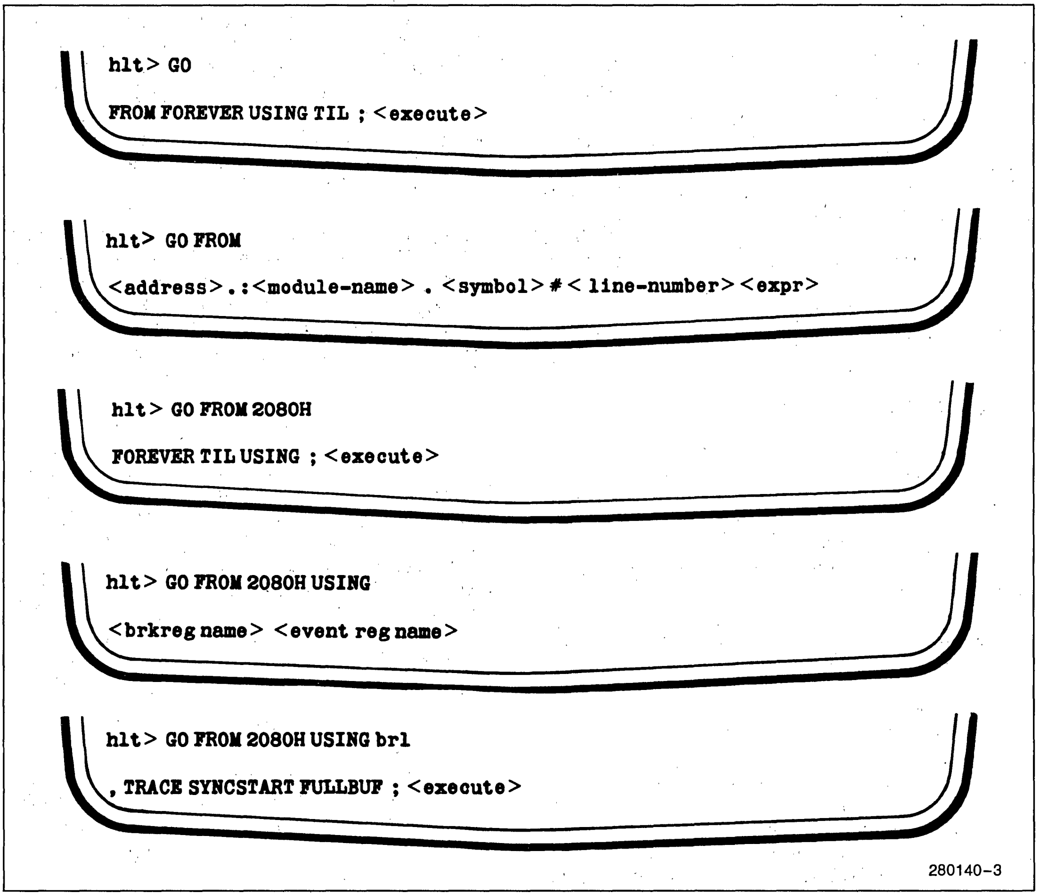
A special syntax guide called the Integrated Command Directory (ICD), at the bottom of the display screen, aids in creating syntactically correct command lines. Figure 3 shows an example of the ICD for the GO command.

HELP

This feature provides assistance with the emulator commands through the host terminal. HELP is available for most of the commands. Figure 4 shows help for one of the commands.

Multi-Synchronous Operation

The VLSiCE-96 emulator can run with other emulators, and lab equipment such as logic analyzers or oscilloscopes. VLSiCE-96 emulators can be daisy-chained together in a network to work simultaneously to test a prototype system. The multi-synchronous operation is facilitated by the optional multi-synchronous accessory.



```
hlt> GO
FROM FOREVER USING TIL ; <execute>

hlt> GO FROM
<address>.:<module-name> . <symbol> # < line-number> <expr>

hlt> GO FROM 2080H
FOREVER TIL USING ; <execute>

hlt> GO FROM 2080H USING
<brkreg name> <event reg name>

hlt> GO FROM 2080H USING brl
, TRACE SYNCSTART FULLBUF ; <execute>
```

280140-3

Figure 3. The Integrated Command Directory for the GO Command

DESIGN CONSIDERATIONS

There are design considerations to be aware of before designing with the VLSiCE-96 emulator.

Electrical Considerations

The user pin timings, thresholds, and loadings are identical to the 8096 component except the RESET and CLKOUT pins have an additional loading of 1 μA and 10 pF. The Non-Maskable Interrupt (NMI) is not supported.

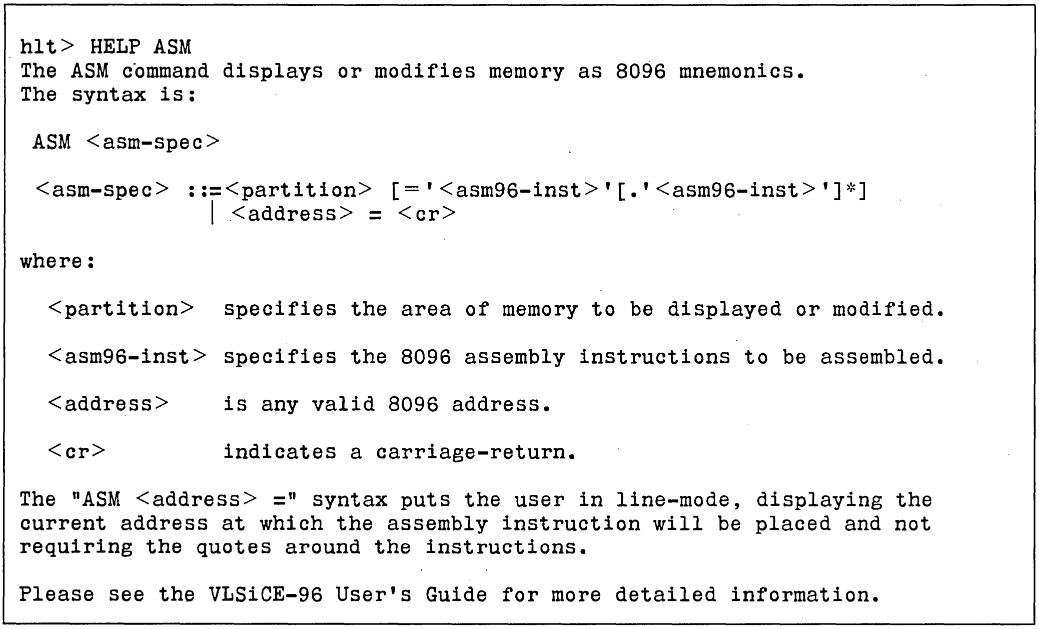
	Min.	Max.
Clock Frequency	6 MHz	10 MHz
V_{CC}	Emulator does not require system power to operate.	
I_{CC}		0 mA

Mechanical Considerations

The user plug is on the end of a three foot (1m) flexible cable. Adequate spacing must be provided on the target system to allow the emulation processor board and user plug to be inserted into the target system.

The height of the user plug should be considered for multiple board system prototypes that need to be debugged and tested. Be sure that the space between the boards is greater than 1½″ (4 cm) to allow for the user plug.

Figure 5 shows the user plug dimensions. The user plug comprises the emulator processor board and the 68-pin or 48-pin adaptor. In the figure, please note the location of pin 1 on each adaptor.



```
hlt> HELP ASM
The ASM command displays or modifies memory as 8096 mnemonics.
The syntax is:

 ASM <asm-spec>

 <asm-spec> ::=<partition> [='<asm96-inst>'[.'<asm96-inst>']*]
             | <address> = <cr>

where:

  <partition>  specifies the area of memory to be displayed or modified.

  <asm96-inst> specifies the 8096 assembly instructions to be assembled.

  <address>    is any valid 8096 address.

  <cr>         indicates a carriage-return.

The "ASM <address> =" syntax puts the user in line-mode, displaying the
current address at which the assembly instruction will be placed and not
requiring the quotes around the instructions.

Please see the VLSiCE-96 User's Guide for more detailed information.
```

Figure 4. HELP Screen

Other Considerations

- The non-maskable interrupt (NMI) is not supported.
- The counters for the pulse width modulator (PWM) and hardware Timer1 can be out of sync if either are disabled during interrogation. Synchronize them by resetting the emulator.
- The Zero flag is always cleared in the SUBC instruction. Therefore, the relational operators <= and > for LONG variables in C96 V1.0 and LONGINT variables in PL/M-96 V1.1, work incorrectly. These languages have been tailored for the 8X9X-90 microcontroller which either sets or resets the Zero flag in the SUBC instruction.

If there is a memory-resident program that is permanent on the PC, use of the DOS shell escape may corrupt the VLSiCE-986 software. To insure reliability, do not use the system escape on host systems that have permanent memory-resident programs.

The VLSiCE-96 emulator has some properties that are inherent in the 8X9XBH component. These are:

- Neither the source nor the destination address of the Multiply or Divide instructions can be a writable special function register.
- The special function registers, except R0, may not be used as base or index registers for indexed or indirect instructions.
- Several of the special function registers can only be accessed as words, while others only as bytes. These restrictions are listed in the 8096 User's Manual.
- The sticky flag is not affected when a skip by 0 is executed.
- To emulate the 8X9X-90 microcontroller, memory location 2018H in both target system emulator mapped memory should be 0FFH.
- The JBS and JBC instructions cannot be used directly on Port 2.1.

SPECIFICATIONS

Host Requirements

An IBM PC XT or PC AT with 512 Kbytes RAM and hard disk. Intel recommends and IBM PC AT with 640 Kbytes of RAM, one floppy drive and one hard disk, running PC-DOS 3.1 or later.

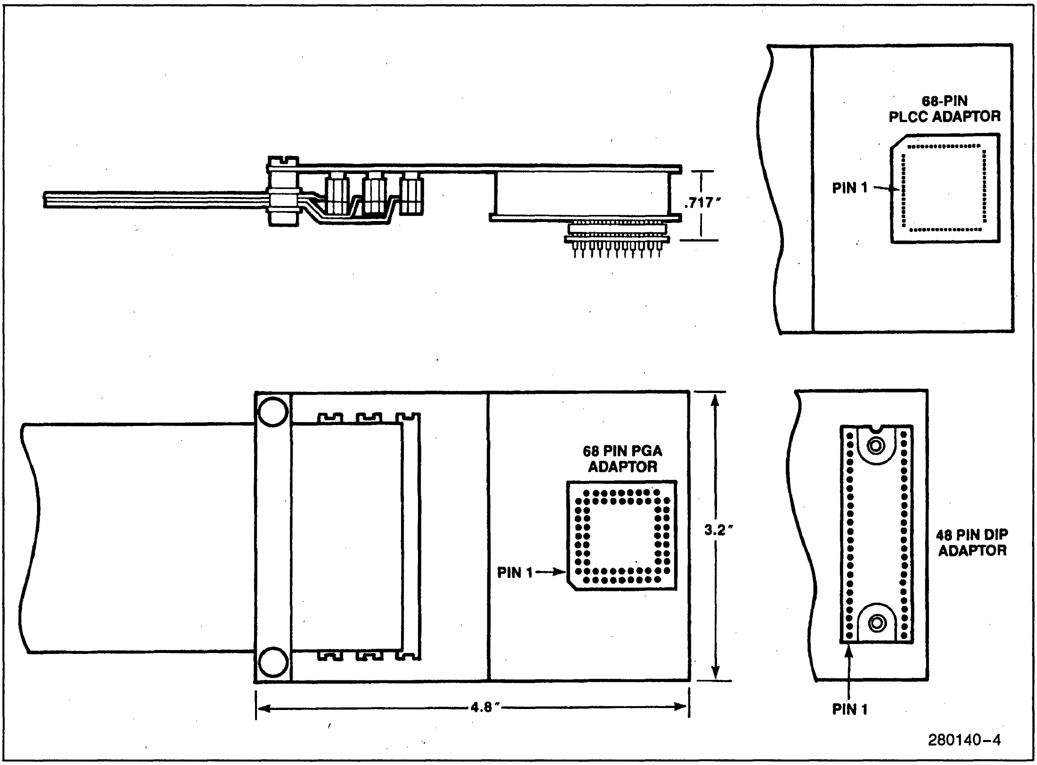


Figure 5. Dimensions for the Emulator Processor Board and Adaptors

An Intellec® Microcomputer Development System, Series III or Series IV, running under ISIS or iNDX, with at least 512 Kbytes of application memory resident, with dual floppy or one hard disk and one floppy drive required.

VLSiCE-96 Software Package

VLSiCE-96 emulator software

VLSiCE-96 confidence tests

VLSiCE-96 tutorial software

System Performance

Mappable zero wait-state (up to 10 MHz).	Min 0 Kbytes Max 64 Kbytes	Mappable to user memory or ICE memory in 1K blocks on 1K boundaries.
Trace Buffer	4 Kbytes × 48 bits	
Virtual Symbol Table	A maximum of 61 Kbytes of host memory space is available for the virtual symbol table (VST). The rest of the VST resides on disk and is paged in and out of host memory as needed.	

Physical Characteristics

Controller Pod

Width:	8¼" (21 cm)
Height:	1½" (4 cm)
Depth:	13½ (34 cm)
Weight:	4 lbs (2 kg)

Power Supply

Width:	7⅝″ (18 cm)
Height:	4″ (10 cm)
Depth:	11″ (28 cm)
Weight:	15 lbs (7 kg)

User Cable

3′ (1 m)

Serial Cable

12′ (4 m)

Electrical Characteristics

Power Supply

100V–120V or 200V–240V (selectable)

50 Hz–60 Hz

2 amps (AC max) @ 120V

1 amp (AC max) @ 240V

Environmental Characteristics

Operating Temperature:	0°C to +40°C (+32°F to +104°F)
Operating Humidity:	Maximum of 85% relative humidity, non-condensing

DOCUMENTATION

VLSiCE-96 In-Circuit Emulator User's Guide, order number 165814

VLSiCE-96 In-Circuit Emulator Pocket Reference, order number 165815

VLSiCE-96 In-Circuit Emulator Installation Supplement, order number 166477

VLSiCE-96 Emulator Tutorial Guide, order number 165816

Debug Editor User's Guide, order 167098

ORDERING INFORMATION

Emulator Hardware and Software

Order Code	Description
V096KITA	VLSiCE-96 power supply and cable, emulation base, user cable, CPA, serial cables for PC AT and PC XT, 68-pin PGA target adaptor, ASM-96, AEDIT text editor. Host, probe, diagnostic and tutorial software is on 5¼″ media for DOS hosts running DOS V3.0 or greater. [Requires software license.]
V096KITD	Same as V096KITAD without ASM-96 and AEDIT text editor.
V096KITAS	VLSiCE-96 power supply and cable, emulation base, user cable, serial cable, CPA, 68-pin PGA target adaptor, ASM-96, and AEDIT text editor. Host, probe, diagnostic and tutorial software is on 8″ single density and 8″ double density media for hosting on an Intel Series III, and 5¼″ media for Series IV [Requires software license.]
V096KITS	Same as V096KITAS without ASM-96 and AEDIT text editor.

Target Adaptor

Order Code	Description
TA096E	Optional 68-pin PLCC Adaptor board
TA096B	Optional 48-pin DIP Target Adaptor board.

Multi-Synchronous Accessory

Order Code	Description
MSA96	Optional Multi-Synchronous Accessory for multi-ICE capability.

Software Only

Order Code	Description
SA096D	Software for host, probe, diagnostic and tutorial on 5¼″ media for use with the PC AT and PC XT under PC-DOS V3.0 or greater. [Requires software license.]
SA096SD	Software for host, probe, diagnostic and tutorial on 5¼″ media for use with the PC AT and PC XT under PC-DOS V3.0 or greater. [Requires software license.]
SA096S	Software for host, probe, diagnostic and tutorial on 8″ single density and 8″ double-density media for use with a Series III, and 5¼″ media for use with a Series IV. [Requires software license.]

Other Useful Intel 8X9X Debug and Development Support Products

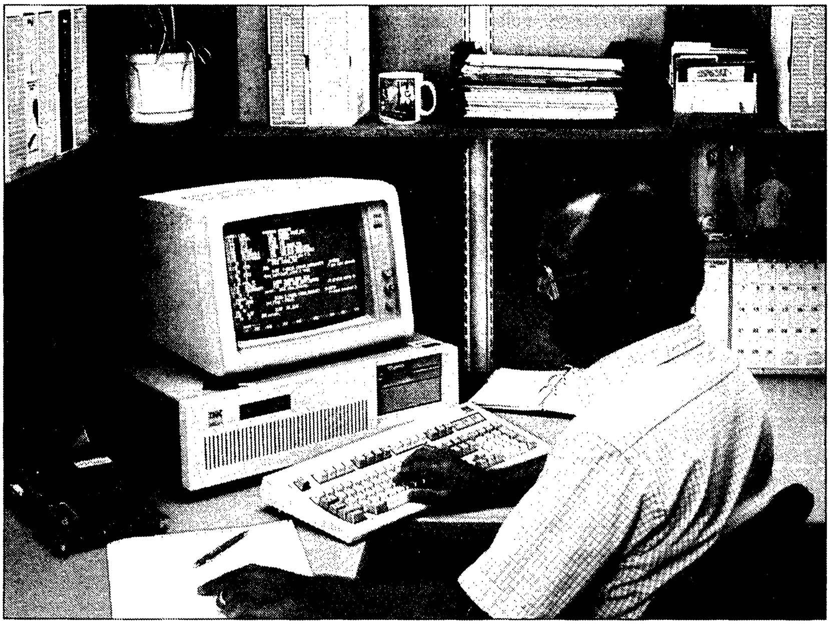
Order Code	Description
I86ASM96	Consists of the ASM 96 macro assembler that translates symbolic assembly language mnemonics into relocatable object code, and the RL96 linker and relocator program that links modules generated by ASM 96 and PL/M 96 and locates the linked object modules to absolute memory locations. System requirements and Intellec System running iNDX.
I86PLM96	Consists of the PL/M 96 compiler that provides high level programming language support, the LIB 96 utility that creates and maintains libraries of software object modules, the FPAL96 floating point arithmetic library, and the RL96 linker and relocator program that links modules generated by ASM 96 and PL/M 96 and locates the linked object modules to absolute memory locations. System requirements and Intellec System running iNDX.
D86ASM96NL	ASM/R&L 96 for PC-DOS. It contains a macro assembler, a linker/locator utility, a floating point utility and a librarian. System requirements are an IBM PC AT or PC XT with 512 Kbytes of RAM and PC-DOS 3.0 or greater.
D86PLM96NL	PL/M 96 and R&L for PC-DOS. It contains a compiler, a linker/locator utility, a floating point utility and a librarian. System requirements are an IBM PC AT or PC XT with 512 Kbytes of RAM and PC-DOS 3.0 or greater.
D86C96NL	C96 and R&L for PC-DOS. Contains a compiler linker/locator utility, and all standard C libraries including STDIO. System requirements are an IBM PC AT or PC XT with 512 Kbytes of RAM and PC-DOS 3.0 or greater.
pSBE96SKIT	iSBE-96 single board emulator for use with the Series III/IV development systems. The kit contains: iSBE-96 single board emulator iSBE-96 Series III/IV upgrade kit (cables and software needed to run on Intel Hosts).
pSBE96DKIT	iSBE-96 single board emulator for use with the IBM PC AT and PC XT computer systems. The kit contains: iSBE-96 single board emulator 8096 software support package for PC-DOS. iSBE-96 DOS upgrade kit (cables and software needed to run on the IBM PC AT or PC XT).

Running the iSBE-96 emulator on the Series II and iPDS system requires software from:

U.S. Software Corporation
5470 N.W. Innisbrook
Portland, OR 97229
Phone: 503-645-5043
International Telex: 4993875

PRELIMINARY

REAL-TIME TRANSPARENT 80C196 IN-CIRCUIT EMULATOR



REAL-TIME TRANSPARENT 80C196 IN-CIRCUIT EMULATOR

The ICE-196PC in-circuit emulator delivers real-time high-level debugging capabilities for developing, integrating and testing 80C196-based designs. Operating at the full speed of the 80C196 microcontroller, the ICE-196PC provides precise I/O pin timings and functionality. The ICE-196PC also allows you to develop code before prototype hardware is available. The ICE-196PC in-circuit emulator represents a low-cost development environment for designing real-time microcontroller-based applications with minimal investment in time and resources.

ICE™-196PC IN CIRCUIT EMULATOR FEATURES

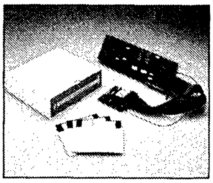
- Real-Time Emulation of the 80C196 Microcontroller
- 64K Bytes of Mappable Memory
- 2K-entry Trace Buffer
- 3 Breakpoints or 1 Range Break
- Symbolic Support and Source Code Display
- Standalone Operation
- Versatile and Powerful Host Software
- Hosted On IBM PC XT, AT* or Compatibles With DOS 3.0 or Later

REAL-TIME EMULATION

The ICE-196PC provides real-time emulation with the precise input/output pin timings and functions across the full operating frequencies of the 80C196 microcontroller. The ICE-196PC connects to the intended 80C196 microcontroller socket via a 16″ flex cable, which terminates in a 68-pin PLCC probe.

MAPPABLE MEMORY

The ICE-196PC has 64K bytes (65,536) of zero wait-state memory that can be enabled or mapped as read-only, write-only or read/write in 4K byte increments to simulate the internal (EP)ROM of the 80C196 or external program memory.



intel

*PC XT, AT are trademarks of IBM.
Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.

AUGUST, 1987
Order Number: 280727-001

TRACE BUFFER

The ICE-196PC contains a 2K (2048) entry trace buffer for keeping a history of actual instruction execution. The trace buffer can be displayed as disassembled instructions or, optionally, disassembled instructions and the original C-96 and PL/M-96 source code.

BREAK SPECIFICATION

Three execution address breakpoints or one range of addresses can be active at any time. The ICE-196PC allows any number of breakpoints to be defined and activated when needed.

SYMBOLIC SUPPORT AND SOURCE CODE DISPLAY

Full ASM-96, PL/M-96 and C-96 language symbolics, including variable typing and scope, are supported by the ICE-196PC memory accesses, trace buffer display, breakpoint specification, and assembler/disassembler. Additionally, C-96 and PL/M-96 source code can be displayed to make development and debug easier.

STANDALONE OPERATION

Product software can be developed prior to hardware availability with the optional Crystal Power Accessory (CPA) and the ICE-196PC mappable memory. The CPA also provides diagnostic testing to assure full functionality of the ICE-196PC.

VERSATILE AND POWERFUL HOST SOFTWARE

The ICE-196PC comes equipped with an on-line help facility, a dynamic command entry and syntax guide, built-in editor, assembler and disassembler, and the ability to customize the command set via literal definitions and debug procedures.

HOSTING

The ICE-196PC is hosted on the IBM PC XT, AT or compatibles with PC-DOS 3.0 or later.

SPECIFICATIONS

HOST REQUIREMENTS

IBM PC XT, AT (or compatible)
512K bytes RAM, Hard Disk
PC-DOS 3.0 or Later
One Unused Peripheral Slot
DC Current 2.5A

TARGET INTERFACE BOARD

Length	2.0" (5.1cm)
Height	1.2" (3.0cm)
Width	2.3" (5.8cm)

USER CABLE

Length	15.6" (39.6cm)

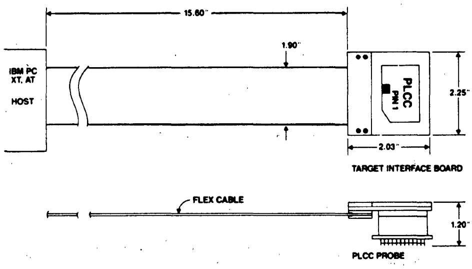
PROBE ELECTRICAL

80C196* plus per pin	50pf loading 5ns propagation delay
Icc (from target system)	50mA @ 12 MHz
Operating Frequency	3.5 to 12 MHz, 12 MHz only with CPA

ENVIRONMENTAL CHARACTERISTICS

Operating Temperature	10°C to 40°C 37.5°F to 104°F
Operating Humidity	Maximum 55% Relative Humidity, non-condensing

*This emulator supports the initial 80C196 microcontroller. The HOLD/HOLDA feature will be supported by a future product.



ORDERING INFORMATION

Order Code	Description
ICE-196PC	Emulation Board, user cable, target interface board (PLCC), host, diagnostic, and tutorial software on 5¼" DOS diskette, and Crystal Power Accessory with power cable
ICE-196PCB	Same as above except does not include Crystal Power Accessory
CPA196	Crystal Power Accessory and power cable only
D86C96NL	C-96 Compiler*
D86PLM96NL	PL/M-96 Compiler*
D86ASM96NL	ASM-96 Assembler*

*Includes: Relocator/Linker, Object-to-hex Converter, Floating Point Arithmetic Library, Librarian

For more information or the number of your nearest sales office call 800-548-4725 (good in the U.S. and Canada).

UNITED STATES, Intel Corporation
3065 Bowers Ave., Santa Clara, CA 95051
Tel: (408) 987-8080

MSC®-96 INDEX

80C196 INDEX

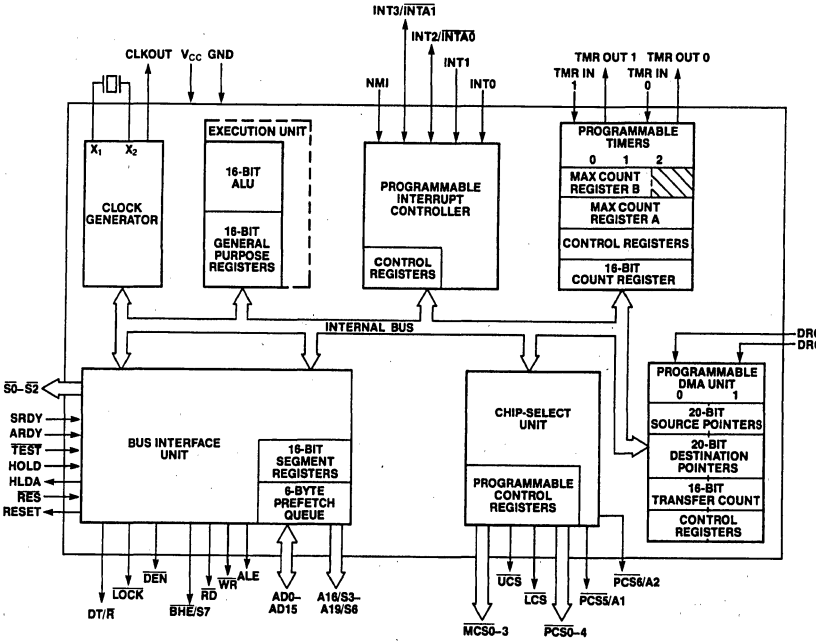
80186 Data Sheets, Application Notes and Development Support Tools

22

80186
HIGH INTEGRATION 16-BIT MICROPROCESSOR

- Integrated Feature Set
  - — Enhanced 8086-2 CPU
  - — Clock Generator
  - — 2 Independent DMA Channels
  - — Programmable Interrupt Controller
  - — 3 Programmable 16-bit Timers
  - — Programmable Memory and Peripheral Chip-Select Logic
  - — Programmable Wait State Generator
  - — Local Bus Controller
- Available in 10 MHz (80186-10) and 8 MHz (80186) Versions
- High-Performance Processor
  - — At 8 MHz provides 2 times the Performance of the Standard 8086
  - — 4 MByte/Sec Bus Bandwidth Interface @ 8 MHz
  - — 5 MByte/Sec Bus Bandwidth Interface @ 10 MHz
- Direct Addressing Capability to 1 MByte of Memory and 64 KByte I/O
- Completely Object Code Compatible with All Existing 8086, 8088 Software
  - — 10 New Instruction Types
- Complete System Development Support
  - — Development Software; Assembler, PL/M, Pascal, Fortan, and System Utilities
  - — In-Circuit-Emulator (I^{2}ICE™-186)
- High Performance Numerical Coprocessing Capability Through 8087 Interface
- Available in 68 Pin:
  - — Plastic Leaded Chip Carrier (PLCC)
  - — Ceramic Pin Grid Array (PGA)
  - — Ceramic Leadless Chip Carrier (LCC)

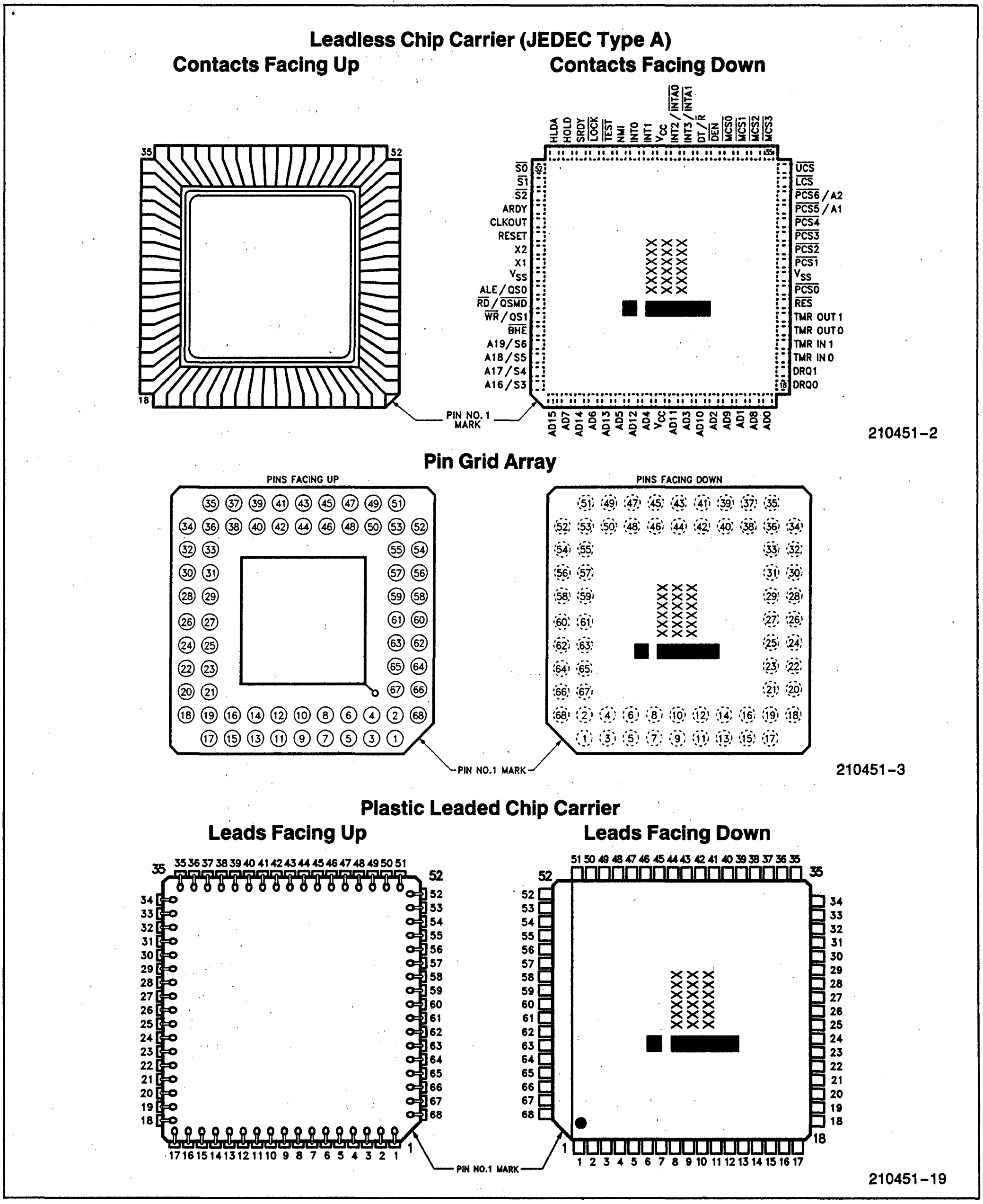
  (See Packaging Outlines and Dimensions, Order #231369)
- Available in EXPRESS
  - — Standard Temperature with Burn-In
  - — Extended Temperature Range (−40°C to +85°C)



210451-1

Figure 1. 80186 Block Diagram

The Intel 80186 is a highly integrated 16-bit microprocessor. The 80186 effectively combines 15–20 of the most common 8086 system components onto one. The 80186 provides two times greater throughput than the standard 5 MHz 8086. The 80186 is upward compatible with 8086 and 8088 software and adds 10 new instruction types to the existing set.


Leadless Chip Carrier (JEDEC Type A)

Contacts Facing Up

Contacts Facing Down

Pin Grid Array

Plastic Leaded Chip Carrier

Leads Facing Up

Leads Facing Down

Figure 2. 80186 Pinout Diagrams

Table 1. 80186 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}, V_{CC}	9, 43	I	**System Power:** +5 volt power supply.
V_{SS}, V_{SS}	26, 60	I	System Ground.
RESET	57	O	Reset Output indicates that the 80186 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{\text{RES}}$ signal.
X1, X2	59, 58	I	Crystal Inputs X1 and X2 provide external connections for a fundamental mode parallel resonant crystal for the internal oscillator. Instead of using a crystal, an external clock may be applied to X1 while minimizing stray capacitance on X2. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	Clock Output provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT. CLKOUT has sufficient MOS drive capabilities for the 8087 Numeric Processor Extension.
$\overline{\text{RES}}$	24	I	System Reset causes the 80186 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80186 clock. The 80186 begins fetching instructions approximately 7 clock cycles after $\overline{\text{RES}}$ is returned HIGH. For proper initialization, V_{CC} must be within specifications and the clock signal must be stable for more than 4 clocks with $\overline{\text{RES}}$ held LOW. $\overline{\text{RES}}$ is internally synchronized. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{\text{RES}}$ generation via an RC network. When $\overline{\text{RES}}$ occurs, the 80186 will drive the status lines to an inactive level for one clock, and then float them.
$\overline{\text{TEST}}$	47	I	$\overline{\text{TEST}}$ is examined by the WAIT instruction. If the $\overline{\text{TEST}}$ input is HIGH when "WAIT" execution begins, instruction execution will suspend. $\overline{\text{TEST}}$ will be resampled until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80186 is waiting for $\overline{\text{TEST}}$, interrupts will be serviced. This input is synchronized internally.
TMR IN 0, TMR IN 1	20 21	I I	Timer Inputs are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0, TMR OUT 1	22 23	O O	Timer outputs are used to provide single pulse or continous waveform generation, depending upon the timer mode selected.
DRQ0 DRQ1	18 19	I I	DMA Request is driven HIGH by an external device when it desires that a DMA channel (Channel 0 or 1) perform a transfer. These signals are active HIGH, level-triggered, and internally synchronized.
NMI	46	I	Non-Maskable Interrupt is an edge-triggered input which causes a type 2 interrupt. NMI is not maskable internally. A transition from a LOW to HIGH initiates the interrupt at the next instruction boundary. NMI is latched internally. An NMI duration of one clock or more will guarantee service. This input is internally synchronized.
INT0, INT1 INT2/$\overline{\text{INTA0}}$ INT3/$\overline{\text{INTA1}}$	45, 44 42 41	I I/O I/O	Maskable Interrupt Requests can be requested by activating one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured via software to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured via software to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When slave mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).

Table 1. 80186 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th colspan="3">Name and Function</th></tr>
<tr><td rowspan="4">A19/S6,
A18/S5,
A17/S4,
A16/S3</td><td rowspan="4">65
66
67
68</td><td rowspan="4">O
O
O
O</td><td colspan="3">Address Bus Outputs (16–19) and Bus Cycle Status (3–6) reflect the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below:</td></tr>
<tr><td></td><td><b>Low</b></td><td><b>High</b></td></tr>
<tr><td>S6</td><td>Processor Cycle</td><td>DMA Cycle</td></tr>
<tr><td colspan="3">S3, S4, and S5 are defined as LOW during T_2–T_4. The status pins float during HOLD/HLDA.</td></tr>
<tr><td>AD15–AD0</td><td>10–17,
1–8</td><td>I/O</td><td colspan="3">Address/Data Bus (0–15) signals constitute the time multiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH. A_0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D_7 through D_0. It is LOW during T_1 when a byte is to be transferred onto the lower portion of the bus in memory or I/O operations.</td></tr>
<tr><td rowspan="7">$\overline{BHE}$/S7</td><td rowspan="7">64</td><td rowspan="7">O</td><td colspan="3">During T_1 the Bus High Enable signal should be used to determine if data is to be enabled onto the most significant half of the data bus; pins D_{15}–D_8. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the higher half of the bus. The S_7 status information is available during T_2, T_3, and T_4. S_7 is logically equivalent to $\overline{BHE}$. $\overline{BHE}$/S7 floats during HOLD.</td></tr>
<tr><td colspan="3"><b>$\overline{BHE}$ and A0 Encodings</b></td></tr>
<tr><td><b>$\overline{BHE}$ Value</b></td><td><b>A0 Value</b></td><td><b>Function</b></td></tr>
<tr><td>0</td><td>0</td><td>Word Transfer</td></tr>
<tr><td>0</td><td>1</td><td>Byte Transfer on upper half of data bus (D15–D8)</td></tr>
<tr><td>1</td><td>0</td><td>Byte Transfer on lower half of data bus (D_7–D_0)</td></tr>
<tr><td>1</td><td>1</td><td>Reserved</td></tr>
<tr><td>ALE/QS0</td><td>61</td><td>O</td><td colspan="3">Address Latch Enable/Queue Status 0 is provided by the 80186 to latch the address into the 8282/8283 address latches. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the standard 8086. The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 8086. Note that ALE is never floated.</td></tr>
<tr><td rowspan="6">$\overline{WR}$/QS1</td><td rowspan="6">63</td><td rowspan="6">O</td><td colspan="3">Write Strobe/Queue Status 1 indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{WR}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during "HOLD." It is driven HIGH for one clock during Reset, and then floated. When the 80186 is in queue status mode, the ALE/QS0 and $\overline{WR}$/QS1 pins provide information about processor/instruction queue interaction.</td></tr>
<tr><td><b>QS1</b></td><td><b>QS0</b></td><td><b>Queue Operation</b></td></tr>
<tr><td>0</td><td>0</td><td>No queue operation</td></tr>
<tr><td>0</td><td>1</td><td>First opcode byte fetched from the queue</td></tr>
<tr><td>1</td><td>1</td><td>Subsequent byte fetched from the queue</td></tr>
<tr><td>1</td><td>0</td><td>Empty the queue</td></tr>
</table>

Table 1. 80186 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$\overline{RD}/\overline{QSMD}$</td><td>62</td><td>O</td><td>Read Strobe indicates that the 80186 is performing a memory or I/O read cycle. $\overline{RD}$ is active LOW for T_2, T_3, and T_W of any read cycle. It is guaranteed not to go LOW in T_2 until after the Address Bus is floated. $\overline{RD}$ is active LOW, and floats during "HOLD". $\overline{RD}$ is driven HIGH for one clock during Reset, and then the output driver is floated. A weak internal pull-up mechanism of the $\overline{RD}$ line holds it HIGH when the line is not driven. During RESET the pin is sampled to determine whether the 80186 should provide ALE, $\overline{WR}$ and $\overline{RD}$, or if the Queue-Status should be provided. $\overline{RD}$ should be connected to GND to provide Queue-Status data.</td></tr>
<tr><td>ARDY</td><td>55</td><td>I</td><td>Asynchronous Ready informs the 80186 that the addressed memory space or I/O device will complete a data transfer. The ARDY input pin will accept an asynchronous input, and is active HIGH. Only the rising edge is internally synchronized by the 80186. This means that the falling edge of ARDY must be synchronized to the 80186 clock. If connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active to terminate a bus cycle. If unused, this line should be tied LOW to yield control to the SRDY pin.</td></tr>
<tr><td>SRDY</td><td>49</td><td>I</td><td>Synchronous Ready must be synchronized externally to the 80186. The use of SRDY provides a relaxed system-timing specification on the Ready input. This is accomplished by eliminating the one-half clock cycle which is required for internally resolving the signal level when using the ARDY input. This line is active HIGH. If this line is connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active before a bus cycle is terminated. If unused, this line should be tied LOW to yield control to the ARDY pin.</td></tr>
<tr><td>$\overline{LOCK}$</td><td>48</td><td>O</td><td>$\overline{LOCK}$ output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of the instruction following the LOCK prefix. No prefetches will occur while $\overline{LOCK}$ is asserted. When executing more than one LOCK instruction, always make sure there are 6 bytes of code between the end of the first LOCK instruction and the start of the second LOCK instruction. $\overline{LOCK}$ is active LOW, is driven HIGH for one clock during RESET, and then floated.</td></tr>
<tr><td>$\overline{S0}$, $\overline{S1}$, $\overline{S2}$</td><td>52–54</td><td>O</td><td>Bus cycle status $\overline{S0}$–$\overline{S2}$ are encoded to provide bus-transaction information:
<table>
<tr><th colspan="4">80186 Bus Cycle Status Information</th></tr>
<tr><th>$\overline{S2}$</th><th>$\overline{S1}$</th><th>$\overline{S0}$</th><th>Bus Cycle Initiated</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Interrupt Acknowledge</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Read I/O</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Write I/O</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Halt</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Instruction Fetch</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Read Data from Memory</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Write Data to Memory</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Passive (no bus cycle)</td></tr>
</table>
The status pins float during HOLD/HLDA.
$\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator.
The status lines are driven HIGH for one clock during Reset, and then floated until a bus cycle begins.</td></tr>
</table>

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
HOLD (input) HLDA (output)	50 51	I O	HOLD indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80186 clock. The 80186 will issue a HLDA (HIGH) in response to a HOLD request at the end of T_4 or T_i. Simultaneous with the issuance of HLDA, the 80186 will float the local bus and control lines. After HOLD is detected as being LOW, the 80186 will lower HLDA. When the 80186 needs to run another bus cycle, it will again drive the local bus and control lines.
$\overline{UCS}$	34	O	Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{UCS}$ is software programmable.
$\overline{LCS}$	33	O	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{LCS}$ is software programmable.
$\overline{MCS0}$–3	38, 37, 36, 35	O	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{MCS0}$–3 are software programmable.
$\overline{PCS0}$ $\overline{PCS1}$–4	25 27, 28, 29, 30	O O	Peripheral Chip Select signals 0–4 are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{PCS0}$–4 are software programmable.
$\overline{PCS5}$/A1	31	O	Peripheral Chip Select 5 or Latched A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{PCS5}$ is software programmable. When programmed to provide latched. A1, rather than $\overline{PCS5}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{PCS6}$/A2	32	O	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{PCS6}$ is software programmable. When programmed to provide latched A2, rather than $\overline{PCS6}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{R}$	40	O	Data Transmit/Receive controls the direction of data flow through the external 8286/8287 data bus transceiver. When LOW, data is transferred to the 80186. When HIGH the 80186 places write data on the data bus.
$\overline{DEN}$	39	O	Data Enable is provided as an 8286/8287 data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80186. This architecture is common to the 8086, 8088, and 80286 microprocessor families as well. The 80186 is a very high integration 16-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip while providing twice the performance of the standard 8086. The 80186 is object code compatible with the 8086/8088 microprocessors and adds 10 new instruction types to the existing 8086/8088 instruction set.

80186 BASE ARCHITECTURE

The 8086, 8088, 80186, and 80286 family all contain the same basic set of registers, instructions, and addressing modes. The 80186 processor is upward compatible with the 8086, 8088, and 80286 CPUs.

Register Set

The 80186 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

General Registers

Eight 16-bit general purpose registers may be used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

Segment Registers

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

Status and Control Registers

Two 16-bit special purpose registers record or alter certain aspects of the 80186 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

Status Word Description

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80186 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

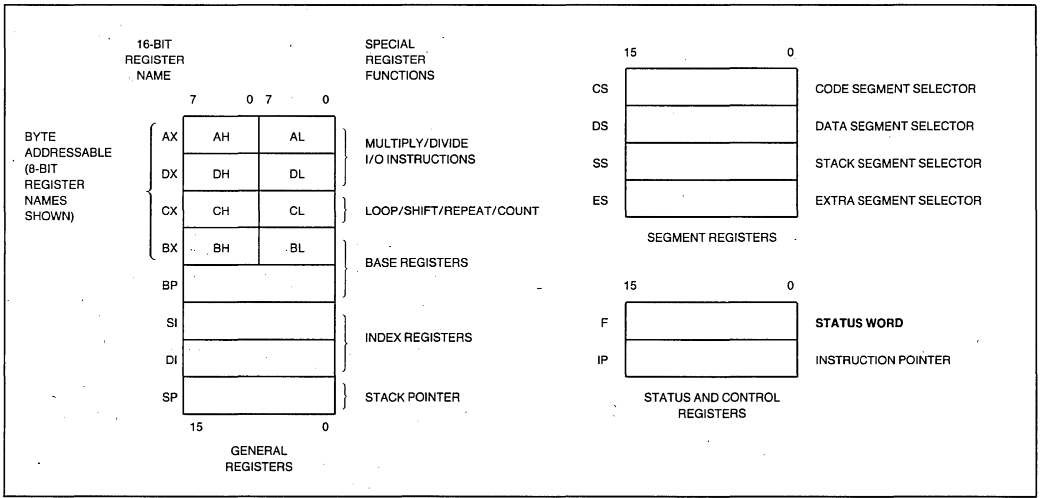


Figure 3a. 80186 Register Set

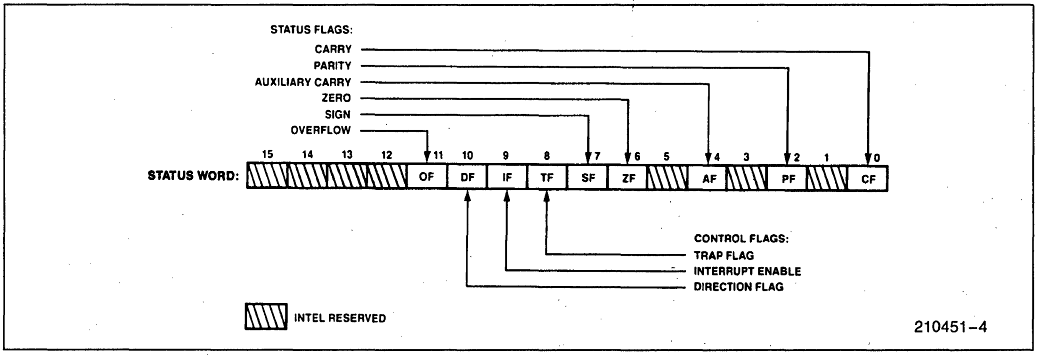


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80186 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment register (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. 80186 Instruction Set

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign

JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4. 80186 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

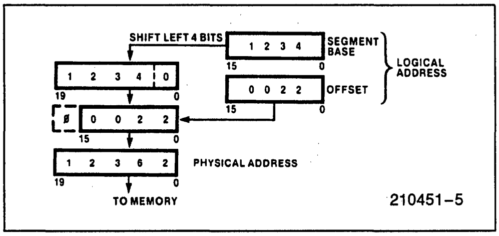


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

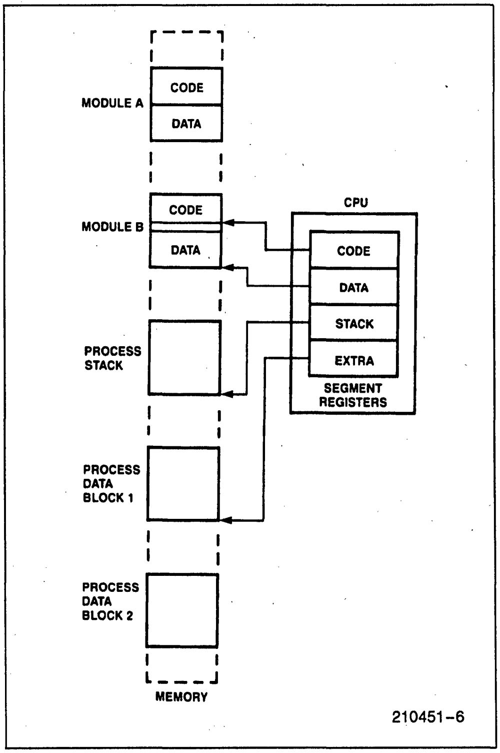


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80186 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an Index register.
- *Based indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80186 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using an 8087 Numeric Data Coprocessor with the 80186.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using an 8087 Numeric Data Coprocessor with the 80186.)

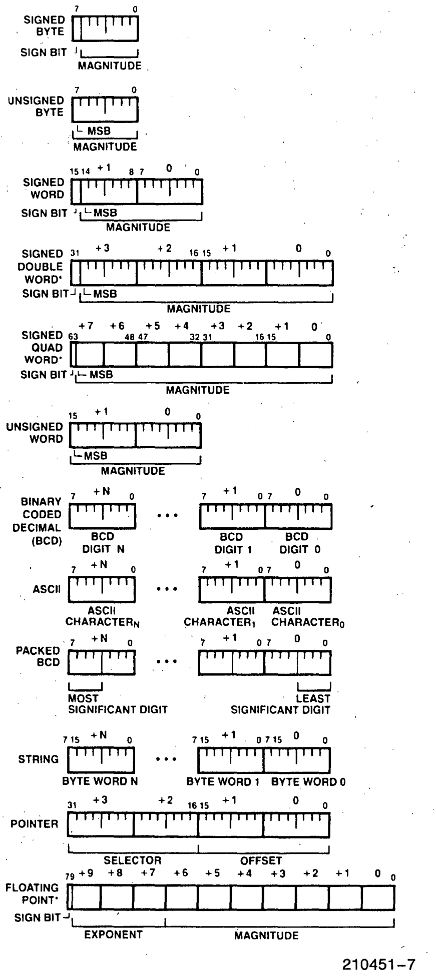
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80186.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



NOTE:
*Supported by using an 8087 Numeric Data Coprocessor with the 80186.

Figure 7. 80186 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80186 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80186 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80186 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INTO, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80186 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80186 Interrupt Vectors

Interrupt Name	Vector Type	Default Priority	Related Instructions
Divide Error Exception	0	*1	DIV, IDIV
Single Step Interrupt	1	12**	All
NMI	2	1	All
Breakpoint Interrupt	3	*1	INT
INTO Detected Overflow Exception	4	*1	INTO
Array Bounds Exception	5	*1	BOUND
Unused-Opcode Exception	6	*1	Undefined Opcodes
ESC Opcode Exception	7	*1***	ESC Opcodes
Timer 0 Interrupt	8	2A****	
Timer 1 Interrupt	18	2B****	
Timer 2 Interrupt	19	2C****	
Reserved	9	3	
DMA 0 Interrupt	10	4	
DMA 1 Interrupt	11	5	
INT0 Interrupt	12	6	
INT1 Interrupt	13	7	
INT2 Interrupt	14	8	
INT3 Interrupt	15	9	

NOTES:

*1. These are generated as the result of an instruction execution.

**2. This is handled as in the 8086.

****3.All three timers constitute one source of request to the interrupt controller. The Timer interrupts all have the same default priority level with respect to all other interrupt sources. However, they have a defined priority ordering amongst themselves. (Priority 2A is higher priority than 2B.) Each Timer interrupt has a separate vector type number.

4. Default priorities for the interrupt sources are used only if the user does not program each source into a unique priority level.

***5. An escape opcode will cause a trap only if the proper bit is set in the peripheral control block relocation register.

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INTO DETECTED OVERFLOW EXCEPTION (TYPE4)

Generated during an INTO instruction if the OF bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H–DFH). This exception will only be generated if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80186 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80186 integrated DMA controller and the integrated timer unit. The vector types for these interrupts is shown in Table 4. Software enables these inputs by setting the interrupt flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80186 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80186 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{\text{RES}}$ input pin LOW. $\overline{\text{RES}}$ forces the 80186 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{\text{RES}}$ is active. After $\overline{\text{RES}}$ becomes inactive and an internal processing interval elapses, the 80186 begins execution with the instruction at physical location FFFF0(H). $\overline{\text{RES}}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80186 Initial Register State after RESET

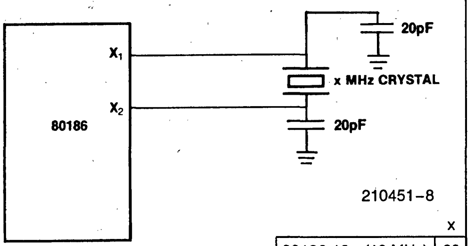
Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

80186 CLOCK GENERATOR

The 80186 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The oscillator circuit of the 80186 is designed to be used with a parallel resonant fundamental mode crystal. This is used as the time base for the 80186. The crystal frequency selected will be double the CPU clock frequency. Use of an LC or RC circuit is not recommended with this oscillator. If an external oscillator is used, it can be connected directly to input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the 80186. The recommended crystal configuration is shown in Figure 8.



	x
80186-10 (10 MHz)	20
80186 (8 MHz)	16

Figure 8. Recommended 80186 Crystal Configuration

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_1 (Load Capacitance):	20 pf ± 2 pf
Drive Level:	1 mW max

Clock Generator

The 80186 clock generator provides the 50% duty cycle processor clock for the 80186. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80186. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80186 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used. Full synchronization is performed only on the rising edge of ARDY, i.e., the falling edge of ARDY must be synchronized to the CLKOUT signal if it will occur during T_2, T_3, or T_W. High-to-LOW transitions of ARDY must be performed synchronously to the CPU clock.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80186, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The 80186 provides both a $\overline{\text{RES}}$ input pin and a synchronized RESET pin for use with other system components. The $\overline{\text{RES}}$ input pin on the 80186 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{\text{RES}}$ input of at least six clocks. RESET may be delayed up to two and one-half clocks behind $\overline{\text{RES}}$.

Multiple 80186 processors may be synchronized through the $\overline{\text{RES}}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{\text{RES}}$ must satisfy a 25 ns setup time before the falling edge of the 80186 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 25 ns setup time before the rising edge of the CLKOUT signal of all the processors.

LOCAL BUS CONTROLLER

The 80186 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides control lines that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80186 provides ALE, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ bus control signals. The $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals are used to strobe data from memory to the 80186 or to strobe data from the 80186 to memory. The ALE line provides a strobe to address latches for the multiplexed address/data bus. The 80186 local bus controller does not provide a memory/$\overline{\text{IO}}$ signal. If this is required, the user will have to use the $\overline{\text{S2}}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80186 generates two control signals to be connected to 8286/8287 transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{\text{R}}$ and $\overline{\text{DEN}}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{\text{DEN}}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{\text{R}}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80186 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80186 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. This requires external circuitry to arbitrate which external device will gain control of the bus from the 80186 when there is more than one alternate local bus master. When the 80186 relinquishes control of the local bus, it floats $\overline{\text{DEN}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{S0}}$–$\overline{\text{S2}}$, $\overline{\text{LOCK}}$, AD0–AD15, A16–A19, $\overline{\text{BHE}}$, and DT/$\overline{\text{R}}$ to allow another master to drive these lines directly.

The 80186 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is the highest-priority activity request which the processor may receive: higher than instruction fetching or internal DMA cycles. However, if a DMA cycle is in progress, the 80186 will complete the transfer before relinquishing the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency time can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd ad-

dress to an odd address. This is a total of 16 clocks or more, if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

Local Bus Controller and Reset

Upon receipt of a RESET pulse from the $\overline{RES}$ input, the local bus controller will perform the following action:

- Drive $\overline{DEN}$, $\overline{RD}$, and $\overline{WR}$ HIGH for one clock cycle, then float.

NOTE:
$\overline{RD}$ is also provided with an internal pull-up device to prevent the processor from inadvertently entering Queue Status mode during reset.

- Drive $\overline{S0}$–$\overline{S2}$ to the passive state (all HIGH) and then float.
- Drive $\overline{LOCK}$ HIGH and then float.
- Float AD0–15, A16–19, $\overline{BHE}$, DT/$\overline{R}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80186 integrated peripherals are controlled via 16-bit registers contained within an internal 256-byte control block. This control block may be mapped into either memory or I/O space. Internal logic will recognize the address and respond to the bus cycle. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{15-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80186 CPU at any time. The location of any register contained within the 256-byte control block is determined by the current base address of the control block.

The control block base address is programmed via a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. The control block is effectively an internal chip select range and must abide by all the rules concerning chip selects (the chip select circuitry is discussed later in this data sheet). Any access to the 256 bytes of the control block activates an internal chip select. Other chip selects may overlap the control block only if they are programmed to zero wait states and ignore external ready. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space, whereas if the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into slave mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register is set to 20FFH. This causes the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

The integrated 80186 peripherals operate semi-autonomously from the CPU. Access to them for the most part is via software read/write of the control block. Most of these registers can be both read and written. A few dedicated lines, such as interrupts and DMA request provide real-time communication between the CPU and peripherals as in a more conventional system utilizing discrete peripheral blocks. The overall interaction and function of the peripheral blocks has not substantially changed.

CHIP-SELECT/READY GENERATION LOGIC

The 80186 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also povide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The 80186 provides 6 memory chip select outputs for 3 address areas; upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address

	15	14	13	12	11 10 9 8 7 6 5 4 3 2 1 0
OFFSET: FEH	ET	SLAVE/$\overline{\text{MASTER}}$	X	M/IO	Relocation Address Bits R19–R8

ET = ESC Trap / No ESC Trap (1/0)
M/IO = Register block located in Memory / I/O Space (1/0)
SLAVE/$\overline{\text{MASTER}}$ = Configure interrupt controller for Slave/Master Mode (I/O)

Figure 9. Relocation Register

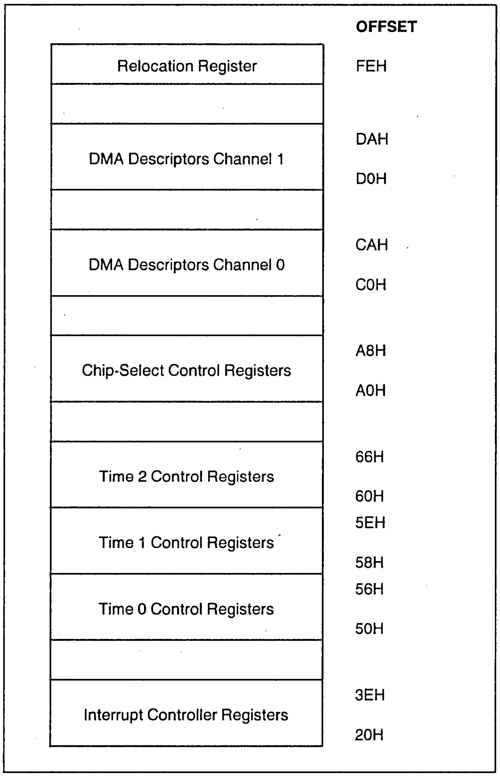


Figure 10. Internal Register Map

of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes, whereas 80186 memory is arranged in words. This means that if, for example, 16 64K x 1 memories are used, the memory block size will be 128K, not 64K.

Upper Memory $\overline{\text{CS}}$

The 80186 provides a chip select, called $\overline{\text{UCS}}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80186 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0 = R1 = R2 = 0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

Any internally generated 20-bit address whose upper 16 bits are greater than or equal to UMCS (with bits 0–5 "0") will cause UCS to be activated. UMCS bits R2–R0 are used to specify READY mode for the area of memory defined by this chip-select register, as explained below.

Lower Memory $\overline{\text{CS}}$

The 80186 provides a chip select for low memory called $\overline{\text{LCS}}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is also defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12). This register is at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{\text{LCS}}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will cause $\overline{\text{LCS}}$ to be active. LMCS register bits R2–R0 are used to specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{\text{CS}}$

The 80186 provides four $\overline{\text{MCS}}$ lines which are active within a user-locatable memory block. This block can be located within the 80186 1M byte memory address space exclusive of the areas defined by $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{\text{MCS}}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. Thus, if the total block size is 32K, each chip select is active for 8K of memory with $\overline{\text{MCS0}}$ being active for the first range and $\overline{\text{MCS3}}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionally as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{\text{MCS}}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{\text{MCS}}$ lines will be active until both the MMCS and MPCS registers are accessed.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A0H	1	1	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 11. UMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A2H	0	0	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 12. LMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A8H	1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

Figure 13. MPCS Register

	15						9						3			0
OFFSET: A6H	U	U	U	U	U	U	U	1	1	1	1	1	1	R2	R1	R0
	A19						A13									

Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all mid-range chip selects. All devices in mid-range memory must use the same number of WAIT states.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{\text{LCS}}$ line was programmed, there would be an internal conflict between the $\overline{\text{LCS}}$ ready generation logic and the $\overline{\text{MCS}}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{\text{UCS}}$ ready generation logic. Since the $\overline{\text{LCS}}$ chip-select line does not become active until programmed, while the $\overline{\text{UCS}}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{\text{LCS}}$ range must not be programmed.

Peripheral Chip Selects

The 80186 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. This base address may be located in either memory or I/O space.

Seven $\overline{\text{CS}}$ lines called $\overline{\text{PCS0}}$–6 are generated by the 80186. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$ can also be programmed to provide latched address bits A1, A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0, A1 pins used for selecting internal registers of 8-bit peripheral chips. This scheme simplifies the hardware interface because the 8-bit registers of peripherals are simply treated as 16-bit registers located on even boundaries in I/O space or memory space where only the lower 8-bits of the register are significant: the upper 8-bits are "don't cares."

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). This register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

	15									6	5		3			0
OFFSET: A4H	U	U	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19									A10						

Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for $\overline{PCS0}$–$\overline{PCS3}$.

Table 10. PCS Address Ranges

PCS Line	Active between Locations
PCS0	PBA —PBA+127
PCS1	PBA+128—PBA+255
PCS2	PBA+256—PBA+383
PCS3	PBA+384—PBA+511
PCS4	PBA+512—PBA+639
PCS5	PBA+640—PBA+767
PCS6	PBA+768—PBA+895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 13). This register is located at offset A8H in the internal control block. Bit 7 is used to select the function of $\overline{PCS5}$ and $\overline{PCS6}$, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space. 0 = Peripherals mapped into I/O space.
EX	0 = 5 $\overline{PCS}$ lines. A1, A2 provided. 1 = 7 $\overline{PCS}$ lines. A1, A2 are not provided.

MPCS bits 0–2 are used to specify READY mode for $\overline{PCS4}$–$\overline{PCS6}$ as outlined below.

READY Generation Logic

The 80186 can generate a "READY" signal internally for each of the memory or peripheral $\overline{CS}$ lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the 80186 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each $\overline{CS}$ line or group of lines generated by the 80186. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). This means, for example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the $\overline{PCS0}$–3 READY mode, R2–R0 of MPCS set the $\overline{PCS4}$–6 READY mode.

Chip Select/Ready Logic and Reset

Upon reset, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the $\overline{UCS}$ line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to allow the maximum number of internal wait states in conjunction with external Ready consideration (i.e., UMCS resets to FFFBH).

- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the $\overline{PCS}$ lines will become active.

DMA CHANNELS

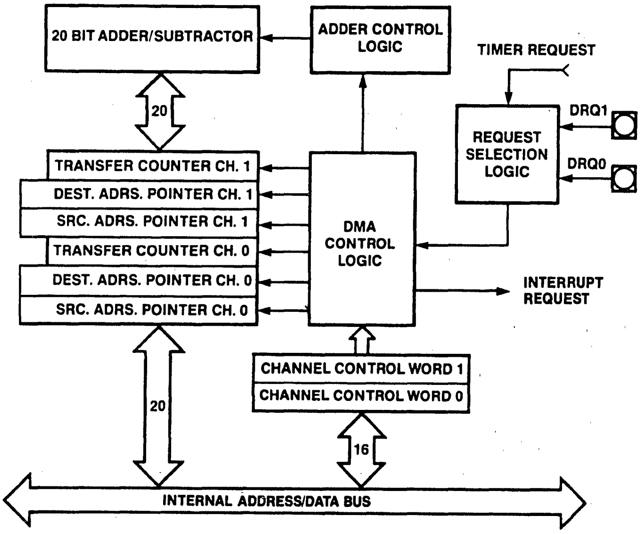
The 80186 DMA controller provides two independent DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Data can be transferred either in bytes (8 bits) or in words (16 bits) to or from even or odd addresses. Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer (by one or two depending on byte or word transfers). Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data. This provides a maximum data transfer rate of 1.25 Mword/sec or 2.5 MBytes/sec at 10 MHz.

DMA Operation

Each channel has six registers in the control block which define each channel's specific operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit destination pointer (2 words), a 16-bit Transfer Counter, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte or word transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 17). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	COH	DOH



210451-9

Figure 16. DMA Unit Block Diagram

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M/ $\overline{\text{IO}}$	DESTINATION DEC	INC	M/ $\overline{\text{IO}}$	SOURCE DEC	INC	TC	INT	SYN		P	T D R Q	X	CHG/ $\overline{\text{NOCHG}}$	ST/ $\overline{\text{STOP}}$	$\overline{\text{B}}$/ W

X = DON'T CARE.

Figure 17. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 81086 DMA channel. This register specifies:

- the mode of synchronization;
- whether bytes or words will be transferred;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

$\overline{\text{B}}$/W: Byte/Word (0/1) Transfers.

ST/$\overline{\text{STOP}}$: Start/stop (1/0) Channel.

CHG/$\overline{\text{NOCHG}}$: Change/Do not change (1/0) ST/$\overline{\text{STOP}}$ bit. If this bit is set when writing to the control word, the ST/$\overline{\text{STOP}}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{\text{STOP}}$ bit will not be altered. This bit is not stored; it will always be a 0 on read.

INT: Enable Interrupts to CPU on Transfer Count termination.

TC: If set, DMA will terminate when the contents of the Transfer Count register reach zero. The ST/$\overline{\text{STOP}}$ bit will also be reset at this point if TC is set. If this bit is cleared, the DMA unit will decrement the transfer count register for each DMA cycle, but the DMA transfer will not stop when the contents of the TC register reach zero.

SYN (2 bits)

00 No synchronization.

NOTE:

When unsynchronized transfers are specified, the TC bit will be ignored and the ST bit will be cleared upon the transfer count reaching zero, stopping the channel.

01 Source synchronization.

10 Destination synchronization.

11 Unused.

SOURCE:INC Increment source pointer by 1 or 2 (depends on $\overline{\text{B}}$/W) after each transfer.

M/$\overline{\text{IO}}$ Source pointer is in M/IO space (1/0).

DEC Decrement source pointer by 1 or 2 (depends on $\overline{\text{B}}$/W) after each transfer.

DEST: INC Increment destination pointer by 1 or 2 ($\overline{\text{B}}$/W) after each transfer.

M/$\overline{\text{IO}}$ Destination pointer is in M/IO space (1/0).

DEC Decrement destination pointer by 1 or 2 (depending on $\overline{\text{B}}$/W) after each transfer.

P Channel priority—relative to other channel.

0 low priority.

1 high priority.

Channels will alternate cycles if both set at same priority level.

TDRQ 0: Disable DMA requests from timer 2.

1: Enable DMA requests from timer 2.

Bit 3 Bit 3 is not used.

If both INC and DEC are specified for the same pointer, the pointer will remain constant after each cycle.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18). These pointers may be individually incremented or decremented after each transfer. If word transfers are performed the pointer is incremented or decremented by two. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers. Higher transfer rates can be obtained if all word transfers are performed to even addresses, since this will allow data to be accessed in a single memory access.

DMA Transfer Count Register

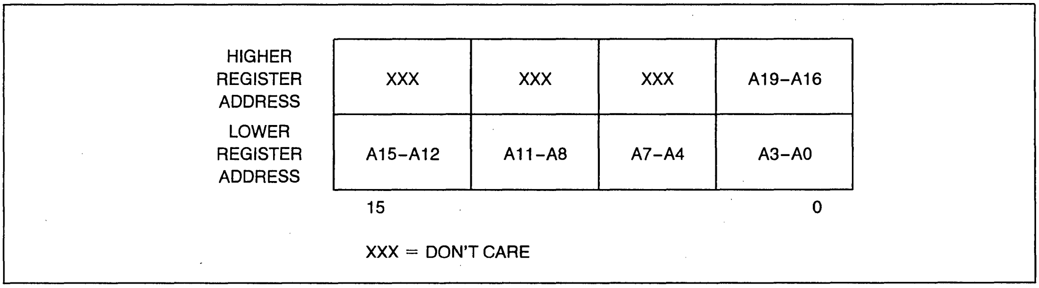
Each DMA channel maintains a 16-bit transfer count register (TC). This register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or if unsynchronized transfers are programmed, however, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). No prefetching occurs when source synchronized or unsynchronized transfers are performed, however. Data will not be fetched from the source address until the destination device signals that it is ready to receive it. When destination synchronized transfers are requested, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another DMA cycle will begin after two processor clocks. This is done to allow the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. These lead to the maximum DMA transfer rates shown in Table 14.

Table 14. Maximum DMA Transfer Rates @ 10 MHz

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	2.5MBytes/sec	2.5MBytes/sec
Source Synch.	2.5MBytes/sec	2.5MBytes/sec
Destination Synch.	1.7MBytes/sec	2.0MBytes/sec



	15			0
HIGHER REGISTER ADDRESS	XXX	XXX	XXX	A19–A16
LOWER REGISTER ADDRESS	A15–A12	A11–A8	A7–A4	A3–A0

XXX = DON'T CARE

Figure 18. DMA Memory Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed such that one channel is always given priority over the other, or they may be programmed such as to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses to odd memory locations; however, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/STOP bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also have been generated. Therefore the source and destination transfer pointers, and the transfer count register (if used) must be programmed before this bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/STOP bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

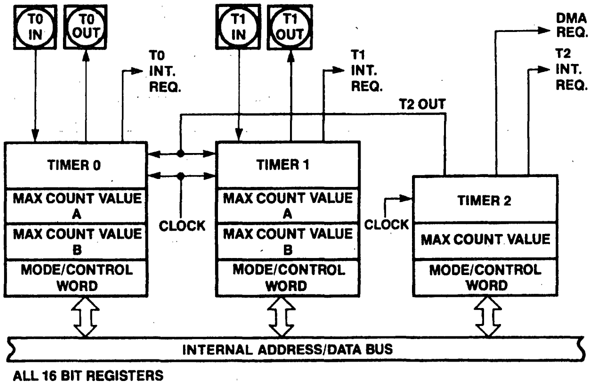
DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The Start/Stop bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80186 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, this third timer can be used as a prescaler to the other two, or as a DMA request source.



210451-10

Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the internal peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values programmed by the user. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 1 clock after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate (2 MHz for an 8 MHz CPU clock). Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

15	14	13	12	11		5	4	3	2	1	0
EN	$\overline{\text{INH}}$	INT	RIU	0	...	MC	RTG	P	EXT	ALT	CONT

Figure 20. Timer Mode/Control Register

ALT:

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT:

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If COUNT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

EXT:

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80186 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

P:

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

RTG:

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80186 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

EN:

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transistions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

$\overline{\text{INH}}$:

The inhibit bit allows for selective updating of the enable (EN) bit. If INH is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If INH is a zero during the write, the EN bit will be unaffected by the operation.This bit is not stored; it will always be a 0 on a read.

INT:

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller).

MC:

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set

regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts.

Programmer intervention is required to clear this bit.

RIU:

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The current contents of this register may be read or written by the processor at any time. If the register is written into while the timer is counting,the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. The condition which causes a timer to reset is equivalent between the current count value and the max count being used. This means that if the count is changed to be above the max count value, or if the max count value is changed to be below the current value, the timer will not reset to zero, but rather will count to its maximum value, "wrap around" to zero, then count until the max count is reached.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- All SEL (Select) bits are reset to zero. This selects MAX COUNT register A, resulting in the Timer Out pins going HIGH upon RESET.

INTERRUPT CONTROLLER

The 80186 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80186 interrupt controller has its own control register that set the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service rountines for lower priority interrupts may themselves be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The 80186 has a special slave mode in which the internal interrupt controller acts as a slave to an external master. The controller is programmed into this mode by setting bit 14 in the peripheral control block relocation register. (See Slave Mode section.)

MASTER MODE OPERATION

Interrupt Controller External Interface

For external interrupt sources, five dedicated pins are provided. One of these pins is dedicated to NMI, non-maskable interrupt. This is typically used for power-fail interrupts, etc. The other four pins may function either as four interrupt input lines with internally generated interrupt vectors, as an interrupt line and an interrupt acknowledge line (called the "cascade mode") along with two other input lines with internally generated interrupt vectors, or as two interrupt lines and two dedicated interrupt acknowledge output lines. When the interrupt lines are configured in cascade mode, the 80186 interrupt controller will not generate internal interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80186 on the second cycle. The capability to interface to external 8259A programmable interrupt controllers is thus provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in master mode are similar to the 8259A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

Fully Nested Mode

When in the fully nested mode four pins are used as direct interrupt requests as in Figure 22. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled without being themselves interrupted by lower-priority interrupts. Since interrupts are enabled, higher-priority interrupts will be serviced.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is issued at the end of the service routine just before the issuance of the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

Cascade Mode

The 80186 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 23. INT0 is an interrupt input interfaced to an 8259A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 8259As. Three levels of priority are created, requiring priority resolution in the 80186 interrupt controller, the master 8259As, and the slave 8259As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

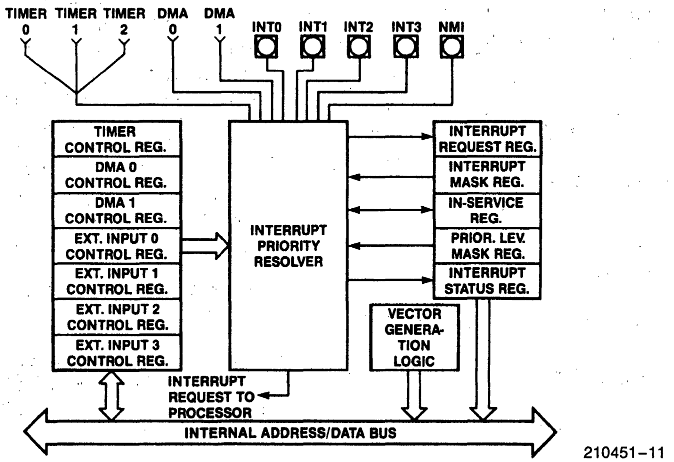


Figure 21. Interrupt Controller Block Diagram

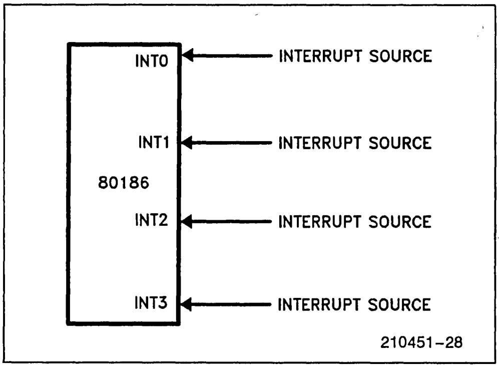


Figure 22. Fully Nested (Direct) Mode Interrupt Controller Connections

Special Fully Nested Mode

This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 8259A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80186 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80186 controller until the 80186 in-service bit is reset. In special fully nested mode, the 80186 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80186 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the external master's IS register is required to determine if there is more than one bit set. If so, the IS bit in the 80186 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 32). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80186 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Master Mode Features

Programmable Priority

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, it allows other requests to be serviced.

End-of-Interrupt Command

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

Trigger Mode

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the 80186 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

Interrupt Vectoring

The 80186 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 24. It contains 15 registers. All registers can both be read or written unless specified otherwise.

In-Service Register

This register can be read from or written into. The format is shown in Figure 25. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command issued by the CPU.

Interrupt Request Register

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 25. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits show exactly when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Writes to the interrupt request register will affect the D0 and D1 interrupt request bits. Setting either bit will cause the corresponding interrupt request while clearing either bit will remove the corresponding interrupt request. All other bits in the register are read-only.

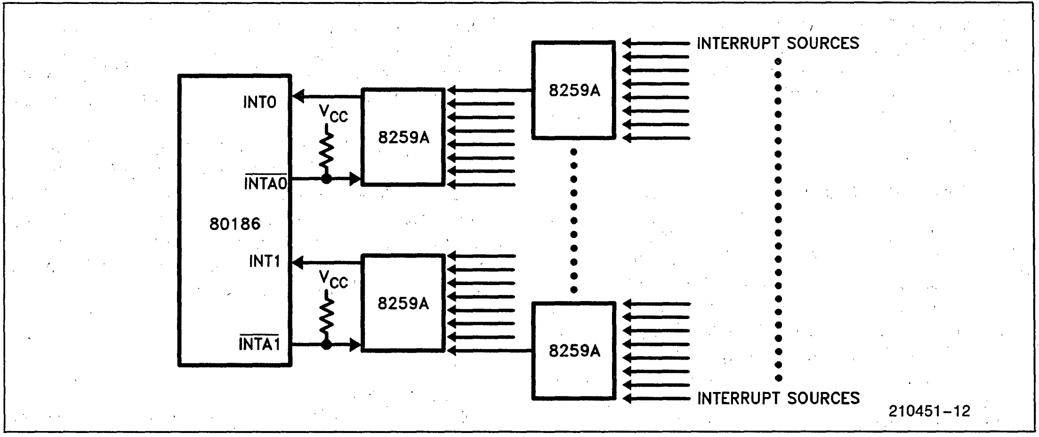


Figure 23. Cascade and Special Fully Nested Mode Interrupt Controller Connections

Mask Register

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 25. A one in a bit position corresponding to a particular source serves to mask the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 24. Interrupt Controller Registers (Master Mode)

Priority Mask Register

This register is used to mask all interrupts below particular interrupt priority levels. The format of this register is shown in Figure 26. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so no interrupts are masked due to priority number.

Interrupt Status Register

This register contains general interrupt controller status information. The format of this register is shown in Figure 27. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. The purpose of this bit is to allow prompt service of all non-maskable interrupts. This bit may also be set by the programmer.

IRTx: These three bits represent the individual timer interrupt request bits. These bits are used to differentiate the timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt request. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

15	14				10	9	8	7	6	5	4	3	2	1	0
0	0	•	•	•	0	0	0	I3	I2	I1	I0	D1	D0	0	TMR

Figure 25. In-Service, Interrupt Request, and Mask Register Formats

15	14											3	2	1	0
0	0	•	•	•	•	•	•	•	•	•		0	PRM2	PRM1	PRM0

Figure 26. Priority Mask Register Format

15	14						7	6	5	4	3	2	1	0
DHLT	0	•	•	•	•	•	0	0	0	0	0	IRT2	IRT1	IRT0

Figure 27. Interrupt Status Register Format (Master Mode)

Timer, DMA 0, 1; Control Register

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 28. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 Control Registers

These registers are the control words for the four external input pins. Figure 29 shows the format of the INT0 and INT1 Control registers; Figure 30 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest Priority = 000, Lowest Priority = 111

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is proceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct

SFNM: Special fully nested mode bit, 1 = SFNM

EOI Register

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 31. It initiates an EOI command when written to by the 80186 CPU.

The bits in the EOI register are encoded as follows:

S_X: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10.

NOTE:

To reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

15	14		4	3	2	1	0
0	0	• • • • • • • •	0	MSK	PR2	PR1	PR0

Figure 28. Timer/DMA Control Registers Formats

15	14		7	6	5	4	3	2	1	0
0	0	• • • • •	0	SFNM	C	LTM	MSK	PR2	PR1	PR0

Figure 29. INT0/INT1 Control Register Formats

15	14		5	4	3	2	1	0
0	0	• • • • • • •	0	LTM	MSK	PR2	PR1	PR0

Figure 30. INT2/INT3 Control Register Formats

NSPEC/: A bit that determines the type of EOI command. Nonspecific = 1, Specific = 0.
SPEC

Poll and Poll Status Registers

These registers contain polling information. The format of these registers is shown in Figure 32. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_X: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

SLAVE MODE OPERATION

When slave mode is used, the internal 80186 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80186 resources will be monitored by the internal interrupt controller, while the external controller functions as the system master interrupt controller. Upon reset, the 80186 will be in master mode. To provide for slave mode operation bit 14 of the relocation register should be set.

Because of pin limitations caused by the need to interface to an external 8259A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80186 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

In slave mode each peripheral must be assigned a unique priority to ensure proper interrupt controller operation. Therefore, it is the programmer's responsibility to assign correct priorities and initialize interrupt control registers before enabling interrupts.

These level assignments must remain fixed in the iRMX 86 mode of operation.

Slave Mode External Interface

The configuration of the 80186 with respect to an external 8259A master is shown in Figure 33. The INT0 (pin 45) input is used as the 80186 CPU interrupt input. INT3 (pin 41) functions as an output to send the 80186 slave-interrupt-request to one of the 8 master-PIC-inputs.

15	14	13		5	4	3	2	1	0
SPEC/ NSPEC	0	0	• • • • • • •	0	S4	S3	S2	S1	S0

Figure 31. EOI Register Format

15	14	13		5	4	3	2	1	0
INT REQ	0	0	• • • • • • •	0	S4	S3	S2	S1	S0

Figure 32. Poll and Poll Status Register Formats

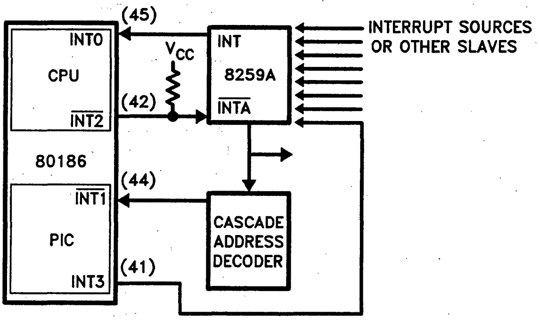


Figure 33. Slave Mode Interrupt Controller Connections

Correct master-slave interface requires decoding of the slave addresses (CAS0-2). Slave 8259As do this internally. Because of pin limitations, the 80186 slave address will have to be decoded externally. $\overline{\text{INT1}}$ (pin 44) is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{\text{INT2}}$ (pin 42) is used as an acknowledge output, suitable to drive the $\overline{\text{INTA}}$ input of an 8259A.

Interrupt Nesting

Slave mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the Slave Mode

Vector generation in slave mode is exactly like that of an 8259A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In slave mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the Slave Mode

All control and command registers are located inside the internal peripheral control block. Figure 34 shows the offsets of these registers.

End-of-Interrupt Register

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 35. It initiates an EOI command when written by the 80186 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

In-Service Register

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 36. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

Interrupt Request Register

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 36. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request. As in master mode, D0 and D1 are read/write; all other bits are read only.

Mask Register

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 36. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

Control Registers

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 37. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_X: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_X bits.

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT-REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 34. Interrupt Controller Registers (Slave Mode)

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	L2	L1	L0

Figure 35. Specific EOI Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 36. In-Service, Interrupt Request, and Mask Register Format

Interrupt Vector Register

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 38. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

Priority-Level Mask Register

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

Interrupt Status Register

This register is defined as in master mode except that DHLT is not implemented. (See Figure 27).

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to master mode.

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	MSK	PR2	PR1	PR0

Figure 37. Control Word Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	t4	t3	t2	t1	t0	0	0	0

Figure 38. Interrupt Vector Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	m2	m1	m0

Figure 39. Priority Level Mask Register

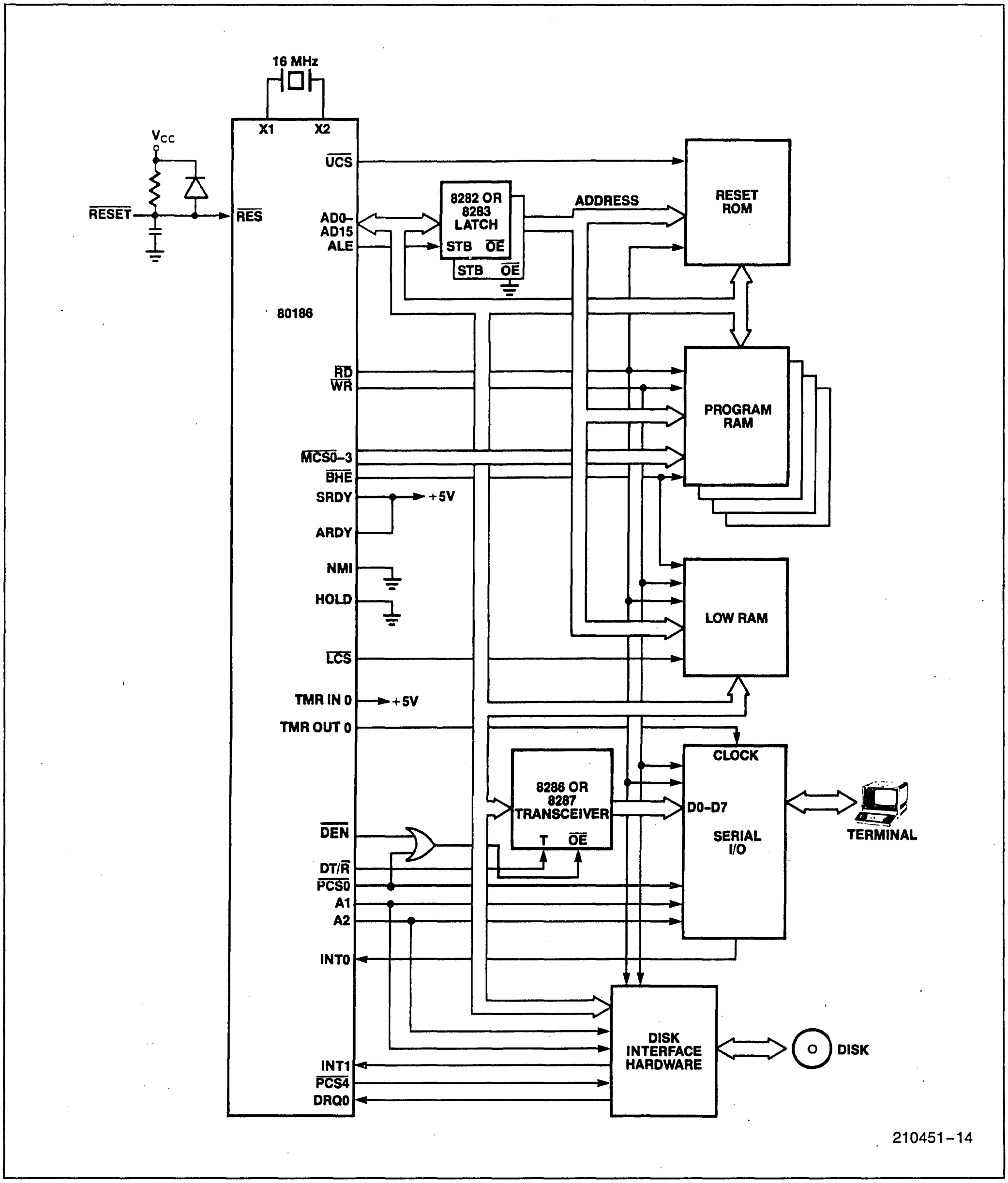


Figure 40. Typical 80186 Computer

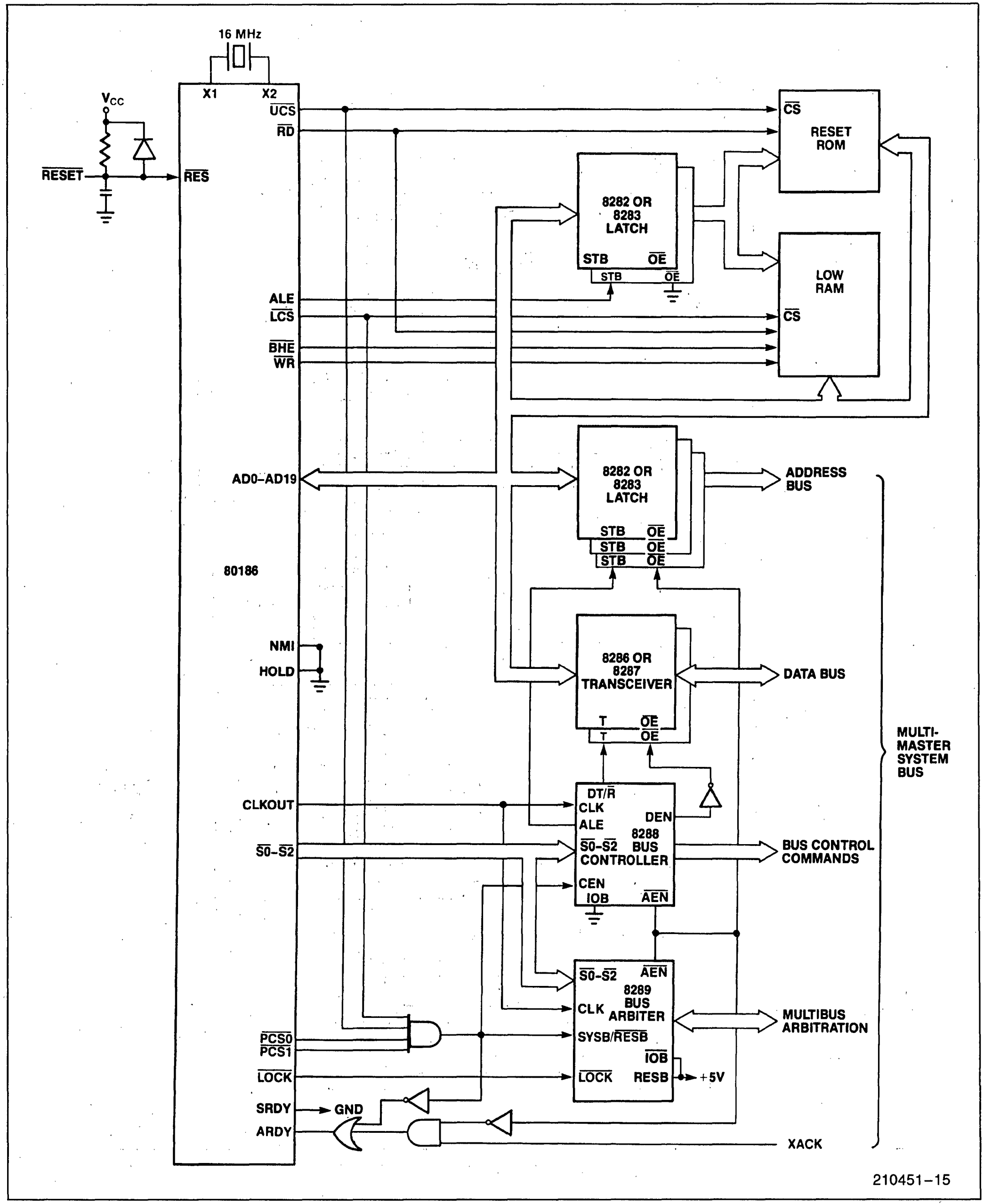


Figure 41. Typical 80186 Multi-Master Bus Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ......0°C to 70°C
Storage Temperature ....−65°C to +150°C
Voltage on any Pin with
  Respect to Ground..............−1.0V to +7V
Power Dissipation 3W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%)

Applicable to 80186 (8 MHz), 80186-10 (10 MHz).

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage (All except X1 and ($\overline{RES}$)	2.0	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_a = 2.5 mA for $\overline{S0}$–$\overline{S2}$ I_a = 2.0 mA for all other Outputs
V_{OH}	Output High Voltage	2.4		V	I_{oa} = −400 μA
I_{CC}	Power Supply Current		600*	mA	T_A = −40°C
			550	mA	T_A = 0°C
			415	mA	T_A = +70°C
I_{LI}	Input Leakage Current		±10	μA	0V < V_{IN} < V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V < V_{OUT} < V_{CC}
V_{CLO}	Clock Output Low		0.6	V	I_a = 4.0 mA
V_{CHO}	Clock Output High	4.0		V	I_{oa} = −200 μA
V_{CLI}	Clock Input Low Voltage	−0.5	0.6	V	
V_{CHI}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

*For extended temperature parts only.

PIN TIMINGS

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%)

80186 Timing Requirements All Timings Measured At 1.5V Unless Otherwise Noted.

Symbol	Parameter	80186 (8 MHz)		80186-10 (10 MHz)		Units	Test Conditions
		Min	Max	Min	Max		
T_{DVCL}	Data in Setup (A/D)	20		15		ns	
T_{CLDX}	Data in Hold (A/D)	10		8		ns	
T_{ARYHCH}	Asynchronous Ready (ARDY) Active Setup Time*	20		15		ns	
T_{ARYLCL}	ARDY Inactive Setup Time	35		25		ns	
T_{CLARX}	ARDY Hold Time	15		15		ns	
T_{ARYCHL}	Asynchronous Ready Inactive Hold Time	15		15		ns	
T_{SRYCL}	Synchronous Ready (SRDY) Transition Setup Time	20		20		ns	
T_{CLSRY}	SRDY Transition Hold Time	15		15		ns	
T_{HVCL}	HOLD Setup*	25		20		ns	
T_{INVCH}	INTR, NMI, $\overline{\text{TEST}}$, TIM IN, Setup*	25		25		ns	
T_{INVCL}	DRQ0, DRQ1, Setup*	25		20		ns	

80186 Master Interface Timing Responses

T_{CLAV}	Address Valid Delay	5	55	5	44	ns	C_L = 20–200 pF all Outputs (Except T_{CLTMV}) @ 8 & 10 MHz
T_{CLAX}	Address Hold	10		10		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	35	T_{CLAX}	30	ns	
T_{CHCZ}	Command Lines Float Delay		45		40	ns	
T_{CHCV}	Command Lines Valid Delay (after Float)		55		45	ns	
T_{LHLL}	ALE Width	$T_{CLCL}-35$		$T_{CLCL}-30$		ns	
T_{CHLH}	ALE Active Delay		35		30	ns	
T_{CHLL}	ALE Inactive Delay		35		30	ns	
T_{LLAX}	Address Hold from ALE Inactive	$T_{CHCL}-25$		$T_{CHCL}-20$		ns	
T_{CLDV}	Data Valid Delay	10	44	10	40	ns	
T_{CLDOX}	Data Hold Time	10		10		ns	
T_{WHDX}	Data Hold after WR	$T_{CLCL}-40$		$T_{CLCL}-34$		ns	
T_{CVCTV}	Control Active Delay 1	5	50	5	40	ns	
T_{CHCTV}	Control Active Delay 2	10	55	10	44	ns	
T_{CVCTX}	Control Inactive Delay	5	55	5	44	ns	
T_{CVDEX}	$\overline{\text{DEN}}$ Inactive Delay (Non-Write Cycle)	10	70	10	56	ns	

*To guarantee recognition at next clock.

PIN TIMINGS (Continued)

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%) (Continued)

80186 Master Interface Timing Responses (Continued)

Symbol	Parameter	80186 (8 MHz)		80186-10 (10 MHz)		Units	Test Conditions
		Min	Max	Min	Max		
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		ns	
T_{CLRL}	$\overline{RD}$ Active Delay	10	70	10	56	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	10	55	10	44	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active	$T_{CLCL}-40$		$T_{CLCL}-40$		ns	
T_{CLHAV}	HLDA Valid Delay	5	50	5	40	ns	
T_{RLRH}	$\overline{RD}$ Width	$2T_{CLCL}-50$		$2T_{CLCL}-46$		ns	
T_{WLWH}	$\overline{WR}$ Width	$2T_{CLCL}-40$		$2T_{CLCL}-34$		ns	
T_{AVAL}	Address Valid to ALE Low	$T_{CLCH}-25$		$T_{CLCH}-19$		ns	
T_{CHSV}	Status Active Delay	10	55	10	45	ns	
T_{CLSH}	Status Inactive Delay	10	65	10	50	ns	
T_{CLTMV}	Timer Output Delay		60		48	ns	100 pF max @ 8 & 10 MHz
T_{CLRO}	Reset Delay		60		48	ns	
T_{CHQSV}	Queue Status Delay		35		28	ns	
T_{CHDX}	Status Hold Time	10		10		ns	
T_{AVCH}	Address Valid to Clock High	10		10		ns	
T_{CLLV}	$\overline{LOCK}$ Valid/Invalid Delay	5	65	5	60	ns	

80186 Chip-Select Timing Responses

T_{CLCSV}	Chip-Select Active Delay		66		45	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	35		35		ns	
T_{CHCSX}	Chip-Select Inactive Delay	5	35	5	32	ns	

80186 CLKIN Requirements

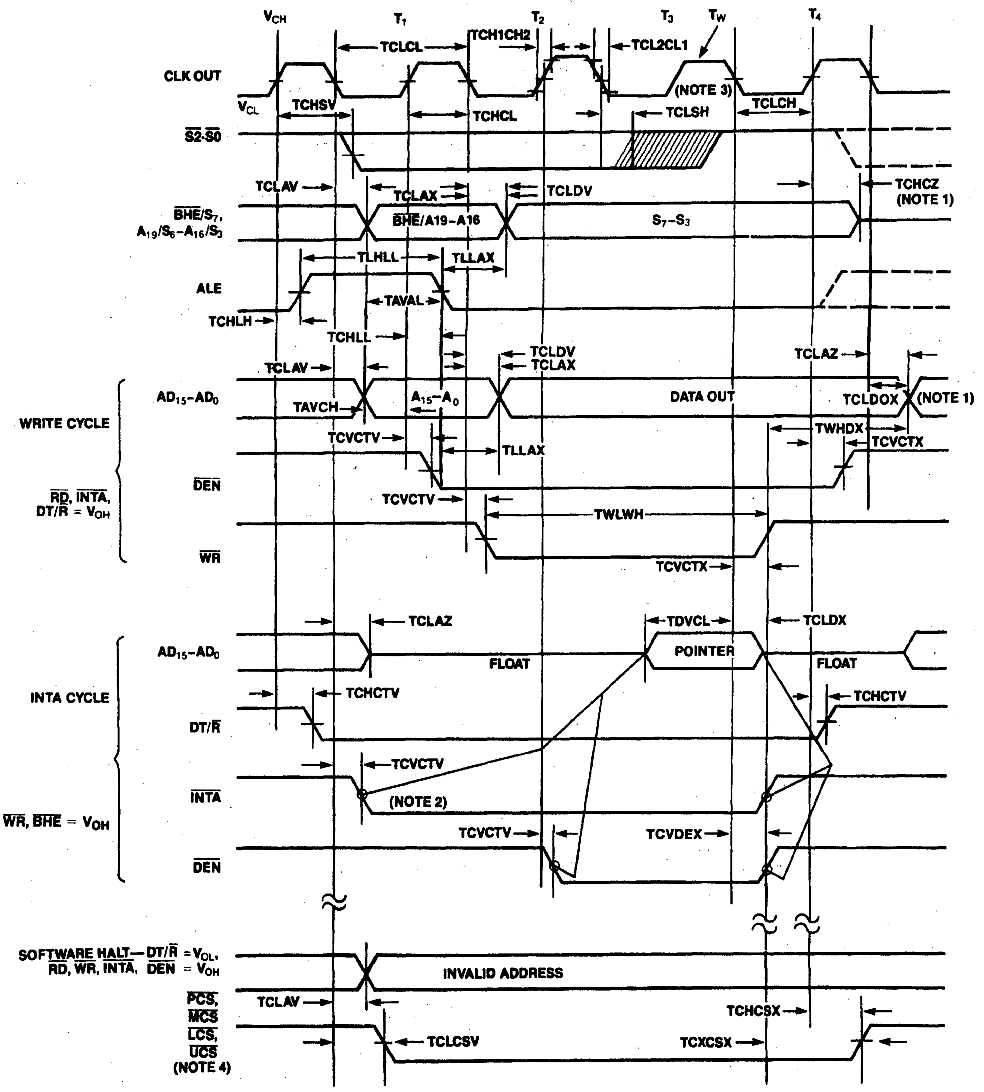
T_{CKIN}	CLKIN Period	62.5	250	50	250	ns	
T_{CKHL}	CLKIN Fall Time		10		10	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		10		10	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	25		20		ns	1.5V
T_{CHCK}	CLKIN High Time	25		20		ns	1.5V

80186 CLKOUT Timing (200 pF load)

T_{CICO}	CLKIN to CLKOUT Skew		50		25	ns	
T_{CLCL}	CLKOUT Period	125	500	100	500	ns	
T_{CLCH}	CLKOUT Low Time	$\frac{1}{2} T_{CLCL}-7.5$		$\frac{1}{2} T_{CLCL}-6.0$		ns	1.5V
T_{CHCL}	CLKOUT High Time	$\frac{1}{2} T_{CLCL}-7.5$		$\frac{1}{2} T_{CLCL}-6.0$		ns	1.5V
T_{CH1CH2}	CLKOUT Rise Time		15		12	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		15		12	ns	3.5 to 1.0V

WAVEFORMS

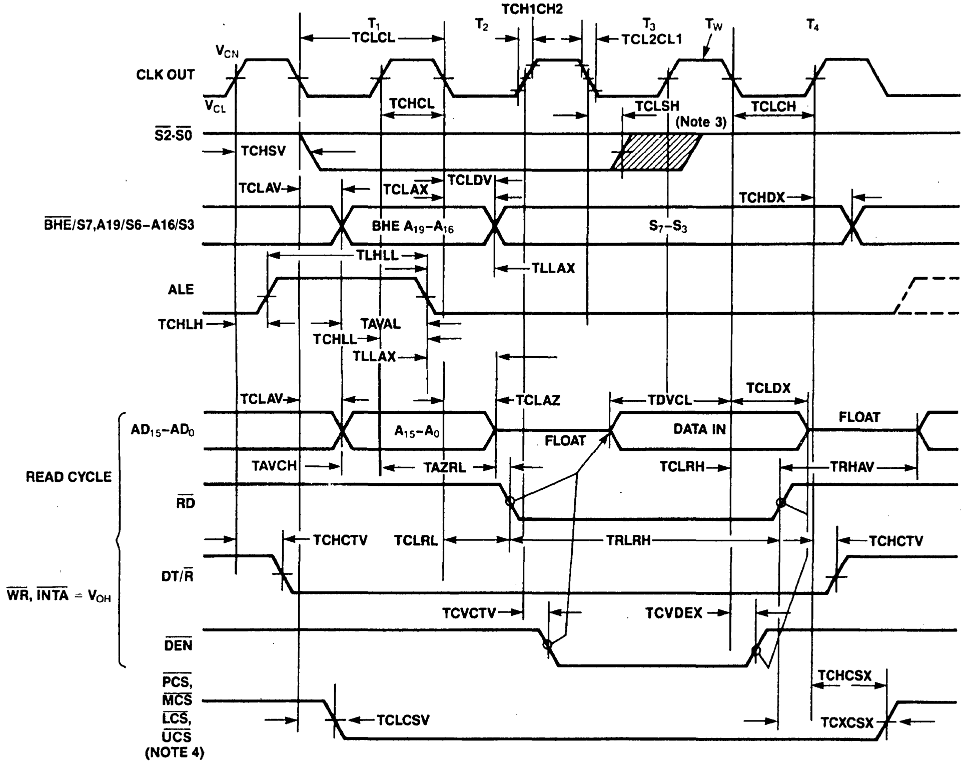
MAJOR CYCLE TIMING



210451-23

WAVEFORMS (Continued)

MAJOR CYCLE TIMING (Continued)

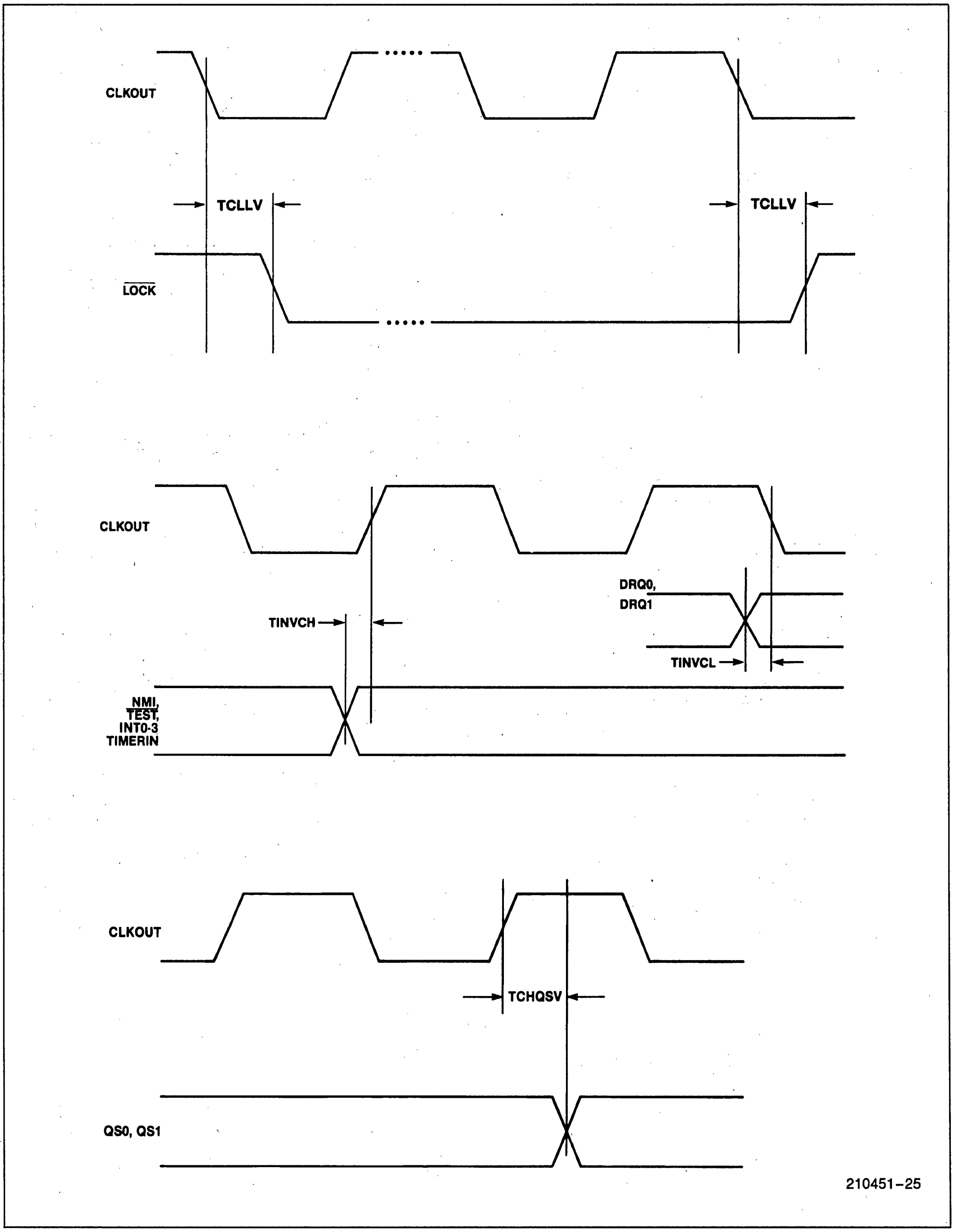


210451-24

NOTES:

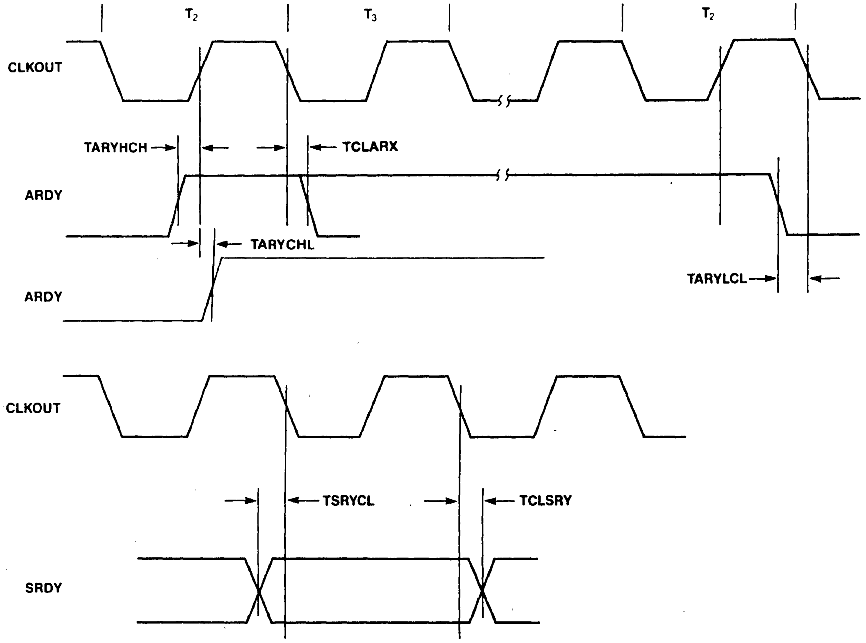
1. Following a Write cycle, the Local Bus is floated by the 80186 only when the 80186 enters a "Hold Acknowledge" state.
2. INTA occurs one clock later in slave mode.
3. Status inactive just prior to T_4.
4. Latched A1 and A2 have same timings as $\overline{PCS5}$ and $\overline{PCS6}$.

WAVEFORMS (Continued)



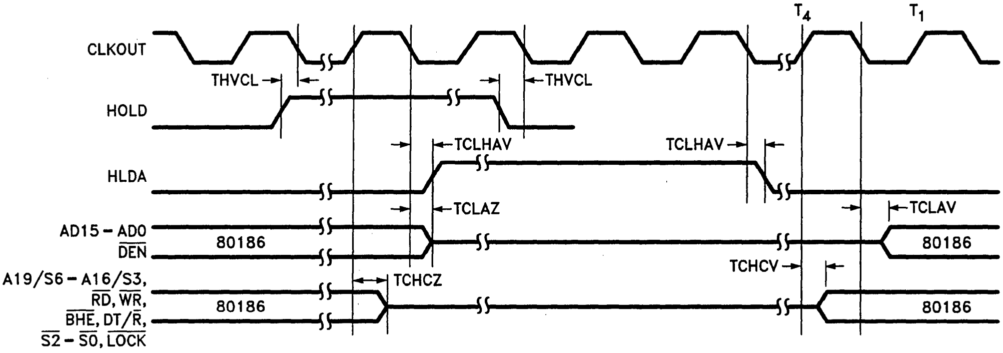
WAVEFORMS (Continued)

READY TIMING



210451-29

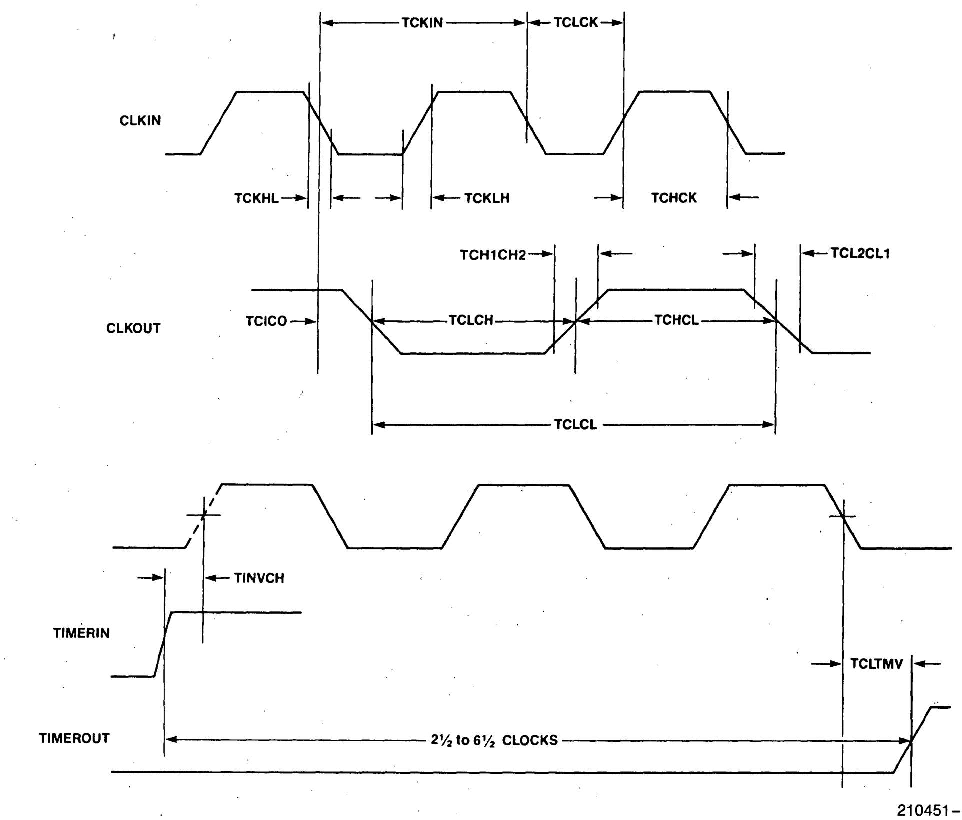
HOLD-HLDA TIMING



210451-26

WAVEFORMS (Continued)

TIMER ON 80186



210451-27

80186 EXECUTION TIMINGS

Since the bus interface unit and execution unit operate independently, a determination of 80186 program execution timing must consider both the bus cycles necessary to prefetch instructions as well as the number of EU cycles necessary to execute instructions. The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDS occur.
- All word-data is located on even-address boundaries.

All instructions which involve memory accesses can also require one or two additional clocks above the minimum timings shown due to the asynchronous handshake between the BIU and execution unit.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

With a 16-bit BIU, the 80186 has sufficient bus performance to ensure that an adequate number of prefetched bytes will reside in the queue most of the time. Therefore, actual program execution will not be substantially greater than that derived from adding the instruction timings shown.

INSTRUCTION SET SUMMARY

Function	Format				Clock Cycles	Comments
DATA TRANSFER						
MOV = Move:						
Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m			2/12	
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m			2/9	
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1	12–13	8/16-bit
Immediate to register	1 0 1 1 w reg	data	data if w = 1		3–4	8/16-bit
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high		8	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high		9	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m			2/9	
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m			2/11	
PUSH = Push:						
Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m			16	
Register	0 1 0 1 0 reg				10	
Segment register	0 0 0 reg 1 1 0				9	
Immediate	0 1 1 0 1 0 s 0	data	data if s = 0		10	
PUSHA = Push All	0 1 1 0 0 0 0 0				36	
POP = Pop:						
Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m			20	
Register	0 1 0 1 1 reg				10	
Segment register	0 0 0 reg 1 1 1	(reg≠01)			8	
POPA = Pop All	0 1 1 0 0 0 0 1				51	
XCHG = Exchange:						
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m			4/17	
Register with accumulator	1 0 0 1 0 reg				3	
IN = Input from:						
Fixed port	1 1 1 0 0 1 0 w	port			10	
Variable port	1 1 1 0 1 1 0 w				8	
OUT = Output to:						
Fixed port	1 1 1 0 0 1 1 w	port			9	
Variable port	1 1 1 0 1 1 1 w				7	
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1				11	
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m			6	
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m	(mod≠11)		18	
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m	(mod≠11)		18	
LAHF = Load AH with flags	1 0 0 1 1 1 1 1				2	
SAHF = Store AH into flags	1 0 0 1 1 1 1 0				3	
PUSHF = Push flags	1 0 0 1 1 1 0 0				9	
POPF = Pop flags	1 0 0 1 1 1 0 1				8	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
DATA TRANSFER (Continued)						
SEGMENT = Segment Override:						
CS	0 0 1 0 1 1 1 0				2	
SS	0 0 1 1 0 1 1 0				2	
DS	0 0 1 1 1 1 1 0				2	
ES	0 0 1 0 0 1 1 0				2	
ARITHMETIC						
ADD = Add:						
Reg/memory with register to either	0 0 0 0 0 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s w = 01	4/16	
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w = 1		3/4	8/16-bit
ADC = Add with carry:						
Reg/memory with register to either	0 0 0 1 0 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s w = 01	4/16	
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w = 1		3/4	8/16-bit
INC = Increment:						
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m			3/15	
Register	0 1 0 0 0 reg				3	
SUB = Subtract:						
Reg/memory and register to either	0 0 1 0 1 0 d w	mod reg r/m			3/10	
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s w = 01	4/16	
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w = 1		3/4	8/16-bit
SBB = Subtract with borrow:						
Reg/memory and register to either	0 0 0 1 1 0 d w	mod reg r/m			3/10	
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s w = 01	4/16	
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w = 1		3/4	8/16-bit
DEC = Decrement						
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m			3/15	
Register	0 1 0 0 1 reg				3	
CMP = Compare:						
Register/memory with register	0 0 1 1 1 0 1 w	mod reg r/m			3/10	
Register with register/memory	0 0 1 1 1 0 0 w	mod reg r/m			3/10	
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w = 01	3/10	
Immediate with accumulator	0 0 1 1 1 1 0 w	data	data if w = 1		3/4	8/16-bit
NEG = Change sign register/memory	1 1 1 1 0 1 1 w	mod 0 1 1 r/m			3/10	
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1				8	
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1				4	
AAS = ASCII adjust for subtract	0 0 1 1 1 1 1 1				7	
DAS = Decimal adjust for subtract	0 0 1 0 1 1 1 1				4	
MUL = Multiply (unsigned):	1 1 1 1 0 1 1 w	mod 100 r/m				
Register-Byte					26–28	
Register-Word					35–37	
Memory-Byte					32–34	
Memory-Word					41–43	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
ARITHMETIC (Continued)						
IMUL = Integer multiply (signed):	1 1 1 1 0 1 1 w	mod 1 0 1 r/m				
Register-Byte					25–28	
Register-Word					34–37	
Memory-Byte					31–34	
Memory-Word					40–43	
IMUL = Integer Immediate multiply (signed)	0 1 1 0 1 0 s 1	mod reg r/m	data	data if s=0	22–25/ 29–32	
DIV = Divide (unsigned):	1 1 1 1 0 1 1 w	mod 1 1 0 r/m				
Register-Byte					29	
Register-Word					38	
Memory-Byte					35	
Memory-Word					44	
IDIV = Integer divide (signed):	1 1 1 1 0 1 1 w	mod 1 1 1 r/m				
Register-Byte					44–52	
Register-Word					53–61	
Memory-Byte					50–58	
Memory-Word					59–67	
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0			19	
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0			15	
CBW = Convert byte to word	1 0 0 1 1 0 0 0				2	
CWD = Convert word to double word	1 0 0 1 1 0 0 1				4	
LOGIC						
Shift/Rotate Instructions:						
Register/Memory by 1	1 1 0 1 0 0 0 w	mod TTT r/m			2/15	
Register/Memory by CL	1 1 0 1 0 0 1 w	mod TTT r/m			5+n/17+n	
Register/Memory by Count	1 1 0 0 0 0 0 w	mod TTT r/m	count		5+n/17+n	
		TTT	**Instruction**			
		0 0 0	ROL			
		0 0 1	ROR			
		0 1 0	RCL			
		0 1 1	RCR			
		1 0 0	SHL/SAL			
		1 0 1	SHR			
		1 1 1	SAR			
AND = And:						
Reg/memory and register to either	0 0 1 0 0 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w=1	4/16	
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w=1		3/4	8/16-bit
TEST = And function to flags, no result:						
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m			3/10	
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w=1	4/10	
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w=1		3/4	8/16-bit
OR = Or:						
Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w=1	4/16	
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w=1		3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
LOGIC (Continued)						
XOR = **Exclusive or:**						
Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w=1	4/16	
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w=1		3/4	8/16-bit
NOT = Invert register/memory	1 1 1 1 0 1 1 w	mod 0 1 0 r/m			3/10	
STRING MANIPULATION						
MOVS = Move byte/word	1 0 1 0 0 1 0 w				14	
CMPS = Compare byte/word	1 0 1 0 0 1 1 w				22	
SCAS = Scan byte/word	1 0 1 0 1 1 1 w				15	
LODS = Load byte/wd to ALAX	1 0 1 0 1 1 0 w				12	
STOS = Stor byte/wd from ALA	1 0 1 0 1 0 1 w				10	
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w				14	
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w				14	
Repeated by count in CX						
MOVS = Move string	1 1 1 1 0 0 1 0	1 0 1 0 0 1 0 w			8+8n	
CMPS = Compare string	1 1 1 1 0 0 1 z	1 0 1 0 0 1 1 w			5+22n	
SCAS = Scan string	1 1 1 1 0 0 1 z	1 0 1 0 1 1 1 w			5+15n	
LODS = Load string	1 1 1 1 0 0 1 0	1 0 1 0 1 1 0 w			6+11n	
STOS = Store string	1 1 1 1 0 0 1 0	1 0 1 0 1 0 1 w			6+9n	
INS = Input string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 0 w			8+8n	
OUTS = Output string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 1 w			8+8n	
CONTROL TRANSFER						
CALL = **Call:**						
Direct within segment	1 1 1 0 1 0 0 0	disp-low	disp-high		15	
Register/memory indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m			13/19	
Direct intersegment	1 0 0 1 1 0 1 0	segment offset			23	
		segment selector				
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	(mod ≠ 11)		38	
JMP = **Unconditional jump:**						
Short/long	1 1 1 0 1 0 1 1	disp-low			14	
Direct within segment	1 1 1 0 1 0 0 1	disp-low	disp-high		14	
Register/memory indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m			11/17	
Direct intersegment	1 1 1 0 1 0 1 0	segment offset			14	
		segment selector				
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	(mod ≠ 11)		26	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
CONTROL TRANSFER (Continued) **RET = Return from CALL:**						
Within segment	11000011				16	
Within seg adding immed to SP	11000010	data-low	data-high		18	
Intersegment	11001011				22	
Intersegment adding immediate to SP	11001010	data-low	data-high		25	
JE/JZ = Jump on equal/zero	01110100	disp			4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100	disp			4/13	
JLE/JNG = Jump on less or equal/not greater	01111110	disp			4/13	
JB/JNAE = Jump on below/not above or equal	01110010	disp			4/13	
JBE/JNA = Jump on below or equal/not above	01110110	disp			4/13	
JP/JPE = Jump on parity/parity even	01111010	disp			4/13	
JO = Jump on overflow	01110 000	disp			4/13	
JS = Jump on sign	01111000	disp			4/13	
JNE/JNZ = Jump on not equal/not zero	01110101	disp			4/13	
JNL/JGE = Jump on not less/greater or equal	01111101	disp			4/13	
JNLE/JG = Jump on not less or equal/greater	01111111	disp			4/13	
JNB/JAE = Jump on not below/above or equal	01110011	disp			4/13	
JNBE/JA = Jump on not below or equal/above	01110111	disp			4/13	
JNP/JPO = Jump on not par/par odd	01111011	disp			4/13	
JNO = Jump on not overflow	01110001	disp			4/13	
JNS = Jump on not sign	01111001	disp			4/13	
JCXZ = Jump on CX zero	11100011	disp			5/15	
LOOP = Loop CX times	11100010	disp			6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001	disp			6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000	disp			6/16	
ENTER = Enter Procedure	11001000	data-low	data-high	L		
L = 0					15	
L = 1					25	
L > 1					22+16(n−1)	
LEAVE = Leave Procedure	11001001				8	
INT = Interrupt:						
Type specified	11001101	type			47	
Type 3	11001100				45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	11001110				48/4	
IRET = Interrupt return	11001111				28	
BOUND = Detect value out of range	01100010	mod reg r/m			33-35	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format		Clock Cycles	Comments
PROCESSOR CONTROL				
CLC = Clear carry	11111000		2	
CMC = Complement carry	11110101		2	
STC = Set carry	11111001		2	
CLD = Clear direction	11111100		2	
STD = Set direction	11111101		2	
CLI = Clear interrupt	11111010		2	
STI = Set interrupt	11111011		2	
HLT = Halt	11110100		2	
WAIT = Wait	10011011		6	if $\overline{\text{test}}$ = 0
LOCK = Bus lock prefix	11110000		2	
ESC = Processor Extension Escape	11011TTT	mod LLL r/m	6	
	(TTT LLL are opcode to processor extension)			

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

intel®

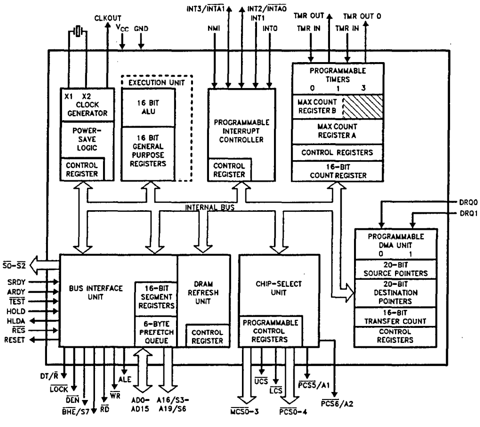
ADVANCE INFORMATION

80C186
CHMOS HIGH INTEGRATION 16-BIT MICROPROCESSOR

- **Operation Modes Include:**
  - **—Enhanced Mode Which Has**
    - **—DRAM Refresh**
    - **—Power-Save Logic**
    - **—Direct Interface to New Numerics Coprocessor**
  - **—Compatible Mode**
    - **—NMOS 80186 Pin for Pin Replacement for Non-Numerics Applications**
- **Integrated Feature Set**
  - **—Enhanced 80C86/C88 CPU**
  - **—Clock Generator**
  - **—2 Independent DMA Channels**
  - **—Programmable Interrupt Controller**
  - **—3 Programmable 16-Bit Timers**
  - **—Dynamic RAM Refresh Control Unit**
  - **—Programmable Memory and Peripheral Chip Select Logic**
  - **—Programmable Wait State Generator**
  - **—Local Bus Controller**
  - **—Power Save Logic**
  - **—System-Level Testing Support (High Impedance Test Mode)**
- **Available in 16 MHz (80C186-16), 12.5 MHz (80C186-12) and 10 MHz (80C186-10) Versions**
- **Direct Addressing Capability to 1 MByte and 64 KByte I/O**
- **Completely Object Code Compatible with All Existing 8086/8088 Software and Also Has 10 Additional Instructions over 8086/8088**
- **Complete System Development Support**
  - **—All 8086 and NMOS 80186 Software Development Tools Can Be Used for 80C186 System Development**
    - **—Assembler, PL/M, Pascal, Fortran, and System Utilities**
    - **—In-Circuit-Emulator (ICE™-C186)**
- **Available in 68 Pin:**
  - **—Plastic Leaded Chip Carrier (PLCC)**
  - **—Ceramic Pin Grid Array (PGA)**
  - **—Ceramic Leadless Chip Carrier (JEDEC A Package)**

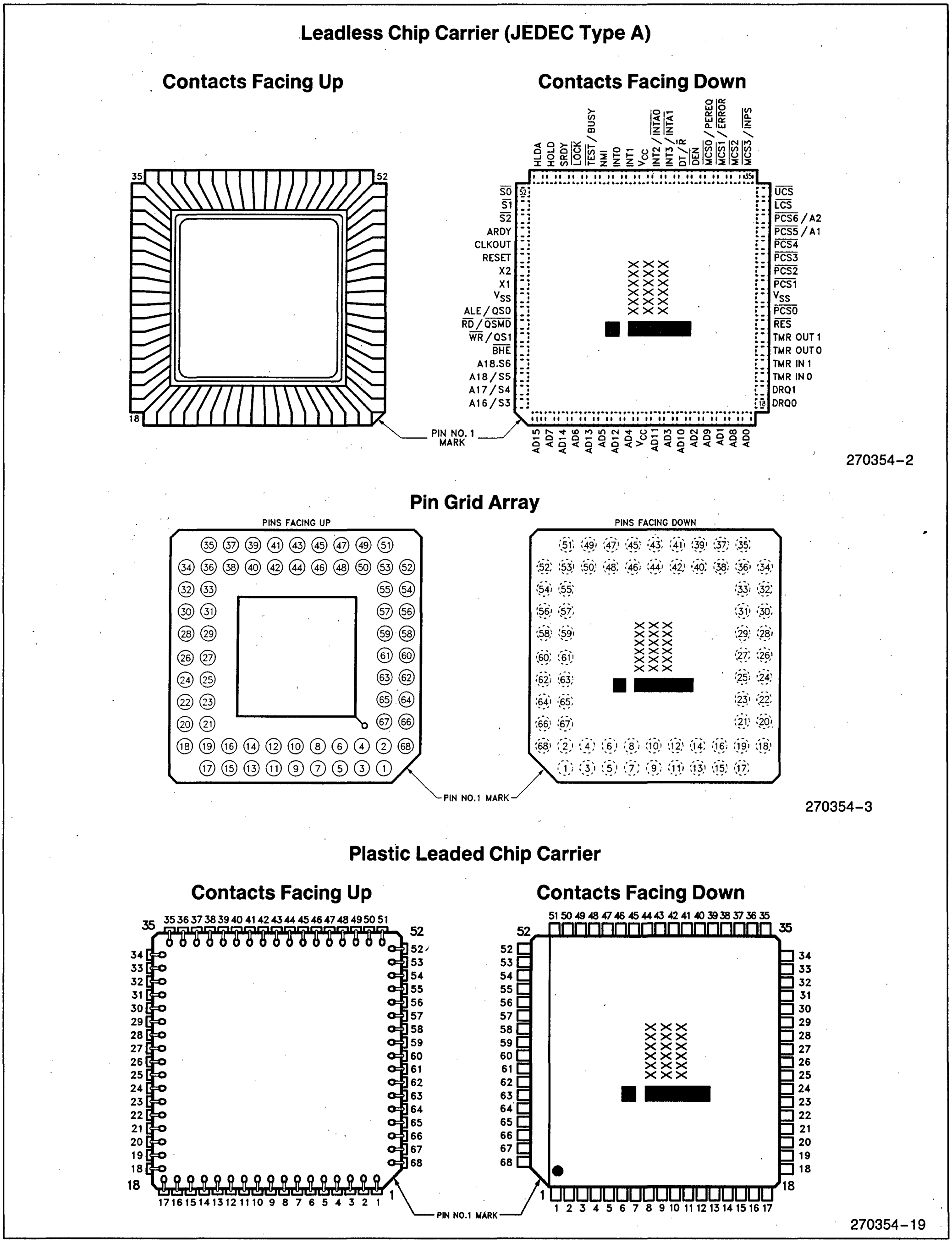
  (See Packaging Outlines and Dimensions, Order Number 231369)
- **Available in EXPRESS:**
  - **—Standard Temperature with Burn-In**
  - **—Extended Temperature Range (−40°C to +85°C)**

The Intel 80C186 is a CHMOS high integration microprocessor. It has features which are new to the 80186 family which include a DRAM refresh control unit, power-save mode and a direct numerics interface. When used in "compatible" mode, the 80C186 is 100% pin-for-pin compatible with the NMOS 80186 (except for 8087 applications). The "enhanced" mode of operation allows the full feature set of the 80C186 to be used. The 80C186 is upward compatible with 8086 and 8088 software and fully compatible with 80186 and 80188 software.



270354-1

Figure 1. 80C186 Block Diagram



Leadless Chip Carrier (JEDEC Type A)

Contacts Facing Up

Contacts Facing Down

270354-2

Pin Grid Array

270354-3

Plastic Leaded Chip Carrier

Contacts Facing Up

Contacts Facing Down

270354-19

Figure 2. 80C186 Pinout Diagrams

Table 1. 80C186 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}, V_{CC}	9, 43	I	**System Power:** +5 volt power supply.
V_{SS}, V_{SS}	26, 60	I	System Ground.
RESET	57	O	Reset Output indicates that the 80C186 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{\text{RES}}$ signal. Reset goes inactive 2 clockout periods after $\overline{\text{RES}}$ goes inactive. When tied to the $\overline{\text{TEST}}$/BUSY pin, Reset forces the 80C186 into enhanced mode.
X1, X2	59, 58	I	Crystal Inputs X1 and X2 provide external connections for a fundamental mode or third overtone parallel resonant crystal for the internal oscillator. X1 can connect to an external clock instead of a crystal. In this case, minimize the capacitance on X2 or drive X2 with complemented X1. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	Clock Output provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT. CLKOUT has sufficient MOS drive capabilities for the Numeric Processor Extension.
$\overline{\text{RES}}$	24	I	System Reset causes the 80C186 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80C186 clock. The 80C186 begins fetching instructions approximately 7 clock cycles after $\overline{\text{RES}}$ is returned HIGH. For proper initialization, V_{CC} must be within specifications and the clock signal must be stable for more than 4 clocks with $\overline{\text{RES}}$ held LOW. $\overline{\text{RES}}$ is internally synchronized. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{\text{RES}}$ generation via an RC network. When $\overline{\text{RES}}$ occurs, the 80C186 will drive the status lines to an inactive level for one clock, and then float them.
$\overline{\text{TEST}}$/BUSY	47	I	The $\overline{\text{TEST}}$ pin is sampled during and after reset to determine whether the 80C186 is to enter Compatible or Enhanced Mode. Enhanced Mode requires $\overline{\text{TEST}}$ to be HIGH on the rising edge of $\overline{\text{RES}}$ and LOW four clocks later. Any other combination will place the 80C186 in Compatible Mode. A weak internal pullup insures a HIGH state when the pin is not driven. $\overline{\text{TEST}}$—In Compatible Mode this pin is configured to operate as $\overline{\text{TEST}}$. This pin is examined by the WAIT instruction. If the $\overline{\text{TEST}}$ input is HIGH when WAIT execution begins, instruction execution will suspend. $\overline{\text{TEST}}$ will be resampled every five clocks until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80C186 is waiting for $\overline{\text{TEST}}$, interrupts will be serviced. BUSY—In Enhanced Mode, this pin is configured to operate as BUSY. The BUSY input is used to notify the 80C186 of Numerics Processor Extension activity. Floating point instructions executing in the 80C186 sample the BUSY pin to determine when the Numerics Processor is ready to accept a new command. BUSY is active HIGH.
TMR IN 0, TMR IN 1	20 21	I I	Timer Inputs are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0, TMR OUT 1	22 23	O O	Timer outputs are used to provide single pulse or continous waveform generation, depending upon the timer mode selected.

Table 1. 80C186 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>DRQ0
DRQ1</td><td>18
19</td><td>I
I</td><td>DMA Request is driven HIGH by an external device when it desires that a DMA channel (Channel 0 or 1) perform a transfer. These signals are active HIGH, level-triggered, and internally synchronized.</td></tr>
<tr><td>NMI</td><td>46</td><td>I</td><td>Non-Maskable Interrupt is an edge-triggered input which causes a type 2 interrupt. NMI is not maskable internally. A transition from a LOW to HIGH initiates the interrupt at the next instruction boundary. NMI is latched internally. An NMI duration of one clock or more will guarantee service. This input is internally synchronized.</td></tr>
<tr><td>INT0, INT1
INT2/$\overline{\text{INTA0}}$
INT3/$\overline{\text{INTA1}}$</td><td>45, 44
42
41</td><td>I
I/O
I/O</td><td>Maskable Interrupt Requests can be requested by activating one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured via software to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured via software to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When slave mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).</td></tr>
<tr><td>A19/S6,
A18/S5,
A17/S4,
A16/S3</td><td>65
66
67
68</td><td>O
O
O
O</td><td>Address Bus Outputs (16–19) and Bus Cycle Status (3–6) reflect the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below:
<table>
<tr><th></th><th>Low</th><th>High</th></tr>
<tr><td>S6</td><td>Processor Cycle</td><td>DMA Cycle</td></tr>
</table>
S3, S4, and S5 are defined as LOW during T_2–T_4.</td></tr>
<tr><td>AD15–AD0</td><td>10–17,
1–8</td><td>I/O</td><td>Address/Data Bus (0–15) signals constitute the time multiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH. A_0 is analogous to $\overline{\text{BHE}}$ for the lower byte of the data bus, pins D_7 through D_0. It is LOW during T_1 when a byte is to be transferred onto the lower portion of the bus in memory or I/O operations.</td></tr>
<tr><td>$\overline{\text{BHE}}$</td><td>64</td><td>O</td><td>The $\overline{\text{BHE}}$ (Bus High Enable) signal is analogous to A0 in that it is used to enable data on to the most significant half of the data bus, pins D15–D8. $\overline{\text{BHE}}$ will be LOW during T_1 when the upper byte is transferred and will remain LOW through T_3 AND T_W. $\overline{\text{BHE}}$ does not need to be latched. $\overline{\text{BHE}}$ will float during HOLD.
In Enhanced Mode, $\overline{\text{BHE}}$ will also be used to signify DRAM refresh cycles. A refresh cycle is indicated by $\overline{\text{BHE}}$ and A0 being HIGH.
<table>
<tr><th colspan="3">$\overline{\text{BHE}}$ and A0 Encodings</th></tr>
<tr><th>$\overline{\text{BHE}}$ Value</th><th>A0 Value</th><th>Function</th></tr>
<tr><td>0</td><td>0</td><td>Word Transfer</td></tr>
<tr><td>0</td><td>1</td><td>Byte Transfer on upper half of data bus (D15–D8)</td></tr>
<tr><td>1</td><td>0</td><td>Byte Transfer on lower half of data bus (D_7–D_0)</td></tr>
<tr><td>1</td><td>1</td><td>Refresh</td></tr>
</table></td></tr>
</table>

Table 1. 80C186 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>ALE/QS0</td><td>61</td><td>O</td><td>Address Latch Enable/Queue Status 0 is provided by the 80C186 to latch the address. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the standard 8086. The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 8086. Note that ALE is never floated.</td></tr>
<tr><td>$\overline{WR}$/QS1</td><td>63</td><td>O</td><td>Write Strobe/Queue Status 1 indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{WR}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during "HOLD." It is driven HIGH for one clock during Reset, and then floated. When the 80C186 is in queue status mode, the ALE/QS0 and $\overline{WR}$/QS1 pins provide information about processor/instruction queue interaction.
<table>
<tr><th>QS1</th><th>QS0</th><th>Queue Operation</th></tr>
<tr><td>0</td><td>0</td><td>No queue operation</td></tr>
<tr><td>0</td><td>1</td><td>First opcode byte fetched from the queue</td></tr>
<tr><td>1</td><td>1</td><td>Subsequent byte fetched from the queue</td></tr>
<tr><td>1</td><td>0</td><td>Empty the queue</td></tr>
</table></td></tr>
<tr><td>$\overline{RD}$/$\overline{QSMD}$</td><td>62</td><td>O</td><td>Read Strobe indicates that the 80C186 is performing a memory or I/O read cycle. $\overline{RD}$ is active LOW for T_2, T_3, and T_W of any read cycle. It is guaranteed not to go LOW in T_2 until after the Address Bus is floated. $\overline{RD}$ is active LOW, and floats during "HOLD". $\overline{RD}$ is driven HIGH for one clock during Reset, and then the output driver is floated. A weak internal pull-up mechanism of the $\overline{RD}$ line holds it HIGH when the line is not driven. During RESET the pin is sampled to determine whether the 80C186 should provide ALE, $\overline{WR}$ and $\overline{RD}$, or if the Queue-Status should be provided. $\overline{RD}$ should be connected to GND to provide Queue-Status data.</td></tr>
<tr><td>ARDY</td><td>55</td><td>I</td><td>Asynchronous Ready informs the 80C186 that the addressed memory space or I/O device will complete a data transfer. The ARDY input pin will accept an asynchronous input, and is active HIGH. Only the rising edge is internally synchronized by the 80C186. This means that the falling edge of ARDY must be synchronized to the 80C186 clock. If connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active to terminate a bus cycle. If unused, this line should be tied LOW to yield control to the SRDY pin.</td></tr>
<tr><td>SRDY</td><td>49</td><td>I</td><td>Synchronous Ready must be synchronized externally to the 80C186. The use of SRDY provides a relaxed system-timing specification on the Ready input. This is accomplished by eliminating the one-half clock cycle which is required for internally resolving the signal level when using the ARDY input. This line is active HIGH. If this line is connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active before a bus cycle is terminated. If unused, this line should be tied LOW to yield control to the ARDY pin.</td></tr>
</table>

Table 1. 80C186 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$\overline{\text{LOCK}}$</td><td>48</td><td>O</td><td>$\overline{\text{LOCK}}$ output indicates that other system bus masters are not to gain control of the system bus while $\overline{\text{LOCK}}$ is active LOW. The $\overline{\text{LOCK}}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of the instruction following the LOCK prefix. No prefetches will occur while $\overline{\text{LOCK}}$ is asserted. $\overline{\text{LOCK}}$ is active LOW, is driven HIGH for one clock during RESET, and then floated.</td></tr>
<tr><td>$\overline{\text{S0}}$, $\overline{\text{S1}}$, $\overline{\text{S2}}$</td><td>52–54</td><td>O</td><td>Bus cycle status $\overline{\text{S0}}$–$\overline{\text{S2}}$ are encoded to provide bus-transaction information:
<table>
<tr><th colspan="4">80C186 Bus Cycle Status Information</th></tr>
<tr><th>$\overline{\text{S2}}$</th><th>$\overline{\text{S1}}$</th><th>$\overline{\text{S0}}$</th><th>Bus Cycle Initiated</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Interrupt Acknowledge</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Read I/O</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Write I/O</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Halt</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Instruction Fetch</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Read Data from Memory</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Write Data to Memory</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Passive (no bus cycle)</td></tr>
</table>
The status pins float during HOLD/HLDA.
$\overline{\text{S2}}$ may be used as a logical M/$\overline{\text{IO}}$ indicator, and $\overline{\text{S1}}$ as a DT/$\overline{\text{R}}$ indicator.
The status lines are driven HIGH for one clock during Reset, and then floated until a bus cycle begins.</td></tr>
<tr><td>HOLD (input)
HLDA (output)</td><td>50
51</td><td>I
O</td><td>HOLD indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80C186 clock. The 80C186 will issue a HLDA (HIGH) in response to a HOLD request at the end of T_4 or T_i. Simultaneous with the issuance of HLDA, the 80C186 will float the local bus and control lines. After HOLD is detected as being LOW, the 80C186 will lower HLDA. When the 80C186 needs to run another bus cycle, it will again drive the local bus and control lines.
In Enhanced Mode, HLDA will go low when a DRAM refresh cycle is pending in the 80C186 and an external bus master has control of the bus. It will be up to the external master to relinquish the bus by lowering HOLD so that the 80C186 may execute the refresh cycle. Lowering HOLD for four clocks and returning HIGH will insure only one refresh cycle to the external master. HLDA will immediately go active after the refresh cycle has taken place.</td></tr>
<tr><td>$\overline{\text{UCS}}$</td><td>34</td><td>O</td><td>Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{\text{UCS}}$ is software programmable.
$\overline{\text{UCS}}$ and $\overline{\text{LCS}}$ are sampled upon the rising edge of $\overline{\text{RES}}$. If both pins are held low, the 80C186 will enter ONCE™ Mode. In ONCE Mode all pins assume a high impedance state and remain so until a subsequent RESET. $\overline{\text{UCS}}$ has a weak internal pullup for normal operation.</td></tr>
</table>

Table 1. 80C186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{\text{LCS}}$	33	O	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{\text{LCS}}$ is software programmable. $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$ are sampled upon the rising edge of $\overline{\text{RES}}$. If both pins are held low, the 80C186 will enter ONCE Mode. In ONCE Mode all pins assume a high impedance state and remain so until a subsequent RESET. $\overline{\text{UCS}}$ has a weak internal pullup for normal operation.
$\overline{\text{MCS0}}$/PEREQ $\overline{\text{MCS1}}$/$\overline{\text{ERROR}}$ $\overline{\text{MCS2}}$ $\overline{\text{MCS3}}$/$\overline{\text{NPS}}$	38 37 36 35	I/O I/O O O	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{\text{MCS0}}$–3 are software programmable. In Enhanced Mode, $\overline{\text{MCS0}}$ becomes a PEREQ input (Processor Extension Request). When connected to the Numerics Processor Extension, this input is used to signal the 80C186 when to make numeric data transfers to and from the NPX. $\overline{\text{MCS3}}$ becomes $\overline{\text{NPS}}$ (Numeric Processor Select) which may only be activated by communication to the Numerics Processor Extension. $\overline{\text{MCS1}}$ becomes $\overline{\text{ERROR}}$ in enhanced mode and is used to signal numerics coprocessor errors.
$\overline{\text{PCS0}}$ $\overline{\text{PCS1}}$–4	25 27, 28, 29, 30	O O	Peripheral Chip Select signals 0–4 are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{\text{PCS0}}$–4 are software programmable.
$\overline{\text{PCS5}}$/A1	31	O	Peripheral Chip Select 5 or Latched A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{\text{PCS5}}$ is software programmable. When programmed to provide latched. A1, rather than $\overline{\text{PCS5}}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{\text{PCS6}}$/A2	32	O	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{\text{PCS6}}$ is software programmable. When programmed to provide latched A2, rather than $\overline{\text{PCS6}}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{\text{R}}$	40	O	Data Transmit/Receive controls the direction of data flow through the external 8286/8287 data bus transceiver. When LOW, data is transferred to the 80C186. When HIGH the 80C186 places write data on the data bus.
$\overline{\text{DEN}}$	39	O	Data Enable is provided as an 8286/8287 data bus transceiver output enable. $\overline{\text{DEN}}$ is active LOW during each memory and I/O access. $\overline{\text{DEN}}$ is HIGH whenever DT/$\overline{\text{R}}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80C186. This architecture is common to the 8086, 8088, 80186 and 80286 microprocessor families as well. The 80C186 is a very high integration 16-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip. The 80C186 is object code compatible with the 8086/8088 microprocessors and adds 10 new instruction types to the existing 8086/8088 instruction set.

The 80C186 has two major modes of operation, Compatible and Enhanced. In Compatible Mode the 80C186 is completely compatible with NMOS 80186, with the exception of 8087 support. All pin functions, timings, and drive capabilities are identical. The Enhanced mode adds three new features to the system design. These are Power-Save control, Dynamic RAM refresh, and an asynchronous Numerics Co-processor interface.

80C186 BASE ARCHITECTURE

The 8086, 8088, 80186, and 80286 family all contain the same basic set of registers, instructions, and addressing modes. The 80C186 processor is upward compatible with the 8086, 8088, and 80286 CPUs.

Register Set

The 80C186 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

General Registers

Eight 16-bit general purpose registers may be used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

Segment Registers

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

Status and Control Registers

Two 16-bit special purpose registers record or alter certain aspects of the 80C186 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

Status Word Description

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80C186 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

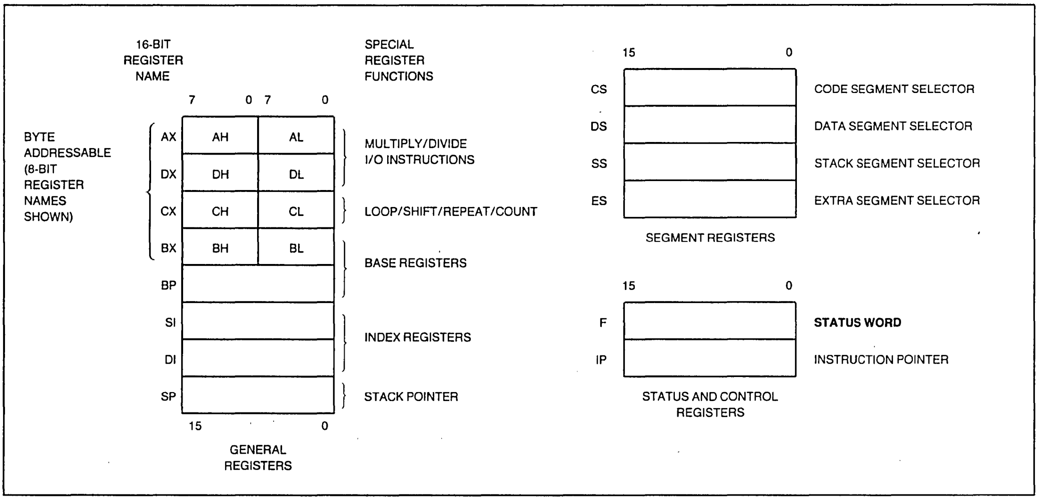


Figure 3a. 80C186 Register Set

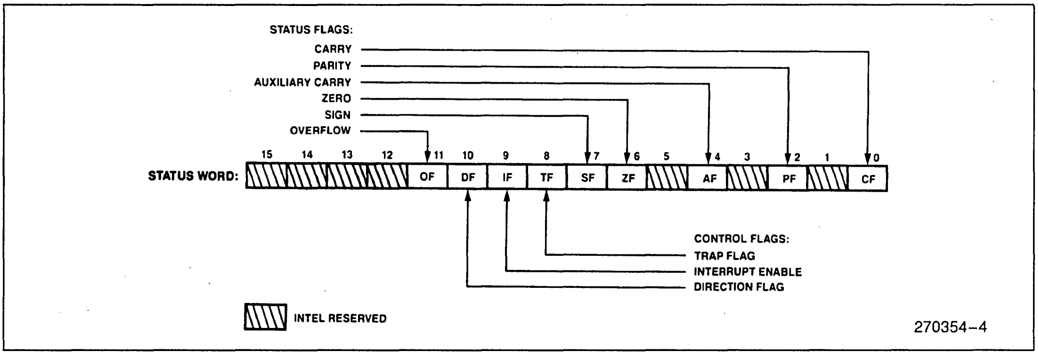


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80C186 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment register (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

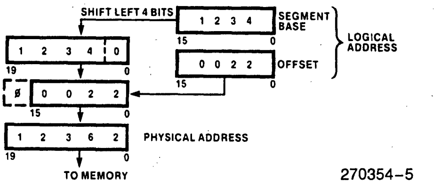
Figure 4. 80C186 Instruction Set

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign

JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4. 80C186 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

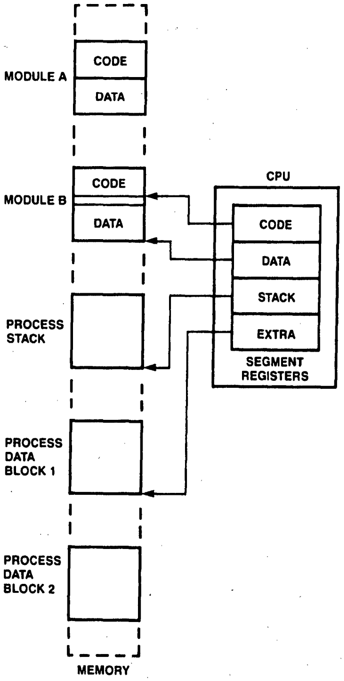


270354-5

Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.



270354-6

Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80C186 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an Index register.
- *Based indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80C186 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using a Numeric Data Coprocessor with the 80C186.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using a Numeric Data Coprocessor with the 80C186.)

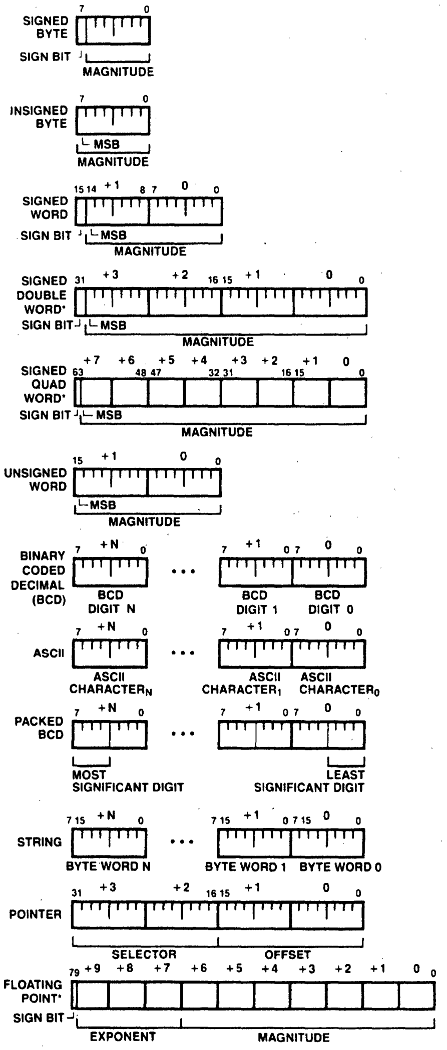
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80C186.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



NOTE:
*Supported by using a Numeric Data Coprocessor with the 80C186.

Figure 7. 80C186 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80C186 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80C186 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80C186 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INTO, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80C186 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80C186 Interrupt Vectors

Interrupt Name	Vector Type	Default Priority	Related Instructions
Divide Error Exception	0	*1	DIV, IDIV
Single Step Interrupt	1	12**	All
NMI	2	1	All
Breakpoint Interrupt	3	*1	INT
INTO Detected Overflow Exception	4	*1	INTO
Array Bounds Exception	5	*1	BOUND
Unused-Opcode Exception	6	*1	Undefined Opcodes
ESC Opcode Exception	7	*1***	ESC Opcodes
Timer 0 Interrupt	8	2A****	
Timer 1 Interrupt	18	2B****	
Timer 2 Interrupt	19	2C****	
Reserved	9	3	
DMA 0 Interrupt	10	4	
DMA 1 Interrupt	11	5	
INT0 Interrupt	12	6	
INT1 Interrupt	13	7	
INT2 Interrupt	14	8	
INT3 Interrupt	15	9	

NOTES:

*1. These are generated as the result of an instruction execution.

**2. This is handled as in the 8086.

****3. All three timers constitute one source of request to the interrupt controller. The Timer interrupts all have the same default priority level with respect to all other interrupt sources. However, they have a defined priority ordering amongst themselves. (Priority 2A is higher priority than 2B.) Each Timer interrupt has a separate vector type number.

4. Default priorities for the interrupt sources are used only if the user does not program each source into a unique priority level.

***5. An escape opcode will cause a trap if the 80C186 is in compatible mode or if the processor is in enhanced mode with the proper bit set in the peripheral control block relocation register.

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INTO DETECTED OVERFLOW EXCEPTION (TYPE4)

Generated during an INTO instruction if the OF bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H-DFH). In compatible mode operation, ESC opcodes will always generate this exception. In enhanced mode operation, the exception will be generated only if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80C186 provides maskable hardware interrupt request pins INT0-INT3. In addition, maskable interrupts may be generated by the 80C186 integrated DMA controller and the integrated timer unit. The vector types for these interrupts is shown in Table 4. Software enables these inputs by setting the interrupt flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80C186 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80C186 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80C186 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80C186 begins execution with the instruction at physical location FFFF0(H). $\overline{RES}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80C186 Initial Register State after RESET

Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

80C186 CLOCK GENERATOR

The 80C186 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The 80C186 oscillator circuit is designed to be used either with a parallel resonant fundamental or third-overtone mode crystal, depending upon the frequency range of the application as shown in Figure 8c. This is used as the time base for the 80C186. The crystal frequency chosen should be twice the required processor frequency. Use of an LC or RC circuit is not recommended.

The output of the oscillator is not directly available outside the 80C186. The two recommended crystal configurations are shown in Figure 8a. When used in third-overtone mode the tank circuit shown in Figure 8b is recommended for stable operation. The sum of the stray capacitances and loading capacitors should equal the values shown. It is advisable to limit stray capacitance between the X1 and X2 pins to less than 10 pF. While a fundamental-mode circuit will require approximately 1 ms for start-up, the third-overtone arrangement may require 1 ms to 3 ms to stabilize.

Alternately the oscillator pins may be driven from an external source in a configuration shown in Figure 8d or Figure 8e. The configuration shown in Figure 8f is not recommended.

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	40Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_1 (Load Capacitance):	20 pF ± 2 pF
Drive Level:	1 mW max

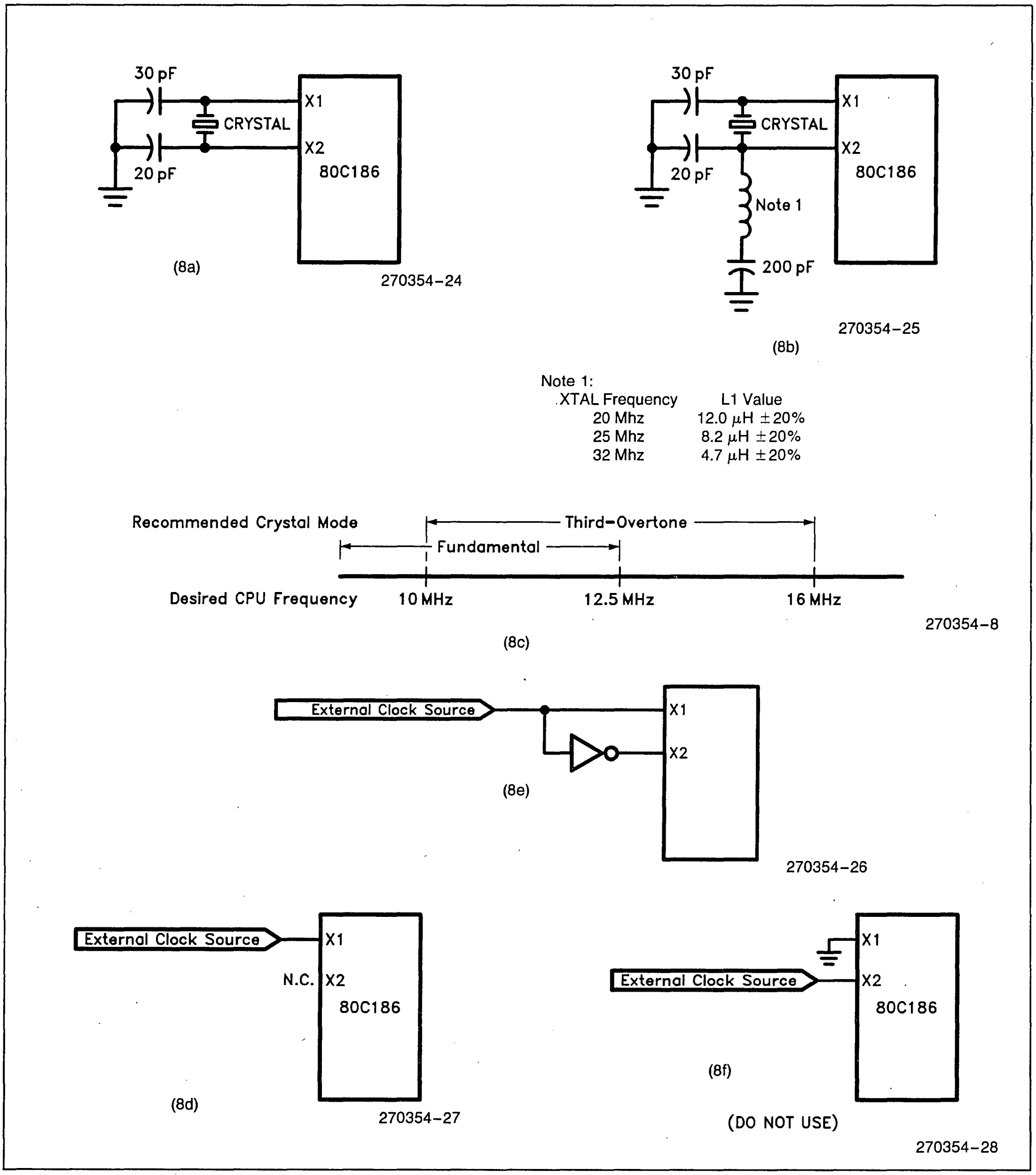
Clock Generator

The 80C186 clock generator provides the 50% duty cycle processor clock for the 80C186. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80C186. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80C186 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used. Full synchronization is performed only on the rising edge of ARDY, i.e., the falling edge of ARDY must be synchronized to the CLKOUT signal if it will occur during T_2, T_3, or T_W. High-to-LOW transitions of ARDY must be performed synchronously to the CPU clock.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.



Note 1:

XTAL Frequency	L1 Value
20 Mhz	12.0 µH ±20%
25 Mhz	8.2 µH ±20%
32 Mhz	4.7 µH ±20%

Figure 8. 80C186 Oscillator Configurations (see text)

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80C186, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The 80C186 provides both a $\overline{\text{RES}}$ input pin and a synchronized RESET pin for use with other system components. The $\overline{\text{RES}}$ input pin on the 80C186 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{\text{RES}}$ input of at least six clocks. RESET may be delayed up to two and one-half clocks behind $\overline{\text{RES}}$.

Multiple 80C186 processors may be synchronized through the $\overline{\text{RES}}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{\text{RES}}$ must satisfy a 25 ns setup time before the falling edge of the 80C186 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 15 ns setup time before the rising edge of the CLKOUT signal of all the processors.

LOCAL BUS CONTROLLER

The 80C186 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides control lines that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80C186 provides ALE, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ bus control signals. The $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals are used to strobe data from memory to the 80C186 or to strobe data from the 80C186 to memory. The ALE line provides a strobe to address latches for the multiplexed address/data bus. The 80C186 local bus controller does not provide a memory/$\overline{\text{IO}}$ signal. If this is required, the user will have to use the $\overline{\text{S2}}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80C186 generates two control signals to be connected to external transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{\text{R}}$ and $\overline{\text{DEN}}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{\text{DEN}}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{\text{R}}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80C186 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80C186 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. This requires external circuitry to arbitrate which external device will gain control of the bus from the 80C186 when there is more than one alternate local bus master. When the 80C186 relinquishes control of the local bus, it floats $\overline{\text{DEN}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{S0}}$–$\overline{\text{S2}}$, $\overline{\text{LOCK}}$, AD0–AD15, A16–A19, $\overline{\text{BHE}}$, and DT/$\overline{\text{R}}$ to allow another master to drive these lines directly.

The 80C186 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is the highest-priority activity request which the processor may receive: higher than instruction fetching or internal DMA cycles. However, if a DMA cycle is in progress, the 80C186 will complete the transfer before relinquishing the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency time can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd ad-

dress to an odd address. This is a total of 16 clocks or more, if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

Local Bus Controller and Reset

Upon receipt of a RESET pulse from the $\overline{\text{RES}}$ input, the local bus controller will perform the following action:

- Drive $\overline{\text{DEN}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ HIGH for one clock cycle, then float.

NOTE:
$\overline{\text{RD}}$ is also provided with an internal pull-up device to prevent the processor from inadvertently entering Queue Status mode during reset.

- Drive $\overline{\text{S0}}$–$\overline{\text{S2}}$ to the passive state (all HIGH) and then float.
- Drive $\overline{\text{LOCK}}$ HIGH and then float.
- Float AD0–15, A16–19, $\overline{\text{BHE}}$, DT/$\overline{\text{R}}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80C186 integrated peripherals are controlled via 16-bit registers contained within an internal 256-byte control block. This control block may be mapped into either memory or I/O space. Internal logic will recognize the address and respond to the bus cycle. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{\text{RD}}$, $\overline{\text{WR}}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{15-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80C186 CPU at any time. The location of any register contained within the 256-byte control block is determined by the current base address of the control block.

The control block base address is programmed via a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. The control block is effectively an internal chip select range and must abide by all the rules concerning chip selects (the chip select circuitry is discussed later in this data sheet). Any access to the 256 bytes of the control block activates an internal chip select. Other chip selects may overlap the control block only if they are programmed to zero wait states and ignore external ready. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space, whereas if the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into slave mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register is set to 20FFH. This causes the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

The integrated 80C186 peripherals operate semi-autonomously from the CPU. Access to them for the most part is via software read/write of the control block. Most of these registers can be both read and written. A few dedicated lines, such as interrupts and DMA request provide real-time communication between the CPU and peripherals as in a more conventional system utilizing discrete peripheral blocks. The overall interaction and function of the peripheral blocks has not substantially changed.

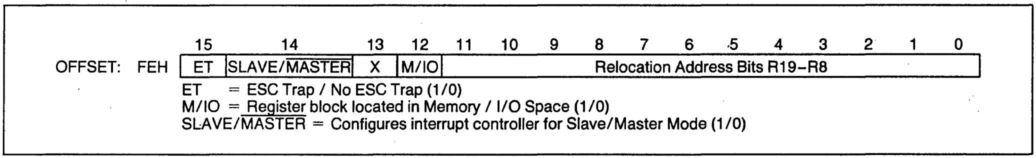
CHIP-SELECT/READY GENERATION LOGIC

The 80C186 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also povide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The 80C186 provides 6 memory chip select outputs for 3 address areas; upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address


15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
OFFSET: FEH
ET
SLAVE/$\overline{\text{MASTER}}$
X
M/$\overline{\text{IO}}$
Relocation Address Bits R19–R8

ET = ESC Trap / No ESC Trap (1/0)
M/$\overline{\text{IO}}$ = Register block located in Memory / I/O Space (1/0)
SLAVE/$\overline{\text{MASTER}}$ = Configures interrupt controller for Slave/Master Mode (1/0)

Figure 9. Relocation Register

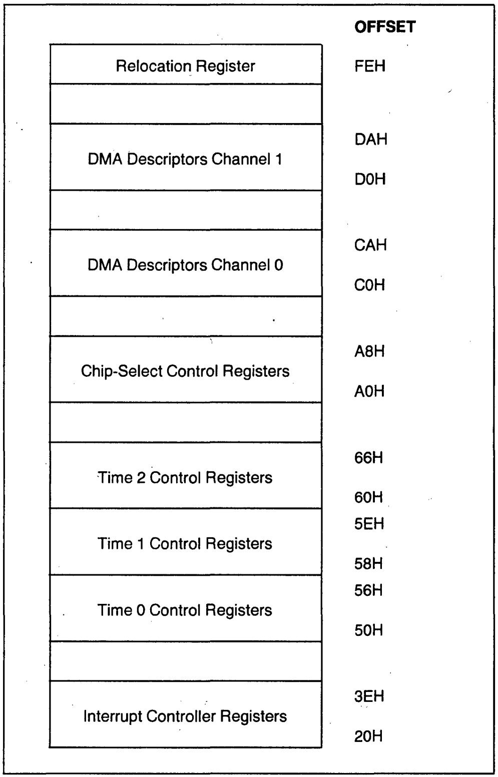


Figure 10. Internal Register Map

of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes, whereas 80C186 memory is arranged in words. This means that if, for example, 16 64K x 1 memories are used, the memory block size will be 128K, not 64K.

Upper Memory $\overline{\text{CS}}$

The 80C186 provides a chip select, called $\overline{\text{UCS}}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80C186 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0=R1=R2=0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

Any internally generated 20-bit address whose upper 16 bits are greater than or equal to UMCS (with bits 0–5 "0") will cause UCS to be activated. UMCS bits R2–R0 are used to specify READY mode for the area of memory defined by this chip-select register, as explained below.

Lower Memory $\overline{\text{CS}}$

The 80C186 provides a chip select for low memory called $\overline{\text{LCS}}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is also defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFFH	1K	0038H
007FFFH	2K	0078H
00FFFFH	4K	00F8H
01FFFFH	8K	01F8H
03FFFFH	16K	03F8H
07FFFFH	32K	07F8H
0FFFFFH	64K	0FF8H
1FFFFFH	128K	1FF8H
3FFFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12). This register is at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{\text{LCS}}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will cause $\overline{\text{LCS}}$ to be active. LMCS register bits R2–R0 are used to specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{\text{CS}}$

The 80C186 provides four $\overline{\text{MCS}}$ lines which are active within a user-locatable memory block. This block can be located within the 80C186 1M byte memory address space exclusive of the areas defined by $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{\text{MCS}}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. Thus, if the total block size is 32K, each chip select is active for 8K of memory with $\overline{\text{MCS0}}$ being active for the first range and $\overline{\text{MCS3}}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionally as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{\text{MCS}}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{\text{MCS}}$ lines will be active until both the MMCS and MPCS registers are accessed.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A0H	1	1	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 11. UMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A2H	0	0	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 12. LMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A8H	1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

Figure 13. MPCS Register

	15						9						3			0
OFFSET: A6H	U	U	U	U	U	U	U	1	1	1	1	1	1	R2	R1	R0
	A19						A13									

Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all mid-range chip selects. All devices in mid-range memory must use the same number of WAIT states.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{\text{LCS}}$ line was programmed, there would be an internal conflict between the $\overline{\text{LCS}}$ ready generation logic and the $\overline{\text{MCS}}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{\text{UCS}}$ ready generation logic. Since the $\overline{\text{LCS}}$ chip-select line does not become active until programmed, while the $\overline{\text{UCS}}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{\text{LCS}}$ range must not be programmed.

Peripheral Chip Selects

The 80C186 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. This base address may be located in either memory or I/O space.

Seven $\overline{\text{CS}}$ lines called $\overline{\text{PCS0}}$–6 are generated by the 80C186. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$ can also be programmed to provide latched address bits A1, A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0, A1 pins used for selecting internal registers of 8-bit peripheral chips. This scheme simplifies the hardware interface because the 8-bit registers of peripherals are simply treated as 16-bit registers located on even boundaries in I/O space or memory space where only the lower 8-bits of the register are significant: the upper 8-bits are "don't cares."

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). This register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

	15									6	5		3			0
OFFSET: A4H	U	U	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19									A10						

Figure 15. PACS Register

The user should program bits 15-6 to correspond to the desired peripheral base location. PACS bits 0-2 are used to specify READY mode for PCS0-PCS3.

Table 10. PCS Address Ranges

PCS Line	Active between Locations
PCS0	PBA —PBA+127
PCS1	PBA+128—PBA+255
PCS2	PBA+256—PBA+383
PCS3	PBA+384—PBA+511
PCS4	PBA+512—PBA+639
PCS5	PBA+640—PBA+767
PCS6	PBA+768—PBA+895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 13). This register is located at offset A8H in the internal control block. Bit 7 is used to select the function of PCS5 and PCS6, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space. 0 = Peripherals mapped into I/O space.
EX	0 = 5 PCS lines. A1, A2 provided. 1 = 7 PCS lines. A1, A2 are not provided.

MPCS bits 0-2 are used to specify READY mode for PCS4-PCS6 as outlined below.

READY Generation Logic

The 80C186 can generate a "READY" signal internally for each of the memory or peripheral CS lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0-3 wait states for all accesses to the area for which the chip select is active. In addition, the 80C186 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each CS line or group of lines generated by the 80C186. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). This means, for example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2-R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2-R0 of PACS set the PCS0-3 READY mode, R2-R0 of MPCS set the PCS4-6 READY mode.

Chip Select/Ready Logic and Reset

Upon reset, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the UCS line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to allow the maximum number of internal wait states in conjunction with external Ready consideration (i.e., UMCS resets to FFFBH).
- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the PCS lines will become active.

DMA CHANNELS

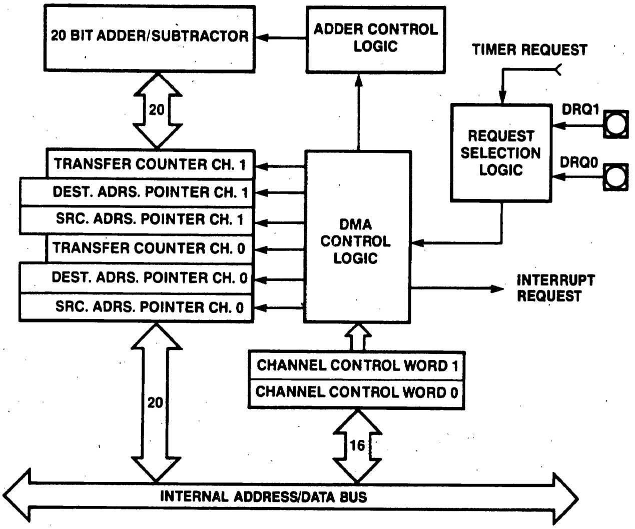
The 80C186 DMA controller provides two independent high-speed DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Data can be transferred either in bytes (8 bits) or in words (16 bits) to or from even or odd addresses. Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer (by one or two depending on byte or word transfers). Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data.

DMA Operation

Each channel has six registers in the control block which define each channel's specific operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit destination pointer (2 words), a 16-bit Transfer Counter, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte or word transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 17). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	COH	DOH



270354–9

Figure 16. DMA Unit Block Diagram

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M/$\overline{\text{IO}}$	DESTINATION DEC	INC	M/$\overline{\text{IO}}$	SOURCE DEC	INC	TC	INT	SYN		P	T D R Q	X	CHG/$\overline{\text{NOCHG}}$	ST/$\overline{\text{STOP}}$	$\overline{\text{B}}$/W

X = DON'T CARE.

Figure 17. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 80C186 DMA channel. This register specifies:

- the mode of synchronization;
- whether bytes or words will be transferred;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

$\overline{\text{B}}$/W: Byte/Word (0/1) Transfers.

ST/$\overline{\text{STOP}}$: Start/stop (1/0) Channel.

CHG/$\overline{\text{NOCHG}}$: Change/Do not change (1/0) ST/$\overline{\text{STOP}}$ bit. If this bit is set when writing to the control word, the ST/$\overline{\text{STOP}}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{\text{STOP}}$ bit will not be altered. This bit is not stored; it will always be a 0 on read.

INT: Enable Interrupts to CPU on Transfer Count termination.

TC: If set, DMA will terminate when the contents of the Transfer Count register reach zero. The ST/$\overline{\text{STOP}}$ bit will also be reset at this point if TC is set. If this bit is cleared, the DMA unit will decrement the transfer count register for each DMA cycle, but the DMA transfer will not stop when the contents of the TC register reach zero.

SYN 00 No synchronization.

NOTE:

When unsynchronized transfers are specified, the TC bit will be ignored and the ST bit will be cleared upon the transfer count reaching zero, stopping the channel.

(2 bits) 01 Source synchronization.

10 Destination synchronization.

11 Unused.

SOURCE:INC Increment source pointer by 1 or 2 (depends on $\overline{\text{B}}$/W) after each transfer.

M/$\overline{\text{IO}}$ Source pointer is in M/IO space (1/0).

DEC Decrement source pointer by 1 or 2 (depends on $\overline{\text{B}}$/W) after each transfer.

DEST: INC Increment destination pointer by 1 or 2 ($\overline{\text{B}}$/W) after each transfer.

M/$\overline{\text{IO}}$ Destination pointer is in M/IO space (1/0).

DEC Decrement destination pointer by 1 or 2 (depending on $\overline{\text{B}}$/W) after each transfer.

P Channel priority—relative to other channel.

0 low priority.

1 high priority.

Channels will alternate cycles if both set at same priority level.

TDRQ 0: Disable DMA requests from timer 2.

1: Enable DMA requests from timer 2.

Bit 3 Bit 3 is not used.

If both INC and DEC are specified for the same pointer, the pointer will remain constant after each cycle.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18). These pointers may be individually incremented or decremented after each transfer. If word transfers are performed the pointer is incremented or decremented by two. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers. Higher transfer rates can be obtained if all word transfers are performed to even addresses, since this will allow data to be accessed in a single memory access.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). This register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or if unsynchronized transfers are programmed, however, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). No prefetching occurs when destination synchronization is performed, however. Data will not be fetched from the source address until the destination device signals that it is ready to receive it. When destination synchronized transfers are requested, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another DMA cycle will begin after two processor clocks. This is done to allow the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. These lead to the maximum DMA transfer rates shown in Table 14.

Table 14. Maximum DMA Transfer Rates at 16 MHz

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	4.0MBytes/sec	4.0MBytes/sec
Source Synch	4.0MBytes/sec	4.0MBytes/sec
Destination Synch	2.7MBytes/sec	3.2MBytes/sec

HIGHER REGISTER ADDRESS	XXX	XXX	XXX	A19–A16
LOWER REGISTER ADDRESS	A15–A12	A11–A8	A7–A4	A3–A0
	15			0

XXX = DON'T CARE

Figure 18. DMA Memory Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed such that one channel is always given priority over the other, or they may be programmed such as to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses to odd memory locations; however, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/STOP bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also have been generated. Therefore the source and destination transfer pointers, and the transfer count register (if used) must be programmed before this bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/STOP bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

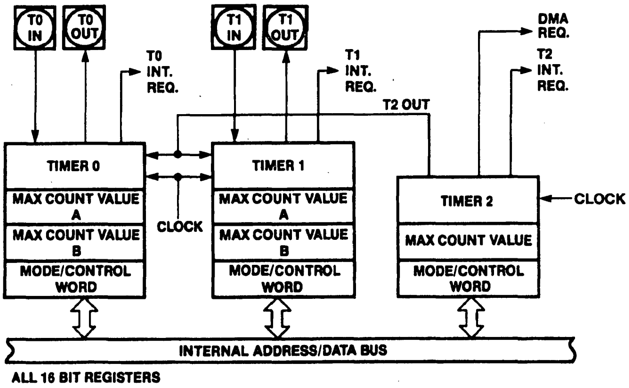
DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The Start/Stop bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80C186 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, this third timer can be used as a prescaler to the other two, or as a DMA request source.



270354–10

Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the internal peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values programmed by the user. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 1 clock after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate. Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

15	14	13	12	11		5	4	3	2	1	0
EN	$\overline{\text{INH}}$	INT	RIU	0	...	MC	RTG	P	EXT	ALT	CONT

Figure 20. Timer Mode/Control Register

ALT:

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT:

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If COUNT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

EXT:

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80C186 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

P:

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

RTG:

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80C186 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

EN:

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transistions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

$\overline{\text{INH}}$:

The inhibit bit allows for selective updating of the enable (EN) bit. If $\overline{\text{INH}}$ is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If $\overline{\text{INH}}$ is a zero during the write, the EN bit will be unaffected by the operation. This bit is not stored; it will always be a 0 on a read.

INT:

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller).

MC:

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set

regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts.

Programmer intervention is required to clear this bit.

RIU:

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The current contents of this register may be read or written by the processor at any time. If the register is written into while the timer is counting,the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. The condition which causes a timer to reset is equivalent between the current count value and the max count being used. This means that if the count is changed to be above the max count value, or if the max count value is changed to be below the current value, the timer will not reset to zero, but rather will count to its maximum value, "wrap around" to zero, then count until the max count is reached.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- All SEL (Select) bits are reset to zero. This selects MAX COUNT register A, resulting in the Timer Out pins going HIGH upon RESET.

INTERRUPT CONTROLLER

The 80C186 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80C186 interrupt controller has its own control register that set the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may themselves be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The 80C186 has a special slave mode in which the internal interrupt controller acts as a slave to an external master. The controller is programmed into this mode by setting bit 14 in the peripheral control block relocation register. (See Slave Mode section.)

MASTER MODE OPERATION

Interrupt Controller External Interface

For external interrupt sources, five dedicated pins are provided. One of these pins is dedicated to NMI, non-maskable interrupt. This is typically used for power-fail interrupts, etc. The other four pins may function either as four interrupt input lines with internally generated interrupt vectors, as an interrupt line and an interrupt acknowledge line (called the "cascade mode") along with two other input lines with internally generated interrupt vectors, or as two interrupt input lines and two dedicated interrupt acknowledge output lines. When the interrupt lines are configured in cascade mode, the 80C186 interrupt controller will not generate internal interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80C186 on the second cycle. The capability to interface to external 82C59A programmable interrupt controllers is thus provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in master mode are similar to the 82C59A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt. pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

Fully Nested Mode

When in the fully nested mode four pins are used as direct interrupt requests as in Figure 22. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled without being themselves interrupted by lower-priority interrupts. Since interrupts are enabled, higher-priority interrupts will be serviced.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is issued at the end of the service routine just before the issuance of the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

Cascade Mode

The 80C186 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 23. INT0 is an interrupt input interfaced to an 82C59A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 82C59As. Three levels of priority are created, requiring priority resolution in the 80C186 interrupt controller, the master 82C59As, and the slave 82C59As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

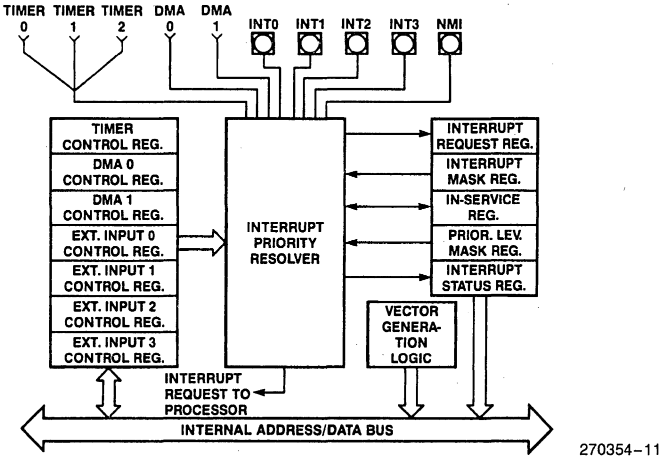


Figure 21. Interrupt Controller Block Diagram

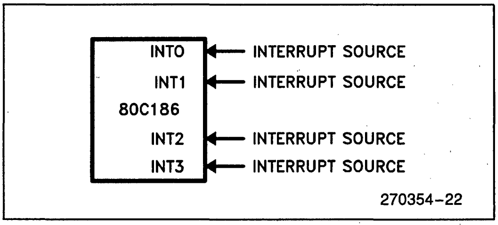


Figure 22. Fully Nested (Direct) Mode Interrupt Controller Connections

Special Fully Nested Mode

This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 82C59A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80C186 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80C186 controller until the 80C186 in-service bit is reset. In special fully nested mode, the 80C186 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80C186 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the external master's IS register is required to determine if there is more than one bit set. If so, the IS bit in the 80C186 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 32). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80C186 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Master Mode Features

Programmable Priority

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, it allows other requests to be serviced.

End-of-Interrupt Command

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

Trigger Mode

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the

80C186 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

Interrupt Vectoring

The 80C186 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 24. It contains 15 registers. All registers can both be read or written unless specified otherwise.

In-Service Register

This register can be read from or written into. The format is shown in Figure 25. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command issued by the CPU.

Interrupt Request Register

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 25. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits show exactly when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Writes to the interrupt request register will affect the D0 and D1 interrupt request bits. Setting either bit will cause the corresponding interrupt request while clearing either bit will remove the corresponding interrupt request. All other bits in the register are read-only.

Mask Register

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 25. A one in a bit position corre-

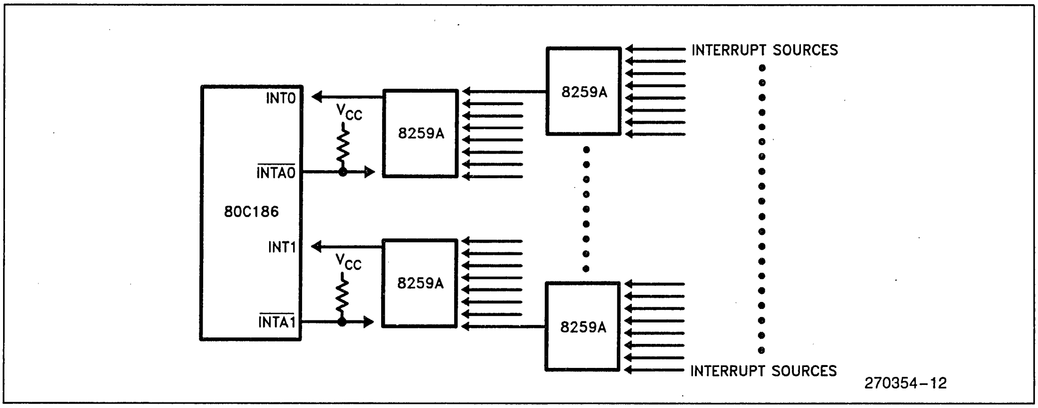


Figure 23. Cascade and Special Fully Nested Mode Interrupt Controller Connections

sponding to a particular source serves to mask the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 24. Interrupt Controller Registers (Master Mode)

Priority Mask Register

This register is used to mask all interrupts below particular interrupt priority levels. The format of this register is shown in Figure 26. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so no interrupts are masked due to priority number.

Interrupt Status Register

This register contains general interrupt controller status information. The format of this register is shown in Figure 27. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. The purpose of this bit is to allow prompt service of all non-maskable interrupts. This bit may also be set by the programmer.

IRTx: These three bits represent the individual timer interrupt request bits. These bits are used to differentiate the timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt request. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

15	14				10	9	8	7	6	5	4	3	2	1	0
0	0	•	•	•	0	0	0	13	12	I1	I0	D1	D0	0	TMR

Figure 25. In-Service, Interrupt Request, and Mask Register Formats

15	14											3	2	1	0
0	0	•	•	•	•	•	•	•	•	•		0	PRM2	PRM1	PRM0

Figure 26. Priority Mask Register Format

15	14						7	6	5	4	3	2	1	0
DHLT	0	•	•	•	•	•	0	0	0	0	0	IRT2	IRT1	IRT0

Figure 27. Interrupt Status Register Format (Master Mode)

Timer, DMA 0, 1; Control Register

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 28. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 Control Registers

These registers are the control words for the four external input pins. Figure 29 shows the format of the INT0 and INT1 Control registers; Figure 30 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest Priority = 000, Lowest Priority = 111

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is preceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct

SFNM: Special fully nested mode bit, 1 = SFNM

EOI Register

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 30. It initiates an EOI command when written to by the 80C186 CPU.

The bits in the EOI register are encoded as follows:

S_x: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10.

NOTE:

To reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

NSPEC/SPEC: A bit that determines the type of EOI command. Nonspecific = 1, Specific = 0.

15	14									4	3	2	1	0
0	0	•	•	•	•	•	•	•	•	0	MSK	PR2	PR1	PRO

Figure 28. Timer/DMA Control Registers Formats

15	14						7	6	5	4	3	2	1	0
0	0	•	•	•	•	•	0	SFNM	C	LTM	MSK	PR2	PR1	PRO

Figure 29. INT0/INT1 Control Register Formats

15	14								5	4	3	2	1	0
0	0	•	•	•	•	•	•	•	0	LTM	MSK	PR2	PR1	PRO

Figure 30. INT2/INT3 Control Register Formats

Poll and Poll Status Registers

These registers contain polling information. The format of these registers is shown in Figure 32. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_X: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

SLAVE MODE OPERATION

When slave mode is used, the internal 80C186 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80C186 resources will be monitored by the internal interrupt controller, while the external controller functions as the system master interrupt controller.

Upon reset, the 80C186 will be in master mode. To provide for slave mode operation bit 14 of the relocation register should be set.

Because of pin limitations caused by the need to interface to an external 82C59A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80C186 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

In slave mode each peripheral must be assigned a unique priority to ensure proper interrupt controller operation. Therefore, it is the programmer's responsibility to assign correct priorities and initialize interrupt control registers before enabling interrupts.

Slave Mode External Interface

The configuration of the 80C186 with respect to an external 82C59A master is shown in Figure 33. The INT0 (Pin 45) input is used as the 80C186 CPU interrupt input. INT3 (Pin 41) functions as an output to send the 80C186 slave-interrupt-request to one of the 8 master-PIC-inputs.

15	14	13		5	4	3	2	1	0
SPEC/ NSPEC	0	0	• • • • • • •	0	S4	S3	S2	S1	S0

Figure 31. EOI Register Format

15	14	13		5	4	3	2	1	0
INT REQ	0	0	• • • • • • •	0	S4	S3	S2	S1	S0

Figure 32. Poll and Poll Status Register Format

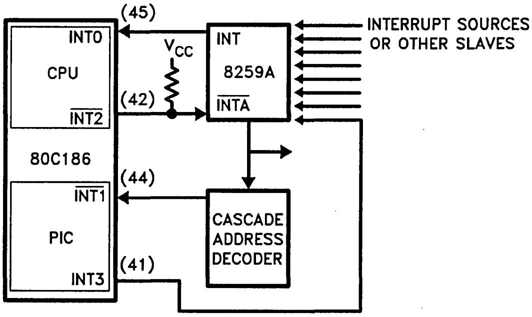


Figure 33. Slave Mode Interrupt Controller Connections

Correct master-slave interface requires decoding of the slave addresses (CAS0-2). Slave 82C59As do this internally. Because of pin limitations, the 80C186 slave address will have to be decoded externally. $\overline{\text{INT1}}$ (Pin 44) is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{\text{INT2}}$ (Pin 42) is used as an acknowledge output, suitable to drive the $\overline{\text{INTA}}$ input of an 82C59A.

Interrupt Nesting

Slave mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the Slave Mode

Vector generation in slave mode is exactly like that of an 82C59A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In slave mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the Slave Mode

All control and command registers are located inside the internal peripheral control block. Figure 34 shows the offsets of these registers.

End-of-Interrupt Register

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 35. It initiates an EOI command when written by the 80C186 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

In-Service Register

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 36. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

Interrupt Request Register

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 36. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request. As in master mode, D0 and D1 are read/write; all other bits are read only.

Mask Register

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 36. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

Control Registers

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 37. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT-REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 34. Interrupt Controller Registers (Slave Mode)

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	L2	L1	L0

Figure 35. Specific EOI Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 36. In-Service, Interrupt Request, and Mask Register Format

Interrupt Vector Register

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 38. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

Priority-Level Mask Register

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

Interrupt Status Register

This register is defined as in master mode except that DHLT is not implemented (see Figure 27).

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to master mode.

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	MSK	PR2	PR1	PR0

Figure 37. Control Word Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	t4	t3	t2	t1	t0	0	0	0

Figure 38. Interrupt Vector Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	m2	m1	m0

Figure 39. Priority Level Mask Register

Enhanced Mode Operation

In Compatible Mode the 80C186 operates with all the features of the NMOS 80186, with the exception of 8087 support (i.e. no numeric coprocessing is possible in Compatible Mode). Queue-Status information is still available for design purposes other than 8087 support.

All the Enhanced Mode features are completely masked when in Compatible Mode. A write to any of the Enhanced Mode registers will have no effect, while a read will not return any valid data.

In Enhanced Mode, the 80C186 will operate with Power-Save, DRAM refresh, and numerics coprocessor support in addition to all the Compatible Mode features.

Entering Enhanced Mode

If connected to a numerics coprocessor, this mode will be invoked automatically. Without a NPX, this mode can be entered by tying the RESET output signal from the 80C186 to the TEST/BUSY input.

Queue-Status Mode

The queue-status mode is entered by strapping the $\overline{\text{RD}}$ pin low. $\overline{\text{RD}}$ is sampled at RESET and if LOW, the 80C186 will reconfigure the ALE and $\overline{\text{WR}}$ pins to be QS0 and QS1 respectively. This mode is available on the 80C186 in both Compatible and Enhanced Modes and is identical to the NMOS 80186.

DRAM Refresh Control Unit Description

The Refresh Control Unit (RCU) automatically generates DRAM refresh bus cycles. The RCU operates only in Enhanced Mode. After a programmable period of time, the RCU generates a memory read request to the BIU. If the address generated during a refresh bus cycle is within the range of a properly programmed chip select, that chip select will be activated when the BIU executes the refresh bus cycle. The ready logic and wait states programmed for that region will also be in force. If no chip select is activated, then external ready is automatically required to terminate the refresh bus cycle.

If the HLDA pin is active when a DRAM refresh request is generated (indicating a bus hold condition), then the 80C186 will deactivate the HLDA pin in order to perform a refresh cycle. The circuit external to the 80C186 must remove the HOLD signal in order to execute the refresh cycle. The sequence of HLDA going inactive while HOLD is being held active can be used to signal a pending refresh request.

All registers controlling DRAM refresh may be read and written in Enhanced Mode. When the processor is operating in Compatible Mode, they are deselected and are therefore inaccessible. Some fields of these registers cannot be written and are always read as zeros.

DRAM Refresh Addresses

The address generated during a refresh cycle is determined by the contents of the MDRAM register (see Figure 40) and the contents of a 9-bit counter. Figure 41 illustrates the origin of each bit.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MDRAM: Offset E0H	M6	M5	M4	M3	M2	M1	M0	0	0	0	0	0	0	0	0	0

Bits 0–8: Reserved, read back as 0.

Bits 9–15: M0–M6, are address bits A13–A19 of the 20-bit memory refresh address. These bits should correspond to the chip select address to be activated for the DRAM partition. These bits are set to 0 on RESET.

Figure 40. Memory Partition Register

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
M6	M5	M4	M3	M2	M1	M0	0	0	0	CA8	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0	1

M6–M0: Bits defined by MDRAM Register

CA8–CA0: Bits defined by refresh address counter

Figure 41. Addresses Generated by RCU

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CDRAM: Offset E2H	0	0	0	0	0	0	0	C8	C7	C6	C5	C4	C3	C2	C1	C0

Bits 0–8: C0–C8, clock divisor register, holds the number of CLKOUT cycles between each refresh request.

Bits 9–15: Reserved, read back as 0.

Figure 42. Clock Pre-Scaler Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EDRAM: Offset E4H	E	0	0	0	0	0	0	T8	T7	T6	T5	T4	T3	T2	T1	T0

Bits 0–8: T0–T8, refresh clock counter outputs. Read only.

Bits 9–14: Reserved, read back as 0.

Bit 15: Enable RCU, set to 0 on RESET.

Figure 43. Enable RCU Register

Refresh Control Unit Programming and Operation

After programming the MDRAM and the CDRAM registers (Figures 40 and 42), the RCU is enabled by setting the "E" bit in the EDRAM register (Figure 43). The clock counter (T0–T8 of EDRAM) will be loaded from C0–C8 of CDRAM during T_3 of instruction cycle that sets the "E" bit. The clock counter is then decremented at each subsequent CLKOUT.

A refresh is requested when the value of the counter has reached 1 and the counter is reloaded from CDRAM. In order to avoid missing refresh requests, the value in the CDRAM register should always be at least 18 (12H). Clearing the "E" bit at anytime will clear the counter and stop refresh requests, but will not reset the refresh address counter.

POWER-SAVE CONTROL

Power Save Operation

The 80C186, when in Enhanced Mode, can enter a power saving state by internally dividing the clock-in frequency by a programmable factor. This divided frequency is also available at the CLKOUT pin. The PDCON register contains the two-bit fields for selecting the clock division factor and the enable bit.

All internal logic, including the Refresh Control Unit and the timers, will have their clocks slowed down by the division factor. To maintain a real time count or a fixed DRAM refresh rate, these peripherals must be re-programmed when entering and leaving the power-save mode.

The power-save mode is exited whenever an interrupt is processed by automatically resetting the enable bit. If the power-save mode is to be re-entered after serving the interrupt, the enable bit will need to be reset in software before returning from the interrupt routine.

The internal clocks of the 80C186 will begin to be divided during the T_3 state of the instruction cycle that sets the enable bit. Clearing the enable bit will restore full speed in the T_3 state of that instruction.

At no time should the internal clock frequency be allowed to fall below 0.5 MHz. This is the minimum operational frequency of the 80C186. For example, an 80C186 running with a 12 MHz crystal (6 MHz CLOCKOUT) should never have a clock divisor greater than eight.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDCON: Offset F0H	E	0	0	0	0	0	0	0	0	0	0	0	0	0	F1	F0

Bits 0–1: Clock Divisor Select

F1	F0	Division Factor
0	0	divide by 1
0	1	divide by 4
1	0	divide by 8
1	1	divide by 16

Bits 2–14: Reserved, read back as zero.

Bit 15: Enable Power Save Mode. Set to zero on RESET.

Figure 44. Power-Save Control Register

Numeric Coprocessor (NPX) Extension

Three of the mid-range memory chip selects are redefined according to Table 16 when using the numerics coprocessor extension. The fourth chip select, $\overline{\text{MCS2}}$ functions as in compatible mode, and may be programmed for activity with ready logic and wait states accordingly. As in compatible mode, $\overline{\text{MCS2}}$ will function for one-fourth a programmed block size.

Table 16. $\overline{\text{MCS}}$ Assignments

Compatible Mode	Enhanced Mode	
$\overline{\text{MCS0}}$	PEREQ	Processor Extension Request
$\overline{\text{MCS1}}$	$\overline{\text{ERROR}}$	NPX Error
$\overline{\text{MCS2}}$	$\overline{\text{MCS2}}$	Mid-Range Chip Select
$\overline{\text{MCS3}}$	$\overline{\text{NPS}}$	Numeric Processor Select

Four port addresses are assigned to the NPX for 16-bit reads and writes by the 80C186. Table 17 shows the port definitions. These ports are not accessible by using the 80C186 I/O instructions. However, numerics operations will cause a $\overline{\text{PCS}}$ line to be activated if it is properly programmed for this I/O range.

Table 17. Numerics Coprocessor I/O Port Assignments

I/O Address	Read Definition	Write Definition
00F8H	Status/Control	Opcode
00FAH	Data	Data
00FCH	reserved	CS:IP, DS:EA
00FEH	Opcode Status	reserved

"ONCE™" Test Mode

To facilitate testing and inspection of devices when fixed into a target system, the 80C186 has a test mode available which allows all pins to be placed in a high-impedance state. "ONCE" stands for "ON Circuit Emulation". When placed in this mode, the 80C186 will put all pins in the high-impedance state until RESET.

The ONCE mode is selected by tying the $\overline{\text{UCS}}$ and the $\overline{\text{LCS}}$ LOW during RESET. These pins are sampled on the low-to-high transition of the $\overline{\text{RES}}$ pin. The $\overline{\text{UCS}}$ and the $\overline{\text{LCS}}$ pins have weak internal pull-up resistors similar to the $\overline{\text{RD}}$ and $\overline{\text{TEST}}$/BUSY pins to guarantee proper normal operation.

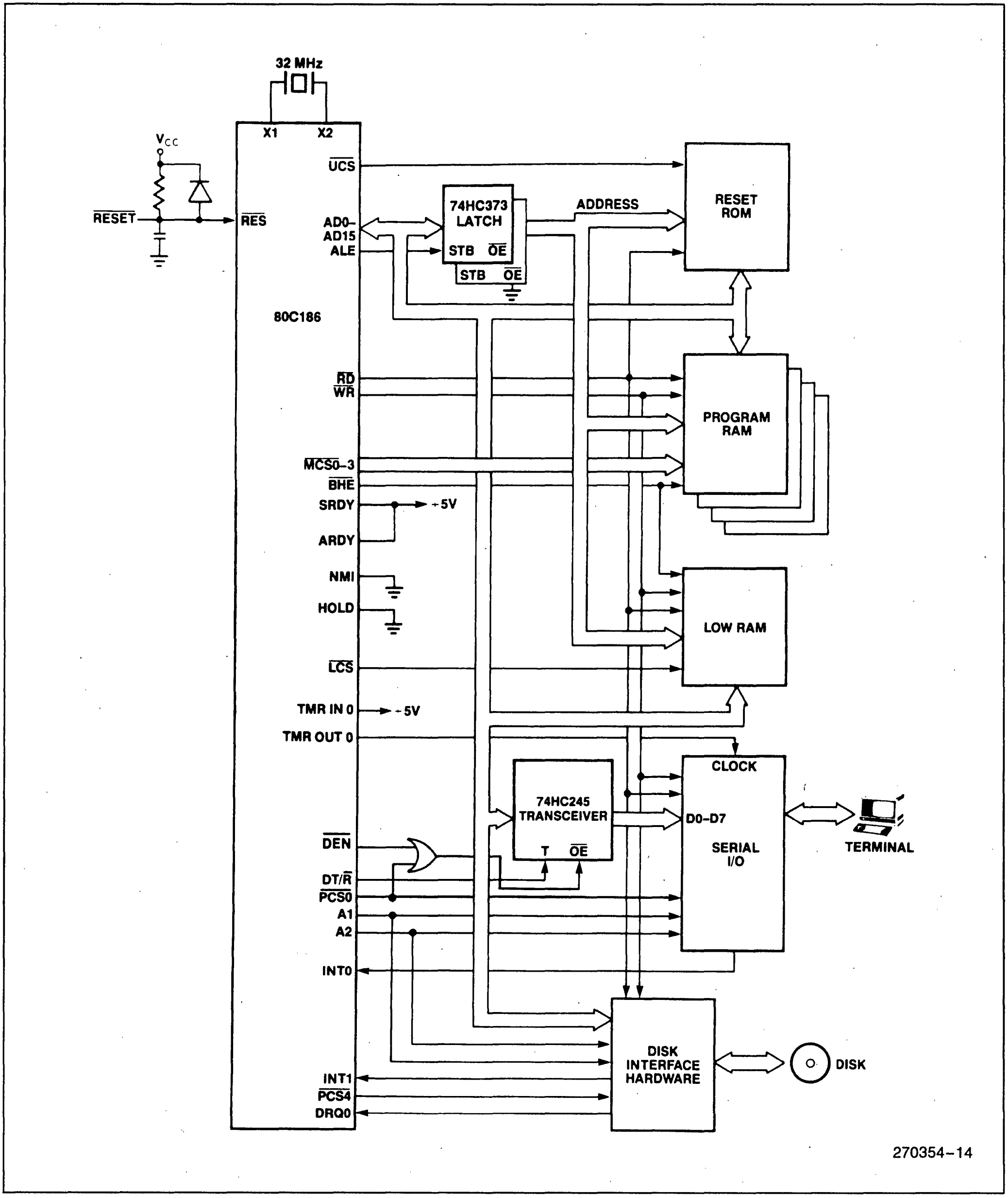


Figure 45. Typical 80C186 Computer

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ....0°C to +70°C
Storage Temperature .... −65°C to +150°C
Voltage on Any Pin with
Respect to Ground .... −1.0V to +7.0V
Package Power Dissipation 3W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

D.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ± 5% at 16 MHz

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.2 V_{CC} − 0.3	V	
V_{IH}	Input High Voltage (All except X1 and $\overline{RES}$)	0.2 V_{CC} + 0.9	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.5 mA (S0, 1, 2) I_{OL} = 2.0 mA (others)
V_{OH}	Output High Voltage	2.4	V_{CC}	V	I_{OH} = −2.4 mA @ 2.4V
		0.8 V_{CC}	V_{CC}	V	I_{OH} = −200 μA @ 0.8 V_{CC}
I_{CC}	Power Supply Current		150	mA	@ 12.5 MHz, 0°C V_{CC} = 5.5V
I_{PS}	Power Save Current	10 mA per MHz + 20		mA	Typical @25°C, V_{CC} = 5.0V
I_{LI}	Input Leakage Current		±10	μA	0.45V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}(1)
V_{CLO}	Clock Output Low		0.5	V	I_{CLO} = 4.0 mA
V_{CHO}	Clock Output High	0.8 V_{CC}		V	I_{CHO} = −500 μA
V_{CLI}	Clock Input Low Voltage (X1)	−0.5	0.6	V	
V_{CHI}	Clock Input High Voltage (X1)	3.9	V_{CC} + 0.5	V	
C_{IN}	Input Capacitance		10	pF	@ 1 MHz(2)
C_{IO}	I/O Capacitance		20	pF	@ 1 MHz(2)

NOTES:
1. Pins being floated during HOLD or by invoking the ONCE Mode.
2. Characterization conditions are a) Frequency = 1 MHz; b) Unmeasured pins at GND; c) V_{IN} at + 5.0V or 0.45V. This parameter is not tested.

PIN TIMINGS

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at 16 MHz

All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
Input V_{IL} = 0.45V and V_{IH} = 2.4V for A.C. tests.

Symbol	Parameter	80C186-10		80C186-12		80C186-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C186 TIMING REQUIREMENTS									
T_{DVCL}	Data In Setup (A/D)	15		15		10		ns	
T_{CLDX}	Data In Hold (A/D)	5		5		5		ns	
T_{ARYCH}	ARDY Resolution Transition Setup Time(1)	15		15		15		ns	
T_{ARYLCL}	Asynchronous Ready (ARDY) Setup Time	25		25		25		ns	
T_{CLARX}	ARDY Active Hold Time	15		15		15		ns	
T_{ARYCHL}	ARDY Inactive Hold Time	15		15		15		ns	
T_{SRYCL}	Synchronous Ready (SRDY) Transition Setup Time(1)	15		15		15		ns	
T_{CLSRY}	SRDY Transition Hold Time	15		15		15		ns	
T_{HVCL}	HOLD Setup(1)	15		15		15		ns	
T_{INVCH}	INTR, NMI, TEST, TMR IN Setup Time(1)	15		15		15		ns	
T_{INVCL}	DRQ0, DRQ1, Setup Time(1)	15		15		15		ns	
80C186 MASTER INTERFACE TIMING RESPONSES									
T_{CLAV}	Address Valid Delay	5	50	5	36	5	33	ns	C_L = 50 pF −200 pF all outputs (except T_{CLTMV}) @ 10 MHz
T_{CLAX}	Address Hold	0		0		0		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	30	T_{CLAX}	25	T_{CLAX}	20	ns	
T_{CHCZ}	Command Lines Float Delay		40		33		28	ns	
T_{CHCV}	Command Lines Valid Delay (after Float)		45		37		32	ns	
T_{LHLL}	ALE Width (min)	$T_{CLCL} - 30$		$T_{CLCL} - 30$		$T_{CLCL} - 30$		ns	C_L = 50 pF −100 pF all outputs @ 12.5 & 16 MHz
T_{CHLH}	ALE Active Delay		30		25		20	ns	
T_{CHLL}	ALE Inactive Delay		30		25		20	ns	
T_{LLAX}	Address Hold to ALE Inactive (min)	$T_{CHCL} - 20$		$T_{CHCL} - 15$		$T_{CHCL} - 15$		ns	
T_{CLDV}	Data Valid Delay	5	40	5	36	5	33	ns	
T_{CLDOX}	Data Hold Time	5		5		5		ns	
T_{WHDX}	Data Hold after $\overline{WR}$ (min)	$T_{CLCL} - 34$		$T_{CLCL} - 20$		$T_{CLCL} - 20$		ns	
T_{CVCTV}	Control Active Delay 1	5	56	5	47	5	31	ns	
T_{CHCTV}	Control Active Delay 2	5	44	5	37	5	31	ns	
T_{CVCTX}	Control Inactive Delay	5	44	5	37	5	31	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)	5	56	5	47	5	35	ns	

NOTE:
1. To guarantee recognition at next clock.

PIN TIMINGS (Continued)

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at 16 MHz

All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
Input V_{IL} = 0.45V and V_{IH} = 2.4V for A.C. tests.

Symbol	Parameter	80C186-10		80C186-12		80C186-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C186 MASTER INTERFACE TIMING RESPONSES (Continued)									
T_{AZRL}	Address Float to $\overline{\text{RD}}$ Active	0		0		0		ns	C_L = 50–200 pF all outputs (except T_{CLTMV}) @ 10 MHz
T_{CLRL}	$\overline{\text{RD}}$ Active Delay	5	44	5	37	5	31	ns	
T_{CLRH}	$\overline{\text{RD}}$ Inactive Delay	5	44	5	37	5	31	ns	
T_{RHAV}	$\overline{\text{RD}}$ Inactive to Address Active (min)	T_{CLCL} − 40		T_{CLCL} − 20		T_{CLCL} − 20		ns	C_L = 50–100 pF all outputs @ 12.5 & 16 MHz
T_{CLHAV}	HLDA Valid Delay	5	40	5	33	5	25	ns	
T_{RLRH}	$\overline{\text{RD}}$ Pulse Width (min)	$2T_{CLCL}$ − 46		$2T_{CLCL}$ − 40		$2T_{CLCL}$ − 30		ns	
T_{WLWH}	$\overline{\text{WR}}$ Pulse Width (min)	$2T_{CLCL}$ − 34		$2T_{CLCL}$ − 30		$2T_{CLCL}$ − 25		ns	
T_{AVLL}	Address Valid to ALE Low (min)	T_{CLCH} − 19		T_{CLCH} − 15		T_{CLCH} − 15		ns	Equal Loading
T_{CHSV}	Status Active Delay	5	45	5	35	5	31	ns	
T_{CLSH}	Status Inactive Delay	5	50	5	35	5	30	ns	
T_{CLTMV}	Timer Output Delay		48		40		30	ns	100 pF max @ 10 MHz
T_{CLRO}	Reset Delay		48		40		30	ns	C_L = 50–200 pF All outputs (except T_{CLTMV}) @ 10 MHz
T_{CHQSV}	Queue Status Delay		28		28		25	ns	
T_{CHDX}	Status Hold Time	5		5		5		ns	
T_{AVCH}	Address Valid to Clock High	0		0		0		ns	C_L = 50–100 pF All outputs @ 12.5 & 16 MHz
T_{CLLV}	$\overline{\text{LOCK}}$ Valid/Invalid Delay	5	45	5	40	5	35	ns	
T_{DXDL}	$\overline{\text{DEN}}$ Inactive to DT/$\overline{\text{R}}$ Low	0		0		0		ns	Equal Loading
80C186 CHIP-SELECT TIMING RESPONSES									
T_{CLCSV}	Chip-Select Active Delay		45		33		30	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	T_{CLCH} − 10		T_{CLCH} − 10		T_{CLCH} − 10		ns	Equal Loading
T_{CHCSX}	Chip-Select Inactive Delay	5	32	5	28	5	23	ns	

PIN TIMINGS (Continued)

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at 16 MHz

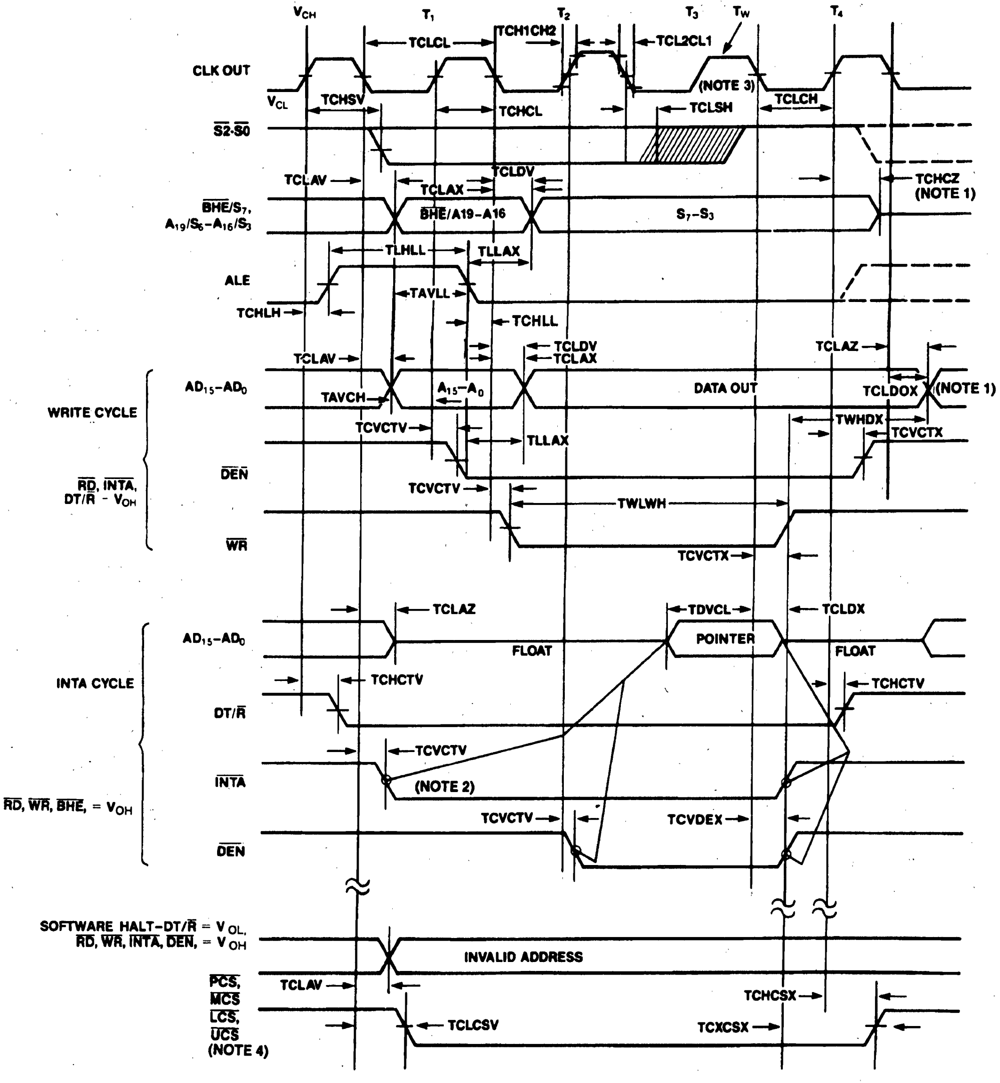
All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
Input V_{IL} = 0.45V and V_{IH} = 2.4V for A.C. tests.

Symbol	Parameter	80C186-10		80C186-12		80C186-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C186 CLKIN REQUIREMENTS Measurements taken with following conditions: External clock input to X1 and X2 not connected (float)									
T_{CKIN}	CLKIN Period	50	1000	40	1000	31.25	1000	ns	
T_{CKHL}	CLKIN Fall Time		5		5		5	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		5		5		5	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	20		15		13		ns	1.5V(2)
T_{CHCK}	CLKIN High Time	20		15		13		ns	1.5V(2)
80C186 CLKOUT TIMING 200 pF load maximum for 10 MHz or less, 100 pF load maximum above 10 MHz									
T_{CICO}	CLKIN to CLKOUT Skew		25		21		17	ns	
T_{CLCL}	CLKOUT Period	100	2000	80	2000	62.5	2000	ns	
T_{CLCH}	CLKOUT Low Time (min)	0.5 T_{CLCL}−6		0.5 T_{CLCL}−5		0.5 T_{CLCL}−5		ns	1.5V
T_{CHCL}	CLKOUT High Time (min)	0.5 T_{CLCL}−6		0.5 T_{CLCL}−5		0.5 T_{CLCL}−5		ns	1.5V
T_{CH1CH2}	CLKOUT Rise Time		10		10		8	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		10		10		8	ns	3.5 to 1.0V

NOTE:
2. T_{CLCK} and T_{CHCK} (CLKIN Low and High times) should not have a duration less than 40% of T_{CKIN}.

WAVEFORMS

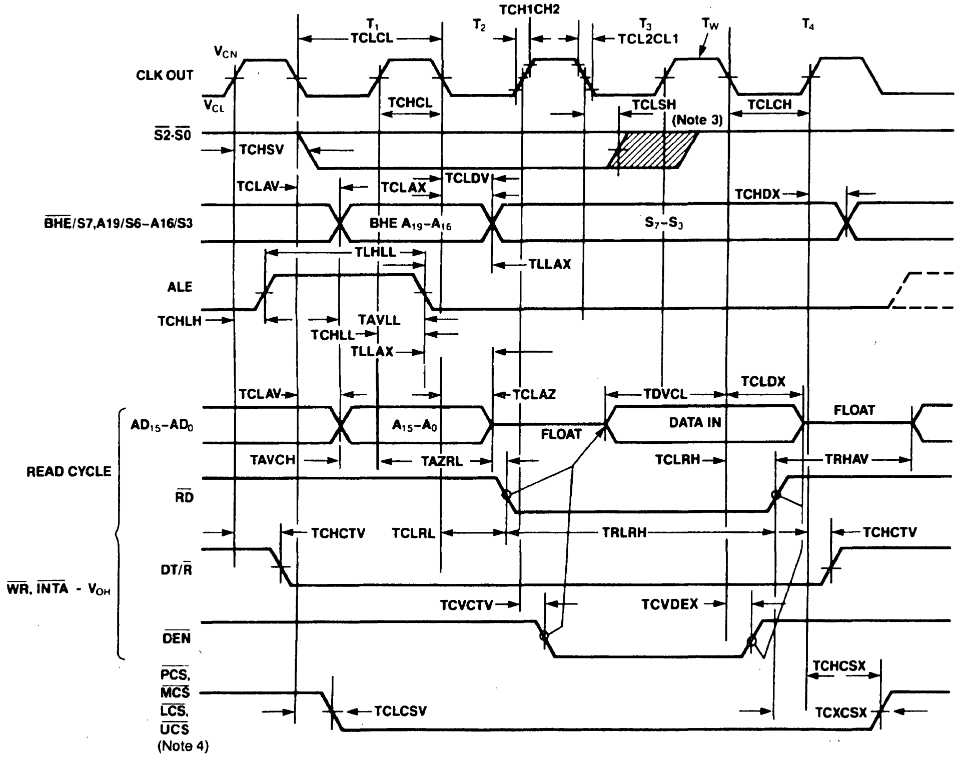
MAJOR CYCLE TIMING



270354-16

WAVEFORMS (Continued)

MAJOR CYCLE TIMING (Continued)

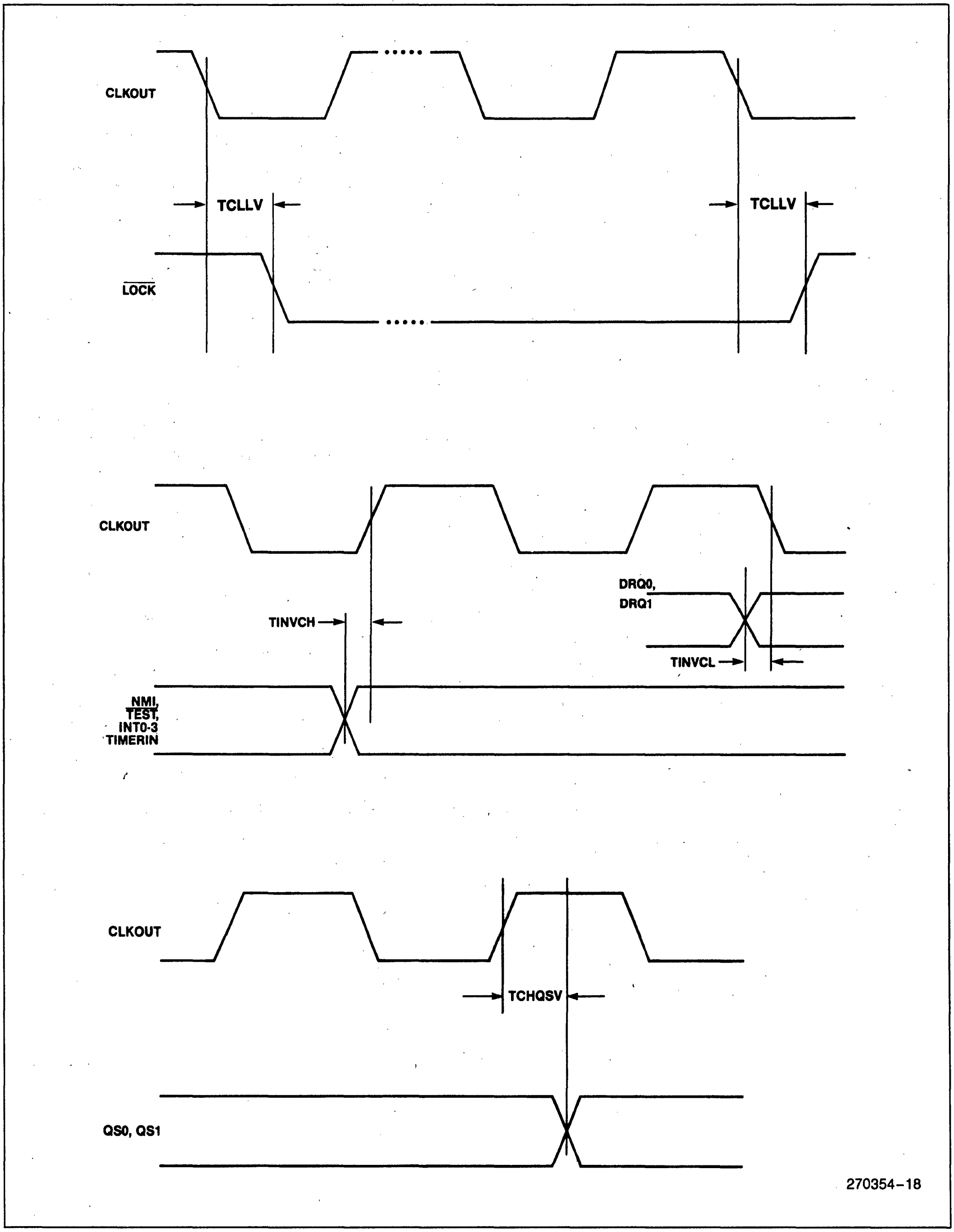


270354-17

NOTES:

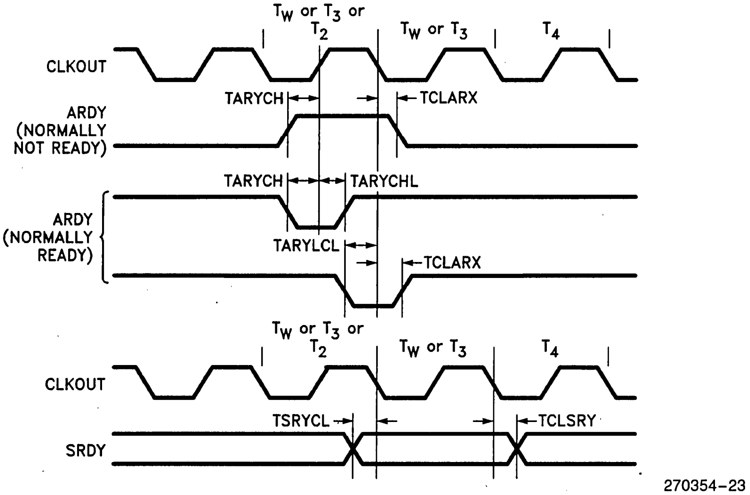
1. Following a Write cycle, the Local Bus is floated by the 80C186 only when the 80C186 enters a "Hold Acknowledge" state.
2. INTA occurs one clock later in slave mode.
3. Status inactive just prior to T_4.
4. Latched A1 and A2 have the same timings as $\overline{PCS5}$ and $\overline{PCS6}$.

WAVEFORMS (Continued)



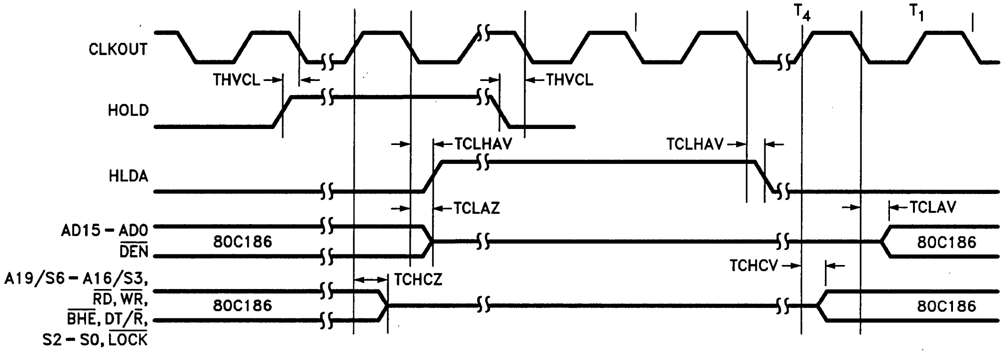
WAVEFORMS (Continued)

READY TIMING



270354–23

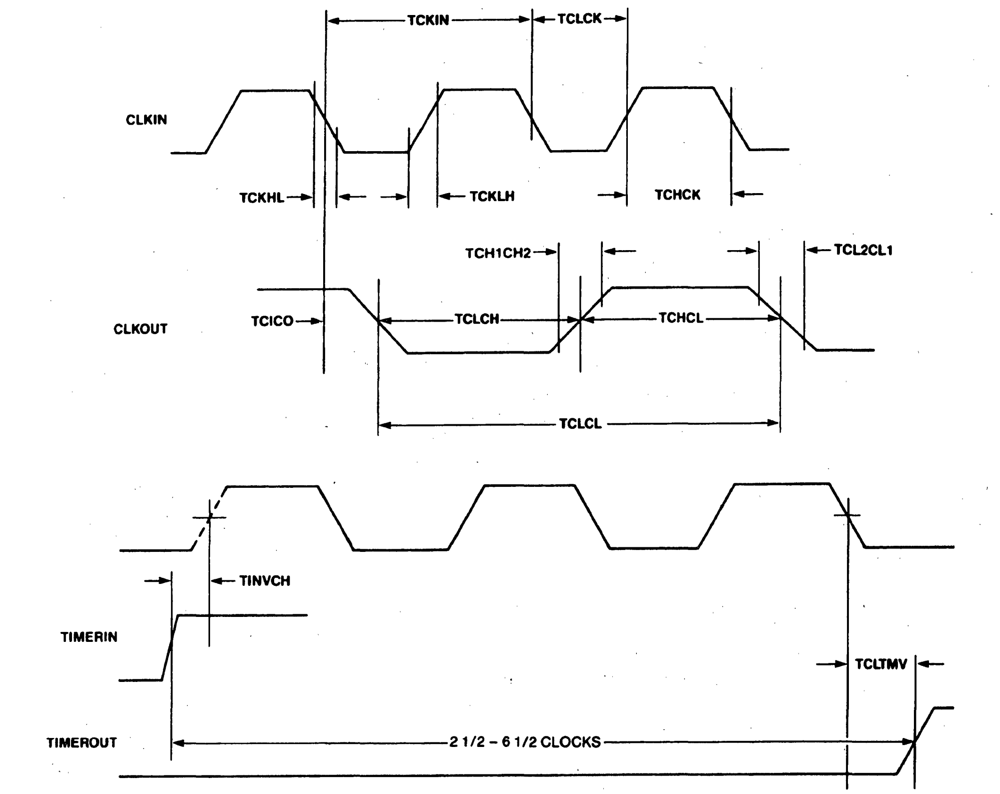
HOLD-HLDA TIMING



270354–20

WAVEFORMS (Continued)

TIMER ON 80C186



270354-21

80C186 EXECUTION TIMINGS

A determination of 80C186 program execution timing must consider both the bus cycles necessary to prefetch instructions as well as the number of execution unit cycles necessary to execute instructions. The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDs occur.
- All word-data is located on even-address boundaries.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

All instructions which involve memory accesses can require one or two additional clocks above the minimum timings shown due to the asynchronous handshake between the BIU and execution unit.

With a 16-bit BIU, the 80C186 has sufficient bus performance to ensure that an adequate number of prefetched bytes will reside in the queue most of the time. Therefore, actual program execution will not be substantially greater than that derived from adding the instruction timings shown.

INSTRUCTION SET SUMMARY

Function	Format				Clock Cycles	Comments
DATA TRANSFER						
MOV = Move:						
Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m			2/12	
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m			2/9	
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w=1	12–13	8/16-bit
Immediate to register	1 0 1 1 w reg	data	data if w=1		3–4	8/16-bit
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high		8	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high		9	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m			2/9	
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m			2/11	
PUSH = Push:						
Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m			16	
Register	0 1 0 1 0 reg				10	
Segment register	0 0 0 reg 1 1 0				9	
Immediate	0 1 1 0 1 0 s 0	data	data if s=0		10	
PUSHA = Push All	0 1 1 0 0 0 0 0				36	
POP = Pop:						
Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m			20	
Register	0 1 0 1 1 reg				10	
Segment register	0 0 0 reg 1 1 1	(reg≠01)			8	
POPA = Pop All	0 1 1 0 0 0 0 1				51	
XCHG = Exchange:						
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m			4/17	
Register with accumulator	1 0 0 1 0 reg				3	
IN = Input from:						
Fixed port	1 1 1 0 0 1 0 w	port			10	
Variable port	1 1 1 0 1 1 0 w				8	
OUT = Output to:						
Fixed port	1 1 1 0 0 1 1 w	port			9	
Variable port	1 1 1 0 1 1 1 w				7	
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1				11	
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m			6	
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m	(mod≠11)		18	
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m	(mod≠11)		18	
LAHF = Load AH with flags	1 0 0 1 1 1 1 1				2	
SAHF = Store AH into flags	1 0 0 1 1 1 1 0				3	
PUSHF = Push flags	1 0 0 1 1 1 0 0				9	
POPF = Pop flags	1 0 0 1 1 1 0 1				8	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
DATA TRANSFER (Continued)						
SEGMENT = Segment Override:						
CS	0 0 1 0 1 1 1 0				2	
SS	0 0 1 1 0 1 1 0				2	
DS	0 0 1 1 1 1 1 0				2	
ES	0 0 1 0 0 1 1 0				2	
ARITHMETIC						
ADD = Add:						
Reg/memory with register to either	0 0 0 0 0 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s w = 01	4/16	
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w = 1		3/4	8/16-bit
ADC = Add with carry:						
Reg/memory with register to either	0 0 0 1 0 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s w = 01	4/16	
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w = 1		3/4	8/16-bit
INC = Increment:						
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m			3/15	
Register	0 1 0 0 0 reg				3	
SUB = Subtract:						
Reg/memory and register to either	0 0 1 0 1 0 d w	mod reg r/m			3/10	
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s w = 01	4/16	
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w = 1		3/4	8/16-bit
SBB = Subtract with borrow:						
Reg/memory and register to either	0 0 0 1 1 0 d w	mod reg r/m			3/10	
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s w = 01	4/16	
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w = 1		3/4	8/16-bit
DEC = Decrement						
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m			3/15	
Register	0 1 0 0 1 reg				3	
CMP = Compare:						
Register/memory with register	0 0 1 1 1 0 1 w	mod reg r/m			3/10	
Register with register/memory	0 0 1 1 1 0 0 w	mod reg r/m			3/10	
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w = 01	3/10	
Immediate with accumulator	0 0 1 1 1 1 0 w	data	data if w = 1		3/4	8/16-bit
NEG = Change sign register/memory	1 1 1 1 0 1 1 w	mod 0 1 1 r/m			3/10	
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1				8	
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1				4	
AAS = ASCII adjust for subtract	0 0 1 1 1 1 1 1				7	
DAS = Decimal adjust for subtract	0 0 1 0 1 1 1 1				4	
MUL = Multiply (unsigned):	1 1 1 1 0 1 1 w	mod 100 r/m				
Register-Byte					26–28	
Register-Word					35–37	
Memory-Byte					32–34	
Memory-Word					41–43	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
ARITHMETIC (Continued)						
IMUL = Integer multiply (signed):	1 1 1 1 0 1 1 w	mod 1 0 1 r/m				
Register-Byte					25–28	
Register-Word					34–37	
Memory-Byte					31–34	
Memory-Word					40–43	
IMUL = Integer Immediate multiply (signed)	0 1 1 0 1 0 s 1	mod reg r/m	data	data if s=0	22–25/ 29–32	
DIV = Divide (unsigned):	1 1 1 1 0 1 1 w	mod 1 1 0 r/m				
Register-Byte					29	
Register-Word					38	
Memory-Byte					35	
Memory-Word					44	
IDIV = Integer divide (signed):	1 1 1 1 0 1 1 w	mod 1 1 1 r/m				
Register-Byte					44–52	
Register-Word					53–61	
Memory-Byte					50–58	
Memory-Word					59–67	
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0			19	
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0			15	
CBW = Convert byte to word	1 0 0 1 1 0 0 0				2	
CWD = Convert word to double word	1 0 0 1 1 0 0 1				4	
LOGIC						
Shift/Rotate Instructions:						
Register/Memory by 1	1 1 0 1 0 0 0 w	mod TTT r/m			2/15	
Register/Memory by CL	1 1 0 1 0 0 1 w	mod TTT r/m			5+n/17+n	
Register/Memory by Count	1 1 0 0 0 0 0 w	mod TTT r/m	count		5+n/17+n	
		TTT Instruction				
		0 0 0 ROL				
		0 0 1 ROR				
		0 1 0 RCL				
		0 1 1 RCR				
		1 0 0 SHL/SAL				
		1 0 1 SHR				
		1 1 1 SAR				
AND = And:						
Reg/memory and register to either	0 0 1 0 0 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w=1	4/16	
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w=1		3/4	8/16-bit
TEST = And function to flags, no result:						
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m			3/10	
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w=1	4/10	
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w=1		3/4	8/16-bit
OR = Or:						
Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w=1	4/16	
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w=1		3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
LOGIC (Continued)						
XOR = Exclusive or:						
Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m			3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1	4/16	
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w = 1		3/4	8/16-bit
NOT = Invert register/memory	1 1 1 1 0 1 1 w	mod 0 1 0 r/m			3/10	
STRING MANIPULATION						
MOVS = Move byte/word	1 0 1 0 0 1 0 w				14	
CMPS = Compare byte/word	1 0 1 0 0 1 1 w				22	
SCAS = Scan byte/word	1 0 1 0 1 1 1 w				15	
LODS = Load byte/wd to ALAX	1 0 1 0 1 1 0 w				12	
STOS = Stor byte/wd from ALA	1 0 1 0 1 0 1 w				10	
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w				14	
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w				14	
Repeated by count in CX						
MOVS = Move string	1 1 1 1 0 0 1 0	1 0 1 0 0 1 0 w			8+8n	
CMPS = Compare string	1 1 1 1 0 0 1 z	1 0 1 0 0 1 1 w			5+22n	
SCAS = Scan string	1 1 1 1 0 0 1 z	1 0 1 0 1 1 1 w			5+15n	
LODS = Load string	1 1 1 1 0 0 1 0	1 0 1 0 1 1 0 w			6+11n	
STOS = Store string	1 1 1 1 0 0 1 0	1 0 1 0 1 0 1 w			6+9n	
INS = Input string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 0 w			8+8n	
OUTS = Output string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 1 w			8+8n	
CONTROL TRANSFER						
CALL = Call:						
Direct within segment	1 1 1 0 1 0 0 0	disp-low	disp-high		15	
Register/memory indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m			13/19	
Direct intersegment	1 0 0 1 1 0 1 0	segment offset			23	
		segment selector				
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	(mod ≠ 11)		38	
JMP = Unconditional jump:						
Short/long	1 1 1 0 1 0 1 1	disp-low			14	
Direct within segment	1 1 1 0 1 0 0 1	disp-low	disp-high		14	
Register/memory indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m			11/17	
Direct intersegment	1 1 1 0 1 0 1 0	segment offset			14	
		segment selector				
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	(mod ≠ 11)		26	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
CONTROL TRANSFER (Continued)						
RET = Return from CALL:						
Within segment	11000011				16	
Within seg adding immed to SP	11000010	data-low	data-high		18	
Intersegment	11001011				22	
Intersegment adding immediate to SP	11001010	data-low	data-high		25	
JE/JZ = Jump on equal/zero	01110100	disp			4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100	disp			4/13	
JLE/JNG = Jump on less or equal/not greater	01111110	disp			4/13	
JB/JNAE = Jump on below/not above or equal	01110010	disp			4/13	
JBE/JNA = Jump on below or equal/not above	01110110	disp			4/13	
JP/JPE = Jump on parity/parity even	01111010	disp			4/13	
JO = Jump on overflow	01110 000	disp			4/13	
JS = Jump on sign	01111000	disp			4/13	
JNE/JNZ = Jump on not equal/not zero	01110101	disp			4/13	
JNL/JGE = Jump on not less/greater or equal	01111101	disp			4/13	
JNLE/JG = Jump on not less or equal/greater	01111111	disp			4/13	
JNB/JAE = Jump on not below/above or equal	01110011	disp			4/13	
JNBE/JA = Jump on not below or equal/above	01110111	disp			4/13	
JNP/JPO = Jump on not par/par odd	01111011	disp			4/13	
JNO = Jump on not overflow	01110001	disp			4/13	
JNS = Jump on not sign	01111001	disp			4/13	
JCXZ = Jump on CX zero	11100011	disp			5/15	
LOOP = Loop CX times	11100010	disp			6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001	disp			6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000	disp			6/16	
ENTER = Enter Procedure	11001000	data-low	data-high	L		
L = 0					15	
L = 1					25	
L > 1					22+16(n−1)	
LEAVE = Leave Procedure	11001001				8	
INT = **Interrupt:**						
Type specified	11001101	type			47	
Type 3	11001100				45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	11001110				48/4	
IRET = Interrupt return	11001111				28	
BOUND = Detect value out of range	01100010	mod reg r/m			33–35	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
PROCESSOR CONTROL			
CLC = Clear carry	11111000	2	
CMC = Complement carry	11110101	2	
STC = Set carry	11111001	2	
CLD = Clear direction	11111100	2	
STD = Set direction	11111101	2	
CLI = Clear interrupt	11111010	2	
STI = Set interrupt	11111011	2	
HLT = Halt	11110100	2	
WAIT = Wait	10011011	6	if $\overline{\text{test}}$ = 0
LOCK = Bus lock prefix	11110000	2	
ESC = Processor Extension Escape	11011TTT | mod LLL r/m	6	
	(TTT LLL are opcode to processor extension)		

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	**Segment Register**
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	**8-Bit (w = 0)**
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

80188
HIGH INTEGRATION 8-BIT MICROPROCESSOR

- **Integrated Feature Set**
  - **— Enhanced 8086-2 CPU**
  - **— Clock Generator**
  - **— 2 Independent DMA Channels**
  - **— Programmable Interrupt Controller**
  - **— 3 Programmable 16-Bit Timers**
  - **— Programmable Memory and Peripheral Chip-Select Logic**
  - **— Programmable Wait State Generator**
  - **— Local Bus Controller**
- **High-Performance 8 MHz Processor**
  - **— At 8 MHz Provides 2 Times the Performance of the Standard 8088**
  - **— 2 MByte/Sec Bus Bandwidth Interface @8 MHz**
- **Available in EXPRESS**
  - **— Standard Temperature with Burn-In**
  - **— Extended Temperature Range (−40°C to +85°C)**
- **8-Bit Data Bus Interface; 16-Bit Internal Architecture**
- **Completely Object Code Compatible with All Existing 8086/8088 Software**
  - **— 10 New Instruction Types**
- **DRAM Refresh Capability via DMA Channel and Timer 2**
- **Direct Addressing Capability to 1 MByte of Memory and 64 KByte I/O**
- **Complete System Development Support**
  - **— Development Software; Assembler, PL/M, Pascal, Fortran, and System Utilities**
  - **— In-Circuit-Emulator (I²ICE™-186/188)**
- **High Performance Numerical Coprocessing Capability Through 8087 Interface**
- **Available in 68 Pin:**
  - **— Ceramic Leadless Chip Carrier (LCC)**
  - **— Ceramic Pin Grid Array (PGA)**
  - **— Plastic Leaded Chip Carrier (PLCC)**

(See Packaging Outlines and Dimensions, Order #231369)

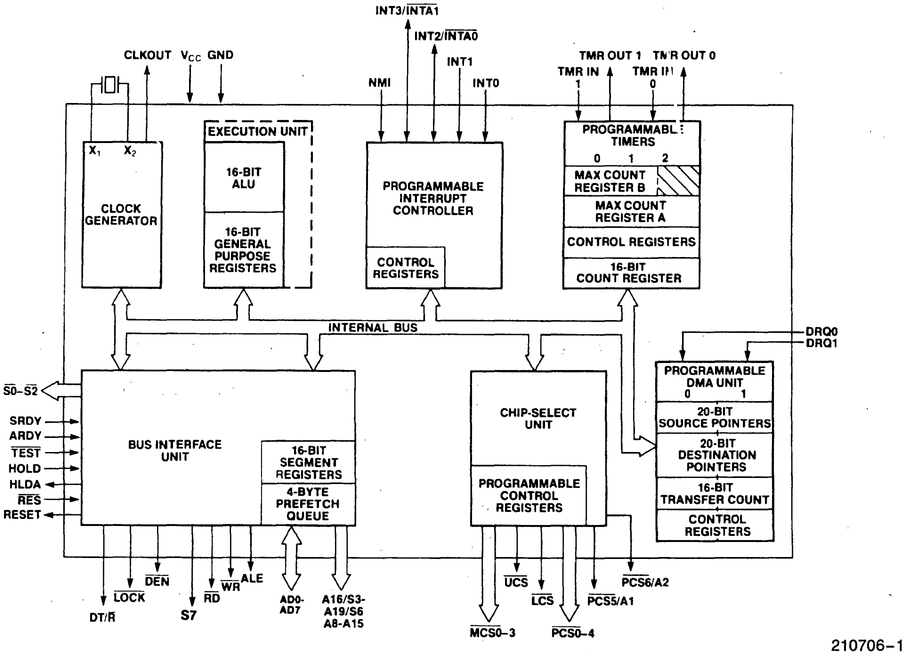


Figure 1. 80188 Block Diagram

The Intel 80188 is a highly integrated microprocessor with an 8-bit data bus interface and a 16-bit internal architecture to give high performance. The 80188 effectively combines 15–20 of the most common 8088 system components onto one. The 80188 provides two times greater throughput than the standard 5 MHz 8088. The 80188 is upward compatible with 8086 and 8088 software and adds 10 new instruction types to the existing set.

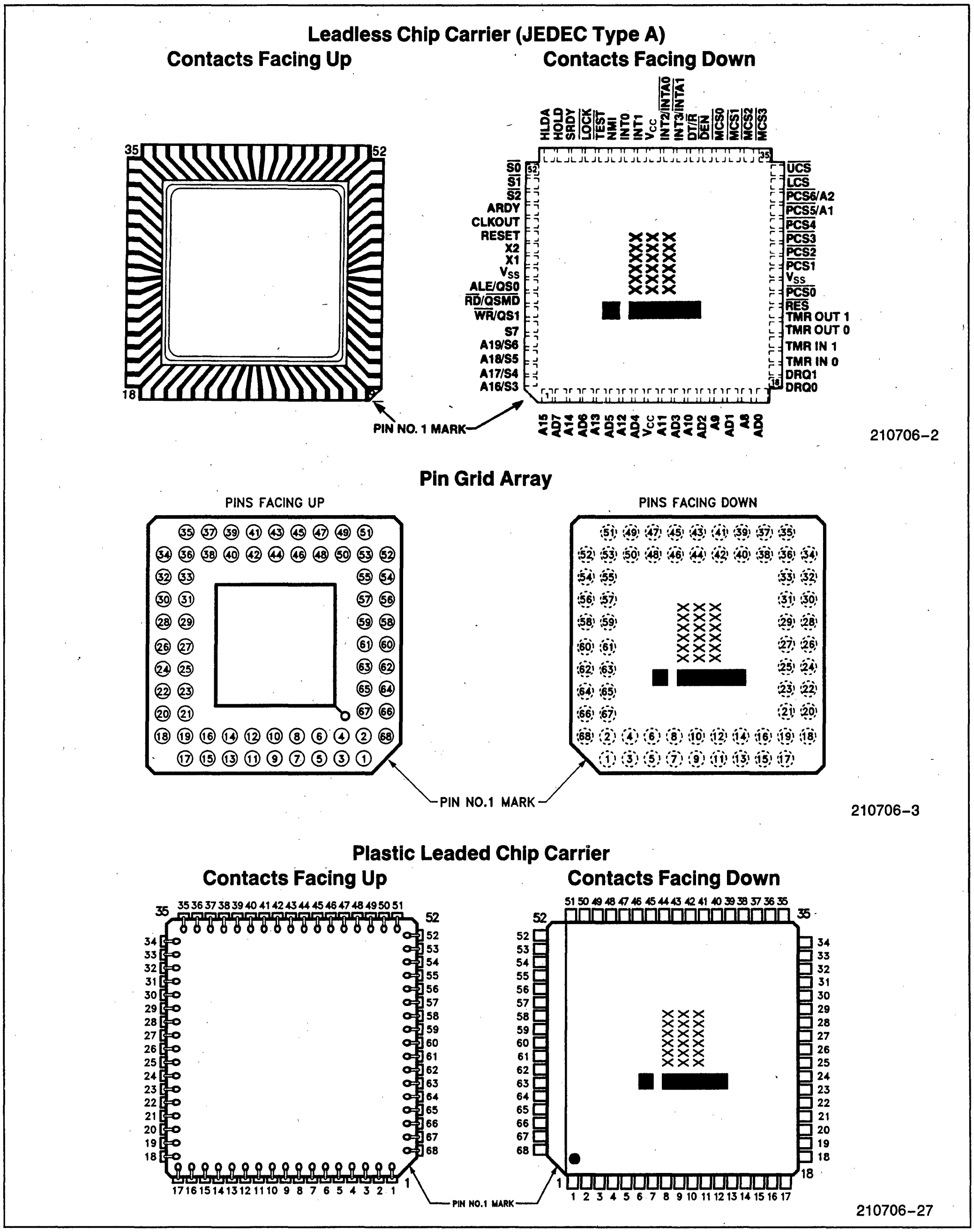


Figure 2. 80188 Pinout Diagram

Table 1. 80188 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}, V_{CC}	9, 43	I	**SYSTEM POWER:** +5 volt power supply.
V_{SS}, V_{SS}	26, 60	I	**SYSTEM GROUND**
RESET	57	O	**RESET OUTPUT:** Indicates that the 80188 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{\text{RES}}$ signal.
X1, X2	59, 58	I	**CRYSTAL INPUTS:** X1 and X2 provide external connections for a fundamental mode parallel resonant crystal for the internal oscillator. Instead of using a crystal, an external clock may be applied to X1 while minimizing stray capacitance on X2. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	**CLOCK OUTPUT:** Provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT. CLKOUT has sufficient MOS drive capabilities for the 8087 Numeric Processor Extension.
$\overline{\text{RES}}$	24	I	**SYSTEM RESET:** Causes the 80188 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80188 clock. The 80188 begins fetching instructions approximately 7 clock cycles after $\overline{\text{RES}}$ is returned HIGH. For proper initialization, V_{CC} must be within specifications and the clock signal must be stable for more than 4 clocks with $\overline{\text{RES}}$ held low. $\overline{\text{RES}}$ is internally synchronized. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{\text{RES}}$ generation via an RC network. When $\overline{\text{RES}}$ occurs, the 80188 will drive the status lines to an inactive level for one clock, and then float them.
$\overline{\text{TEST}}$	47	I	**$\overline{\text{TEST}}$:** Is examined by the WAIT instruction. If the $\overline{\text{TEST}}$ input is HIGH when "WAIT" execution begins, instruction execution will suspend. $\overline{\text{TEST}}$ will be resampled until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80188 is waiting for $\overline{\text{TEST}}$, interrupts will be serviced. This input is synchronized internally.
TMR IN 0, TMR IN 1	20 21	I I	**TIMER INPUTS:** Are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0, TMR OUT 1	22 23	O O	**TIMER OUTPUTS:** Are used to provide single pulse or continuous waveform generation, depending upon the timer mode selected.
DRQ0, DRQ1	18 19	I I	**DMA REQUEST:** Is driven HIGH by an external device when it desires that a DMA channel (Channel 0 or 1) perform a transfer. These signals are active HIGH, level-triggered, and internally synchronized.
NMI	46	I	**NON-MASKABLE INTERRUPT:** Is an edge-triggered input which causes a type 2 interrupt. NMI is not maskable internally. A transition from a LOW to HIGH initiates the interrupt at the next instruction boundary. NMI is latched internally. An NMI duration of one clock or more will guarantee service. This input is internally synchronized.

Table 1. 80188 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>INT0, INT1,
INT2/$\overline{\text{INTA0}}$,
INT3/$\overline{\text{INTA1}}$</td><td>45, 44
42
41</td><td>I
I/O
I/O</td><td><b>MASKABLE INTERRUPT REQUESTS:</b> Can be requested by activating one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured via software to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured via software to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When slave mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).</td></tr>
<tr><td>A19/S6,
A18/S5,
A17/S4,
A16/S3</td><td>65
66
67
68</td><td>O
O
O
O</td><td><b>ADDRESS BUS OUTPUTS (16–19) and BUS CYCLE STATUS (3–6):</b> Reflect the four most significant address bits during T$_1$. These signals are active HIGH. During T$_2$, T$_3$, T$_W$, and T$_4$, status information is available on these lines as encoded below:
<table>
<tr><th></th><th>Low</th><th>High</th></tr>
<tr><td>S6</td><td>Processor Cycle</td><td>DMA Cycle</td></tr>
</table>
S3, S4, and S5 are defined as LOW during T$_2$–T$_4$. The status pins float during HOLD/HLDA.</td></tr>
<tr><td>AD7–AD0</td><td>2, 4, 6, 8
11, 13, 15, 17</td><td>I/O</td><td><b>ADDRESS/DATA BUS (0-7):</b> Signals constitute the time multiplexed memory or I/O address (T$_1$) and data (T$_2$, T$_3$, T$_W$, and T$_4$) bus. The bus is active HIGH.</td></tr>
<tr><td>A15–A8</td><td>1, 3, 5, 7
10, 12, 14, 16</td><td>O</td><td><b>ADDRESS-ONLY BUS (8-15):</b> Containing valid address from T$_1$-T$_4$. The bus is active HIGH.</td></tr>
<tr><td>S7</td><td>64</td><td>O</td><td>This signal is HIGH to indicate that the 80188 has an 8-bit data bus. S7 floats during HOLD.</td></tr>
<tr><td>ALE/QS0</td><td>61</td><td>O</td><td><b>ADDRESS LATCH ENABLE/QUEUE STATUS 0:</b> Is provided by the 80188 to latch the address into the 8282/8283 address latches. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T$_1$ of the associated bus cycle, effectively one-half clock cycle earlier than in the standard 8088. The trailing edge is generated off the CLKOUT rising edge in T$_1$ as in the 8088. Note that ALE is never floated.</td></tr>
<tr><td>$\overline{\text{WR}}$/QS1</td><td>63</td><td>O</td><td><b>WRITE STROBE/QUEUE STATUS 1:</b> Indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{\text{WR}}$ is active for T$_2$, T$_3$, and T$_W$ of any write cycle. It is active LOW, and floats during "HOLD." It is driven HIGH for one clock during Reset, and then floated. When the 80188 is in queue status mode, the ALE/QS0 and $\overline{\text{WR}}$/QS1 pins provide information about processor/instruction queue interaction.
<table>
<tr><th>QS1</th><th>QS0</th><th>Queue Operation</th></tr>
<tr><td>0</td><td>0</td><td>No Queue Operation</td></tr>
<tr><td>0</td><td>1</td><td>First Opcode Byte Fetched from the Queue</td></tr>
<tr><td>1</td><td>1</td><td>Subsequent Byte Fetched from the Queue</td></tr>
<tr><td>1</td><td>0</td><td>Empty the Queue</td></tr>
</table></td></tr>
</table>

Table 1. 80188 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$\overline{RD}$/$\overline{QSMD}$</td><td>62</td><td>O</td><td><b>READ STROBE:</b> Indicates that the 80188 is performing a memory or I/O read cycle. $\overline{RD}$ is active LOW for T_2, T_3, and T_W of any read cycle. It is guaranteed not to go LOW in T_2 until after the Address Bus is floated. $\overline{RD}$ is active LOW, and floats during "HOLD". $\overline{RD}$ is driven HIGH for one clock during Reset, and then the output driver is floated. A weak internal pull-up mechanism on the $\overline{RD}$ line holds it HIGH when the line is not driven. During RESET the pin is sampled to determine whether the 80188 should provide ALE, $\overline{WR}$, and $\overline{RD}$, or if the Queue-Status should be provided. $\overline{RD}$ should be connected to GND to provide Queue-Status data.</td></tr>
<tr><td>ARDY</td><td>55</td><td>I</td><td><b>ASYNCHRONOUS READY:</b> Informs the 80188 that the addressed memory space or I/O device will complete a data transfer. The ARDY input pin will accept an asynchronous input, and is active HIGH. Only the rising edge is internally synchronized by the 80188. This means that the falling edge of ARDY must be synchronized to the 80188 clock. If connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active to terminate a bus cycle. If unused, this line should be tied LOW to yield control to the SRDY pin.</td></tr>
<tr><td>SRDY</td><td>49</td><td>I</td><td><b>SYNCHRONOUS READY:</b> Must be synchronized externally to the 80188. The use of SRDY provides a relaxed system-timing specification on the Ready input. This is accomplished by eliminating the one-half clock cycle which is required for internally resolving the signal level when using the ARDY input. This line is active HIGH. If this line is connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active before a bus cycle is terminated. If unused, this line should be tied LOW to yield control to the ARDY pin.</td></tr>
<tr><td>$\overline{LOCK}$</td><td>48</td><td>O</td><td><b>$\overline{LOCK}$:</b> Output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of the instruction following the LOCK prefix. No prefetches will occur while $\overline{LOCK}$ is asserted. When executing more than one LOCK instruction, always make sure there are 6 bytes of code between the end of the first LOCK instruction and the start of the second LOCK instruction. $\overline{LOCK}$ is active LOW, is driven HIGH for one clock during RESET, and then floated.</td></tr>
<tr><td>$\overline{S0}$, $\overline{S1}$, $\overline{S2}$</td><td>52–54</td><td>O</td><td><b>BUS CYCLE STATUS $\overline{S0}$–$\overline{S2}$:</b> Are encoded to provide bus-transaction information:
<table>
<tr><th colspan="4">80188 Bus Cycle Status Information</th></tr>
<tr><th>$\overline{S2}$</th><th>$\overline{S1}$</th><th>$\overline{S0}$</th><th>Bus Cycle Initiated</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Interrupt Acknowledge</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Read I/O</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Write I/O</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Halt</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Instruction Fetch</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Read Data from Memory</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Write Data to Memory</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Passive (no bus cycle)</td></tr>
</table>
The status pins float during "HOLD."
$\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator.
The status lines are driven HIGH for one clock during Reset, and then floated until a bus cycle begins.</td></tr>
</table>

Table 1. 80188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
HOLD (input) HLDA (output)	50 51	I O	**HOLD:** Indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80188 clock. The 80188 will issue a HLDA in response to a HOLD request at the end of T_4 or T_i. Simultaneous with the issuance of HLDA, the 80188 will float the local bus and control lines. After HOLD is detected as being LOW, the 80188 will lower HLDA. When the 80188 needs to run another bus cycle, it will again drive the local bus and control lines.
$\overline{\mathrm{UCS}}$	34	O	**UPPER MEMORY CHIP SELECT:** Is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{\mathrm{UCS}}$ is software programmable.
$\overline{\mathrm{LCS}}$	33	O	**LOWER MEMORY CHIP SELECT:** Is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{\mathrm{LCS}}$ is software programmable.
$\overline{\mathrm{MCS0}}$–3	38, 37, 36, 35	O	**MID-RANGE MEMORY CHIP SELECT SIGNALS:** Are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{\mathrm{MCS0}}$–3 are software programmable.
$\overline{\mathrm{PCS0}}$–4	25, 27-30	O	**PERIPHERAL CHIP SELECT SIGNALS 0–4:** Are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{\mathrm{PCS0}}$–4 are software programmable.
$\overline{\mathrm{PCS5}}$/A1	31	O	**PERIPHERAL CHIP SELECT 5 or LATCHED A1:** May be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{\mathrm{PCS5}}$ is software programmable. When programmed to provide latched A1, rather than $\overline{\mathrm{PCS5}}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{\mathrm{PCS6}}$/A2	32	O	**PERIPHERAL CHIP SELECT 6 or LATCHED A2:** May be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{\mathrm{PCS6}}$ is software programmable. When programmed to provide latched A2, rather than $\overline{\mathrm{PCS6}}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{\mathrm{R}}$	40	O	**DATA TRANSMIT/RECEIVE:** Controls the direction of data flow through the external 8286/8287 data bus transceiver. When LOW, data is transferred to the 80188. When HIGH the 80188 places write data on the data bus.
$\overline{\mathrm{DEN}}$	39	O	**DATA ENABLE:** Is provided as an 8286/8287 data bus transceiver output enable. $\overline{\mathrm{DEN}}$ is active LOW during each memory and I/O access. $\overline{\mathrm{DEN}}$ is HIGH whenever DT/$\overline{\mathrm{R}}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80188. This architecture is common to the 8086, 8088 and 80286 microprocessor families as well. The 80188 is a very high integration 8-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip while providing twice the performance of the standard 8088. The 80188 is object code compatible with the 8086, 8088 microprocessors and adds 10 new instruction types to the existing 8086, 8088 instruction set.

80188 BASE ARCHITECTURE

The 8086, 8088, 80186, 80188 and 80286 family all contain the same basic set of registers, instructions, and addressing modes. The 80188 processor is upward compatible with the 8086, 8088, 80186, and 80286 CPUs.

Register Set

The 80188 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

GENERAL REGISTERS

Eight 16-bit general purpose registers may be used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

SEGMENT REGISTERS

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

BASE AND INDEX REGISTERS

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

STATUS AND CONTROL REGISTERS

Two 16-bit special purpose registers record or alter certain aspects of the 80188 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

STATUS WORD DESCRIPTION

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80188 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

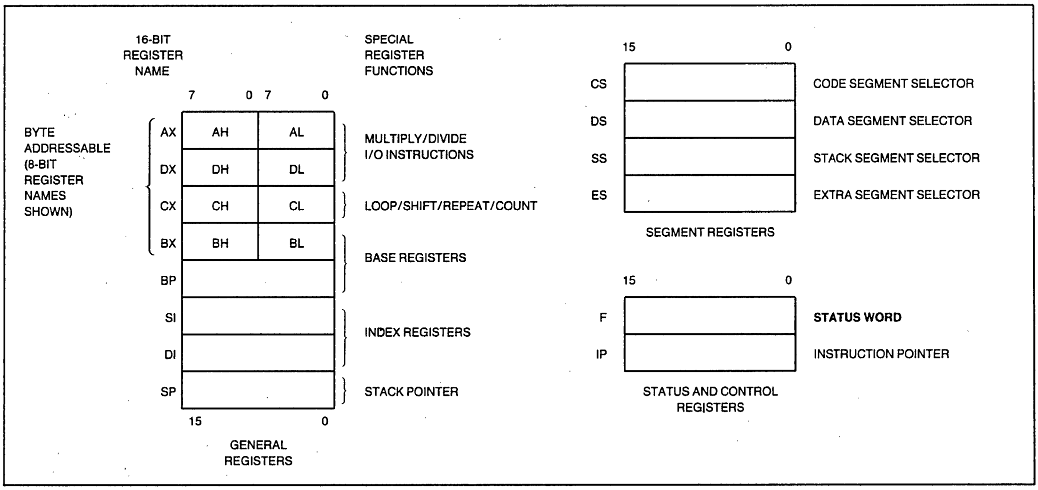
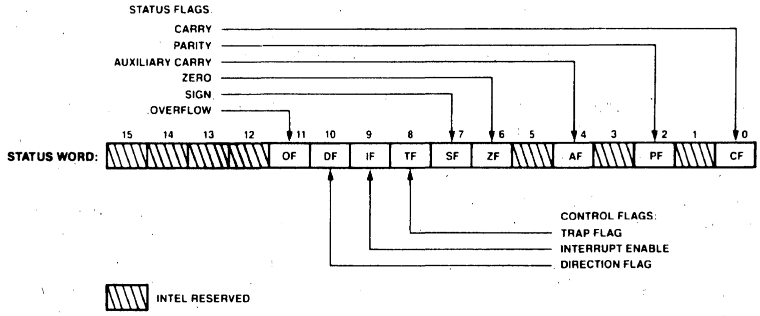


Figure 3a. 80188 Register Set



210706-4

Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-Enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80188 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment registers (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. 80188 Instruction Set

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign

JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4. 80188 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

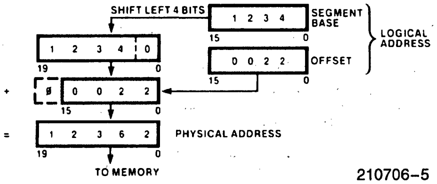


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

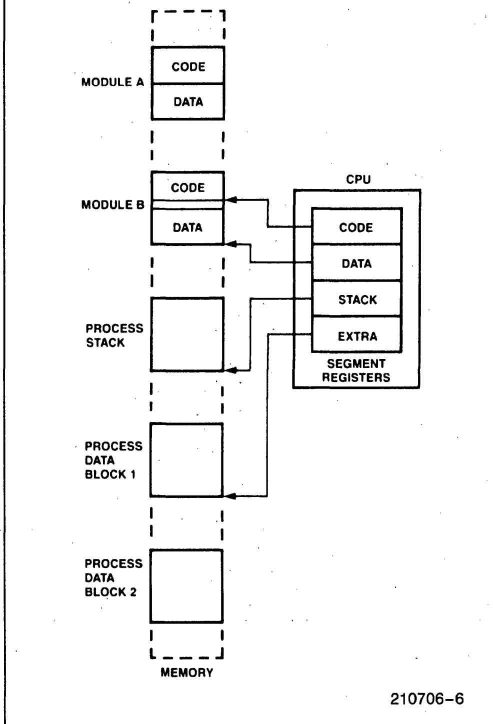


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80188 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an index register.
- *Based Indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80188 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using the 8087 Numeric Data Coprocessor with the 80188.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using the 8087 Numeric Data Coprocessor with the 80188.)

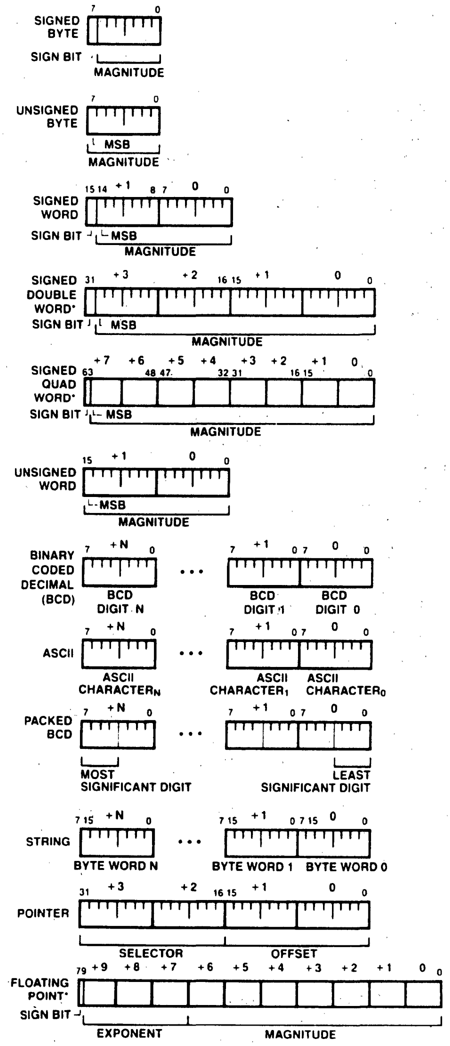
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80188.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



210706-7

NOTE:
*Supported using an 8078 Numeric Data Coprocessor with the 80188.

Figure 7. 80188 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80188 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80188 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80188 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INTO, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80188 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80188 Interrupt Vectors

Interrupt Name	Vector Type	Default Priority	Related Instructions
Divide Error Exception	0	*1	DIV, IDIV
Single Step Interrupt	1	12**	All
NMI	2	1	All
Breakpoint Interrupt	3	*1	INT
INTO Detected Overflow Exception	4	*1	INTO
Array Bounds Exception	5	*1	BOUND
Unused-Opcode Exception	6	*1	Undefined Opcodes
ESC Opcode Exception	7	*1***	ESC Opcodes
Timer 0 Interrupt	8	2A****	
Timer 1 Interrupt	18	2B****	
Timer 2 Interrupt	19	2C****	
Reserved	9	3	
DMA 0 Interrupt	10	4	
DMA 1 Interrupt	11	5	
INT0 Interrupt	12	6	
INT1 Interrupt	13	7	
INT2 Interrupt	14	8	
INT3 Interrupt	15	9	

NOTES:

*1. These are generated as the result of an instruction execution.

**2. This is handled as in the 8088.

****3. All three timers constitute one source of request to the interrupt controller. The Timer Interrupts all have the same default priority level with respect to all other interrupt sources. However, they have a defined priority ordering amongst themselves. (Priority 2A is higher priority than 2B.) Each Timer Interrupt has a separate vector type number.

4. Default priorities for the interrupt sources are used only if the user does not program each source into a unique priority level.

***5. An escape opcode will cause a trap only if the proper bit is set in the peripheral control block relocation register.

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INTO DETECTED OVERFLOW EXCEPTION (TYPE 4)

Generated during an INTO instruction if the 0F bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H–DFH). This exception will only be generated if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80188 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80188 integrated DMA controller and the integrated timer unit. The vector types for these interrupts are shown in Table 4. Software enables these inputs by setting the Interrupt Flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80188 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80188 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80188 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80188 begins execution with the instruction at physical location FFFF0(H). $\overline{RES}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80188 Initial Register State after RESET

Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

THE 80188 COMPARED TO THE 80186

The 80188 CPU is an 8-bit processor designed around the 80186 internal structure. Most internal functions of the 80188 are identical to the equivalent 80186 functions. The 80188 handles the external bus the same way the 80186 does with the distinction of handling only 8 bits at a time. Sixteen bit operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. The differences between the 80188 and the 80186 are outlined below. Internally, there are three differences between the 80188 and the 80186. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 80188, whereas the 80186 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 80188 BIU will fetch a new instruction to load into the queue each time there is a 1-byte hole (space available) in the queue. The 80186 waits until a 2-byte space is available.
- The internal execution time of the instruction is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU may also be limited by the speed of instruction fetches when a series of simple operations occur. When the more sophisticated instructions of the 80188 are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

The 80188 and 80186 are completely software compatible by virtue of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally well on an 80188 or an 80186.

The hardware interface of the 80188 contains the major differences between the two CPUs. The pin assignments are nearly identical, however, with the following functional changes.

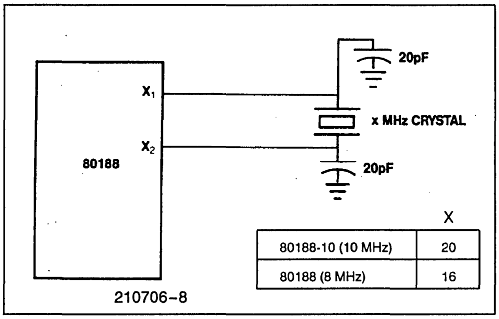
- A8–A15—These pins are only address outputs on the 80188. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.
- $\overline{BHE}$ has no meaning on the 80188 and has been eliminated.

80188 Clock Generator

The 80188 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The oscillator circuit of the 80188 is designed to be used with a parallel resonant fundamental mode crystal. This is used as the time base for the 80188. The crystal frequency selected will be double the CPU clock frequency. Use of an LC or RC circuit is not recommended with this oscillator. If an external oscillator is used, it can be connected directly to input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the 80188. The recommended crystal configuration is shown in Figure 8.



	X
80188-10 (10 MHz)	20
80188 (8 MHz)	16

Figure 8. Recommended 80188 Crystal Configuration

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_L (Load Capacitance):	20 pf ± 2 pf
Drive Level:	1 mW max

Clock Generator

The 80188 clock generator provides the 50% duty cycle processor clock for the 80188. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80188. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80188 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used. Full synchronization is performed only on the rising edge of ARDY, i.e., the falling edge of ARDY must be synchronized to the CLKOUT signal if it will occur during T_2, T_3, or T_W. HIGH-to-LOW transitions of ARDY must be performed synchronously to the CPU clock.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80188, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The 80188 provides both a $\overline{\text{RES}}$ input pin and a synchronized RESET pin for use with other system components. The $\overline{\text{RES}}$ input pin on the 80188 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{\text{RES}}$ input of at least six clocks. RESET may be delayed up to two and one-half clocks behind $\overline{\text{RES}}$.

Multiple 80188 processors may be synchronized through the $\overline{\text{RES}}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{\text{RES}}$ must satisfy a 25 ns setup time before the falling edge of the

80188 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 25 ns setup time before the rising edge of the CLKOUT signal of all the processors.

LOCAL BUS CONTROLLER

The 80188 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides control lines that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80188 provides ALE, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ bus control signals. The $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals are used to strobe data from memory to the 80188 or to strobe data from the 80188 to memory. The ALE line provides a strobe to address latches for the multiplexed address/data bus. The 80188 local bus controller does not provide a memory/$\overline{\text{I/O}}$ signal. If this is required, the user will have to use the $\overline{\text{S2}}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80188 generates two control signals to be connected to 8286/8287 transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{\text{R}}$ and $\overline{\text{DEN}}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{\text{DEN}}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{\text{R}}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80188 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80188 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. This requires external circuitry to arbitrate which external device will gain control of the bus from the 80188 when there is more than one alternate local bus master. When the 80188 relinquishes control of the local bus, it floats $\overline{\text{DEN}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{S0}}$–$\overline{\text{S2}}$, $\overline{\text{LOCK}}$, AD0–AD7, A8–A19, $\overline{\text{S7}}$, and DT/$\overline{\text{R}}$ to allow another master to drive these lines directly.

The 80188 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is the highest-priority activity request which the processor may receive: higher than instruction fetching or internal DMA cycles. However, if a DMA cycle is in progress, the 80188 will complete the transfer before relinquishing the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency time can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd address to an odd address. This is a total of 16 clocks or more, if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

Local Bus Controller and Reset

Upon receipt of a RESET pulse from the $\overline{\text{RES}}$ input, the local bus controller will perform the following actions:

- Drive $\overline{\text{DEN}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ HIGH for one clock cycle, then float.

NOTE:
$\overline{\text{RD}}$ is also provided with an internal pull-up device to prevent the processor from inadvertently entering Queue Status mode during reset.

- Drive $\overline{\text{S0}}$–$\overline{\text{S2}}$ to the passive state (all HIGH) and then float.
- Drive $\overline{\text{LOCK}}$ HIGH and then float.
- Three-state AD0–7, A8–19, S7, DT/$\overline{\text{R}}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80188 integrated peripherals are controlled via 16-bit registers contained within an internal 256-byte control block. This control block may be mapped into either memory or I/O space. Internal logic will recognize the address and respond to the bus cycle. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{7-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80188 CPU at any time. The location of any register contained within the 256-byte control block is determined by the current base address of the control block.

The control block base address is programmed via a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. Note that mapping the control register block into an address range corresponding to a chip-select range is not recommended (the chip select circuitry is discussed later in this data sheet. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space, whereas if the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

Whenever mapping the 188 peripheral control block to another location, the programming of the relocation register should be done with a byte write (i.e. OUT DX,AL). Any access to the control block is done 16 bits at a time. Thus, internally, the relocation register will get written with 16 bits of the AX register while externally, the BIU will run only one 8 bit bus cycle. If a word instruction is used (i.e. OUT DX,AX), the relocation register will be written on the first bus cycle. The BIU will then run a second bus cycle which is unnecessary. The address of the second bus cycle will no longer be within the control block (i.e. the control block was moved on the first cycle), and therefore, will require the generation of an external ready signal to complete the cycle. For this reason we recommend byte operations to the relocation register. Byte instructions may also be used for the other registers in the control block and will eliminate half of the bus cycles required if a word operation had been specified. Byte operations are only valid on even addresses though, and are undefined on odd addresses.

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into slave mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register is set to 20FFH. This causes the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

The integrated 80188 peripherals operate semi-autonomously from the CPU. Access to them for the most part is via software read/write of the control and data locations in the control block. Most of these registers can be both read and written. A few dedicated lines, such as interrupts and DMA request provide real-time communication between the CPU and peripherals as in a more conventional system utilizing discrete peripheral blocks. The overall interaction and function of the peripheral blocks has not substantially changed. The data access from/to the 256-byte internal control block will always be 16-bit and done in one bus cycle. Externally the BIU will still run two bus cycles for each 16-bit operation.

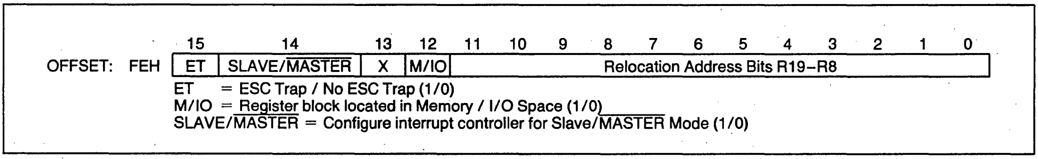
CHIP-SELECT/READY GENERATION LOGIC

The 80188 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also provide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The 80188 provides 6 memory chip select outputs for 3 address areas: upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address


15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
OFFSET: FEH
ET
SLAVE/$\overline{\text{MASTER}}$
X
M/$\overline{\text{IO}}$
Relocation Address Bits R19–R8

ET = ESC Trap / No ESC Trap (1/0)
M/$\overline{\text{IO}}$ = Register block located in Memory / I/O Space (1/0)
SLAVE/$\overline{\text{MASTER}}$ = Configure interrupt controller for Slave/MASTER Mode (1/0)

Figure 9. Relocation Register

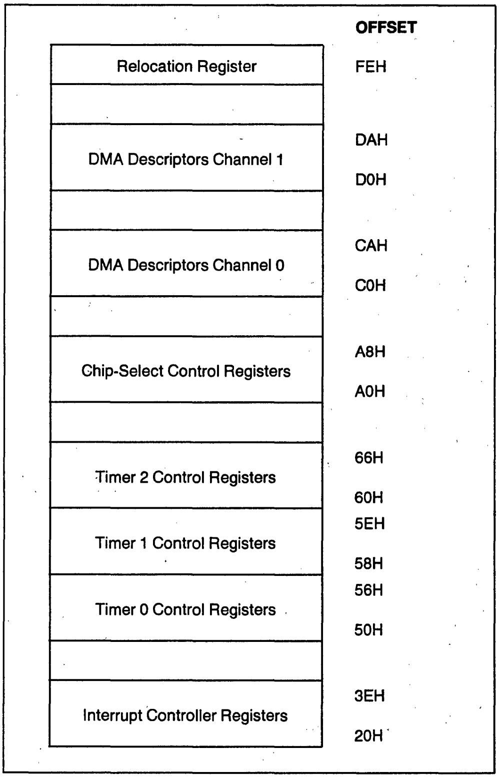

OFFSET
Relocation Register
FEH
DAH
DMA Descriptors Channel 1
D0H
CAH
DMA Descriptors Channel 0
C0H
A8H
Chip-Select Control Registers
A0H
66H
Timer 2 Control Registers
60H
5EH
Timer 1 Control Registers
58H
56H
Timer 0 Control Registers
50H
3EH
Interrupt Controller Registers
20H

Figure 10. Internal Register Map

of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes.

Upper Memory $\overline{\text{CS}}$

The 80188 provides a chip select, called $\overline{\text{UCS}}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80188 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0=R1=R2=0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

Any internally generated 20-bit address whose upper 16 bits are greater than or equal to UMCS (with bits 0–5 "0") will cause $\overline{\text{UCS}}$ to be activated. UMCS bits R2–R0 are used to specify READY mode for the area of memory defined by this chip-select register, as explained below.

Lower Memory $\overline{\text{CS}}$

The 80188 provides a chip select for low memory called $\overline{\text{LCS}}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is also defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12). This register is at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the LCS chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will cause LCS to be active. LMCS register bits R2–R0 are used to specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory CS

The 80188 provides four MCS lines which are active within a user-locatable memory block. This block can be located within the 80188 1M byte memory address space exclusive of the areas defined by UCS and LCS. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the MCS lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. Thus, if the total block size is 32K, each chip select is active for 8K of memory with MCS0 being active for the first range and MCS3 being active for the last range.

The EX and MS in MPCS relate to peripheral functionality as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each MCS line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the MCS lines will be active until both the MMCS and MPCS registers are accessed.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A0H	1	1	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 11. UMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A2H	0	0	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 12. LMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A8H	1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

Figure 13. MPCS Register

	15						9						3			0
OFFSET: A6H	U	U	U	U	U	U	U	1	1	1	1	1	1	R2	R1	R0
	A19						A13									

Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all mid-range chip selects. All devices in mid-range memory must use the same number of WAIT states.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{\text{LCS}}$ line was programmed, there would be an internal conflict between the $\overline{\text{LCS}}$ ready generation logic and the $\overline{\text{MCS}}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{\text{UCS}}$ ready generation logic. Since the $\overline{\text{LCS}}$ chip-select line does not become active until programmed, while the $\overline{\text{UCS}}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{\text{LCS}}$ range must not be programmed.

Peripheral Chip Selects

The 80188 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. This base address may be located in either memory or I/O space.

Seven $\overline{\text{CS}}$ lines called $\overline{\text{PCS0}}$–6 are generated by the 80188. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$ can also be programmed to provide latched address bits A1, A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0, A1 pins used for selecting internal registers of 8-bit peripheral chips. This scheme simplifies the hardware interface because the 8-bit registers of peripherals are simply treated as 16-bit registers located on even boundaries in I/O space or memory space where only the lower 8-bits of the register are significant: the upper 8-bits are "don't cares."

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). This register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

	15									6	5		3			0
OFFSET: A4H	U	U	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19									A10						

Figure 15. PACS Register

The user should program bits 15-6 to correspond to the desired peripheral base location. PACS bits 0-2 are used to specify READY mode for PCS0-PCS3.

Table 10. PCS Address Ranges

PCS Line	Active between Locations
PCS0	PBA —PBA+127
PCS1	PBA+128—PBA+255
PCS2	PBA+256—PBA+383
PCS3	PBA+384—PBA+511
PCS4	PBA+512—PBA+639
PCS5	PBA+640—PBA+767
PCS6	PBA+768—PBA+895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 13). This register is located at offset A8H in the internal control block. Bit 7 is used to select the function of PCS5 and PCS6, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space.
	0 = Peripherals mapped into I/O space.
EX	0 = 5 PCS lines. A1, A2 provided.
	1 = 7 PCS lines. A1, A2 are not provided.

MPCS bits 0-2 are used to specify READY mode for PCS4-PCS6 as outlined below.

READY Generation Logic

The 80188 can generate a "READY" signal internally for each of the memory or peripheral CS lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0-3 wait states for all accesses to the area for which the chip select is active. In addition, the 80188 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each CS line or group of lines generated by the 80188. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). This means, for example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2-R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2-R0 of PACS set the PCS0-3 READY mode, R2-R0 of MPCS set the PCS4-6 READY mode.

Chip Select/Ready Logic and Reset

Upon reset, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the UCS line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to allow the maximum number of internal wait states in conjunction with external Ready consideration (i.e., UMCS resets to FFFBH).
- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the PCS lines will become active.

DMA Channels

The 80188 DMA controller provides two independent DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer. Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data. This provides a data transfer rate of one MByte/sec at 8 MHz.

DMA Operation

Each channel has six registers in the control block which define each channel's specific operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit Destination pointer (2 words), a 16-bit Transfer Counter, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 17). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

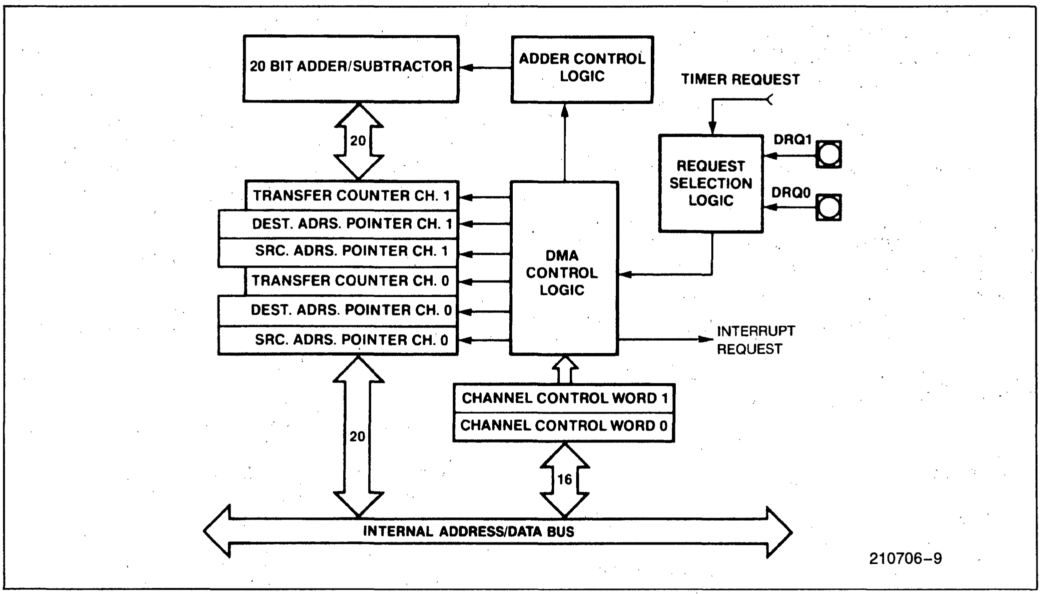


Figure 16. DMA Unit Block Diagram

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M/$\overline{\text{IO}}$	DESTINATION DEC	INC	M/$\overline{\text{IO}}$	SOURCE DEC	INC	TC	INT	SYN		P	TDRQ	X	CHG/$\overline{\text{NOCHG}}$	ST/$\overline{\text{STOP}}$	X

X = Don't Care

Figure 17. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 80188 DMA channel. This register specifies:

- the mode of synchronization;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

ST/$\overline{\text{STOP}}$: Start/stop (1/0) Channel.

CHG/$\overline{\text{NOCHG}}$: Change/Do not change (1/0) ST/$\overline{\text{STOP}}$ bit. If this bit is set when writing to the control word, the ST/$\overline{\text{STOP}}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{\text{STOP}}$ bit will not be altered. This bit is not stored; it will always be a 0 on read.

INT: Enable Interrupts to CPU on byte count termination.

TC: If set, DMA will terminate when the contents of the Transfer Count register reaches zero. The ST/$\overline{\text{STOP}}$ bit will also be reset at this point if TC is set. If this bit is cleared, the DMA unit will decrement the transfer count register for each DMA cycle, but the DMA transfer will not stop when the contents of the TC register reaches zero.

SYN: 00 No Synchronization

(2 bits) 01 Source Synchronization

NOTE:

When unsynchronized transfers are specified, the TC bit will be ignored and the ST bit will be cleared upon the transfer count reaching zero, stopping the channel.

10 Destination Synchronization

11 Unused

SOURCE:INC Increment source pointer by 1 after each transfer.

M/$\overline{\text{IO}}$ Source pointer is in M/$\overline{\text{IO}}$ space (1/0).

DEC Decrement source pointer by 1 after each transfer.

DEST: INC Increment destination pointer by 1 after each transfer.

M/$\overline{\text{IO}}$ Destination pointer is in M/$\overline{\text{IO}}$ space (1/0).

DEC Decrement destination pointer by 1 after each transfer.

P Channel priority—relative to other channel.

0 low priority.

1 high priority.

Channels will alternate cycles if both set at same priority level.

TDRQ 0: Disable DMA requests from timer 2.

1: Enable DMA requests from timer 2.

Bit 3 Bit 3 is not used.

If both INC and DEC are specified for the same pointer, the pointer will remain constant after each cycle.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18). These pointers may be individually incremented or decremented after each transfer. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). This register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or if unsynchronized transfers are programmed, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). No prefetching occurs when source synchronized or unsynchronized transfers are performed, however. Data will not be fetched from the source address until the destination device signals that it is ready to receive it. When destination synchronized transfers are requested, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another DMA cycle will begin after two processor clocks. This is done to allow the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. These lead to the maximum DMA transfer rates shown in Table 14.

Table 14. Maximum DMA Transfer Rates @ 10 MHz

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	1.25 MBytes/sec	1.25 MBytes/sec
Source Synch	1.25 MBytes/sec	1.25 MBytes/sec
Destination Synch	0.83 MBytes/sec	1.0 MBytes/sec

	15			0
HIGHER REGISTER ADDRESS	XXX	XXX	XXX	A19–A16
LOWER REGISTER ADDRESS	A15–A12	A11–A8	A7–A4	A3–A0

XXX = Don't Care

Figure 18. DMA Memory Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed such that one channel is always given priority over the other, or they may be programmed such as to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses to odd memory locations; however, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/STOP bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also have been generated. Therefore, the source and destination transfer pointers, and the transfer count register (if used) must be programmed before this bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/STOP bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The Start/Stop bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80188 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, this third timer can be used as a prescaler to the other two, or as a DMA request source.

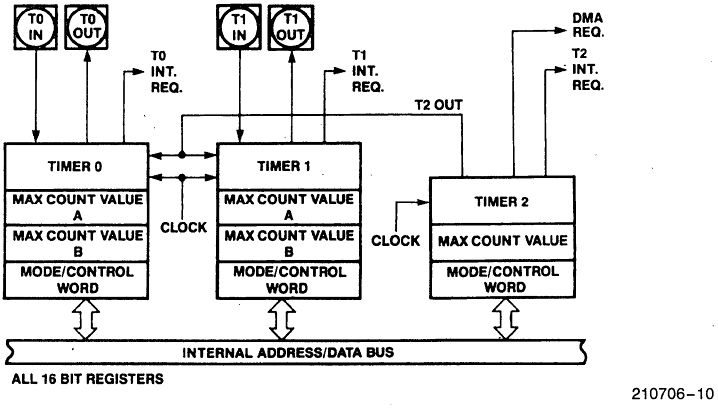


Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the internal peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values programmed by the user. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 2 clocks after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate (2 MHz for an 8 MHz CPU clock). Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

15	14	13	12	11		5	4	3	2	1	0
EN	$\overline{\text{INH}}$	INT	RIU	0	• • •	MC	RTG	P	EXT	ALT	CONT

Figure 20. Timer Mode/Control Register

ALT

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If CONT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

EXT

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80188 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

P

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

RTG

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80188 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

EN

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transitions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

$\overline{\text{INH}}$

The inhibit bit allows for selective updating of the enable (EN) bit. If $\overline{\text{INH}}$ is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If $\overline{\text{INH}}$ is a zero during the write, the EN bit will be unaffected by the operation. This bit is not stored; it will always be a 0 on a read.

INT

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller.)

MC

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set

regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts. Programmer intervention is required to clear this bit.

RIU

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The current contents of this register may be read or written by the processor at any time. If the register is written into while the timer is counting, the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. The condition which causes a timer to reset is equivalent between the current count value and the max count being used. This means that if the count is changed to be above the max count value, or if the max count value is changed to be below the current value, the timer will not reset to zero, but rather will count to its maximum value, "wrap around" to zero, then count until the max count is reached.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- All SEL (Select) bits are reset to zero. This selects MAX COUNT register A, resulting in the Timer Out pins going HIGH upon RESET.

INTERRUPT CONTROLLER

The 80188 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80188 interrupt controller has its own control register that set the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may themselves be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The 80188 has a special slave mode in which the internal interrupt controller acts as a slave to an external master. The controller is programmed into this mode by setting bit 14 in the peripheral control block relocation register. (See Slave Mode section.)

MASTER MODE OPERATION

Interrupt Controller External Interface

For external interrupt sources, five dedicated pins are provided. One of these pins is dedicated to NMI, non-maskable interrupt. This is typically used for power-fail interrupts, etc. The other four pins may function either as four interrupt input lines with internally generated interrupt vectors, as an interrupt line and an interrupt acknowledge line (called the "cascade mode") along with two other input lines with internally generated interrupt vectors, or as two interrupt input lines and two dedicated interrupt acknowledge output lines. When the interrupt lines are configured in cascade mode, the 80188 interrupt controller will not generate internal interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80188 on the second cycle. The capability to interface to external 8259A programmable interrupt controllers is thus provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in master mode are similar to the 8259A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

FULLY NESTED MODE

When in the fully nested mode four pins are used as direct interrupt requests as in Figure 22. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in-service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled without being themselves interrupted by lower-priority interrupts. Since interrupts are enabled, higher-priority interrupts will be serviced.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is issued at the end of the service routine just before the issuance of the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

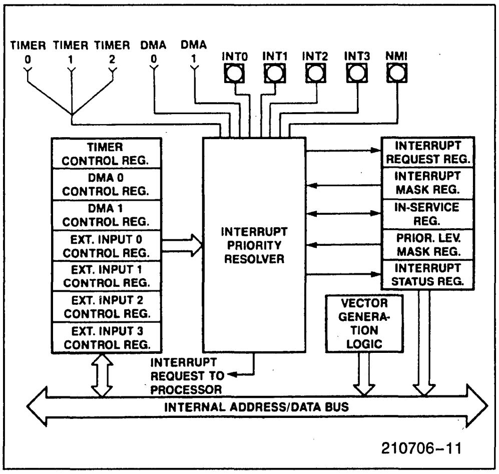


Figure 21. Interrupt Controller Block Diagram

CASCADE MODE

The 80188 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 23. INT0 is an interrupt input interfaced to an 8259A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 8259As. Three levels of priority are created, requiring priority resolution in the 80188 interrupt controller, the master 8259As, and the slave 8259As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

SPECIAL FULLY NESTED MODE

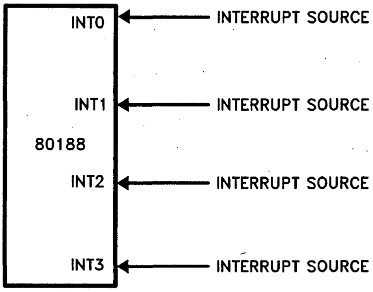
This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 8259A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80188 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80188 controller until the 80188 in-service bit is reset. In special fully nested mode, the 80188 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80188 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the external master's IS register is required to determine if there is more than one bit set. If so, the IS bit in the 80188 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 32). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80188 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.



210706–28

Figure 22. Fully Nested (Direct) Mode Interrupt Controller Connections

Master Mode Features

PROGRAMMABLE PRIORITY

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, it allows other requests to be serviced.

END-OF-INTERRUPT COMMAND

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

TRIGGER MODE

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the 80188 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

INTERRUPT VECTORING

The 80188 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 24. It contains 15 registers. All registers can both be read or written unless specified otherwise.

IN-SERVICE REGISTER

This register can be read from or written into. The format is shown in Figure 25. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command issued by the CPU.

INTERRUPT REQUEST REGISTER

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 25. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits show exactly when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Writes to to the interrupt request register will affect the D0 and D1 interrupt request bits. Setting either bit will cause the corresponding interrupt request while clearing either bit will remove the corresponding interrupt request. All other bits in the register are read-only.

MASK REGISTER

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 25. A one in a bit position corresponding to a particular source serves to mask the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

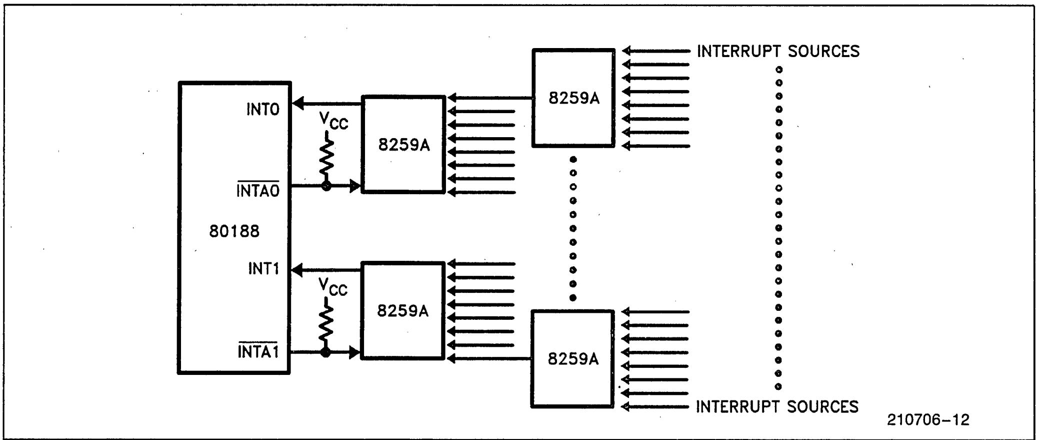


Figure 23. Cascade and Special Fully Nested Mode Interrupt Controller Connections

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 24. Interrupt Controller Registers (Master Mode)

PRIORITY MASK REGISTER

This register is used to mask all interrupts below particular interrupt priority levels. The format of this register is shown in Figure 26. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so no interrupts are masked due to priority number.

INTERRUPT STATUS REGISTER

This register contains general interrupt controller status information. The format of this register is shown in Figure 27. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. The purpose of this bit is to allow prompt service of all non-maskable interrupts. This bit may also be set by the programmer.

IRTx: These three bits represent the individual timer interrupt request bits. These bits are used to differentiate the timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt requests. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

15	14				10	9	8	7	6	5	4	3	2	1	0
0	0	•	•	•	0	0	0	I3	I2	I1	I0	D1	D0	0	TMR

Figure 25. In-Service, Interrupt Request, and Mask Register Formats

15	14											3	2	1	0
0	0	•	•	•	•	•	•	•	•	•	•	0	PRM2	PRM1	PRM0

Figure 26. Priority Mask Register Format

15	14							7	6	5	4	3	2	1	0
DHLT	0	•	•	•	•	•	•	0	0	0	0	0	IRT2	IRT1	IRT0

Figure 27. Interrupt Status Register Format (non-RMX Mode)

TIMER, DMA 0, 1; CONTROL REGISTERS

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 28. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 CONTROL REGISTERS

These registers are the control words for the four external input pins. Figure 29 shows the format of the INT0 and INT1 Control registers; Figure 30 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest priority = 000, lowest priority = 111.

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is preceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct.

SFNM: Special fully nested mode bit, 1 = SFNM.

EOI REGISTER

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 31. It initiates an EOI command when written to by the 80188 CPU.

The bits in the EOI register are encoded as follows:

S_X: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10.

NOTE:
To reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

NSPEC/: A bit that determines the type of EOI com-
SPEC mand. Nonspecific = 1, Specific = 0.

15	14										4	3	2	1	0
0	0	•	•	•	•	•	•	•	•	•	0	MSK	PR2	PR1	PR0

Figure 28. Timer/DMA Control Register Formats

15	14							7	6	5	4	3	2	1	0
0	0	•	•	•	•	•	•	0	SFNM	C	LTM	MSK	PR2	PR1	PR0

Figure 29. INT0/INT1 Control Register Formats

15	14									5	4	3	2	1	0
0	0	•	•	•	•	•	•	•	•	0	LTM	MSK	PR2	PR1	PR0

Figure 30. INT2/INT3 Control Register Formats

POLL AND POLL STATUS REGISTERS

These registers contain polling information. The format of these registers is shown in Figure 32. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_X: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

SLAVE MODE OPERATION

When slave mode is used, the internal 80188 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80188 resources will be monitored by the internal interrupt controller, while the external controller functions as the system master interrupt controller. Upon reset, the 80188 will be in master mode. To provide for slave mode operation bit 14 of the relocation register should be set.

Because of pin limitations caused by the need to interface to an external 8259A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80188 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

In slave mode each peripheral must be assigned a unique priority to ensure proper interrupt controller operation. Therefore, it is the programmer's responsibility to assign correct priorities and initialize interrupt control regisers before enable interrupts.

Slave Mode External Interface

The configuration of the 80188 with respect to an external 8259A master is shown in Figure 33. The INT0 (pin 45) input is used as the 80188 CPU interrupt input. INT3 (pin 41) functions as an output to send the 80188 slave-interrupt-request to one of the 8 master-PIC-inputs.

15	14	13								5	4	3	2	1	0
SPEC/ NSPEC	0	0	•	•	•	•	•	•	•	0	S4	S3	S2	S1	S0

Figure 31. EOI Register Format

15	14	13								5	4	3	2	1	0
INT REQ	0	0	•	•	•	•	•	•	•	0	S4	S3	S2	S1	S0

Figure 32. Poll and Poll Status Register Format

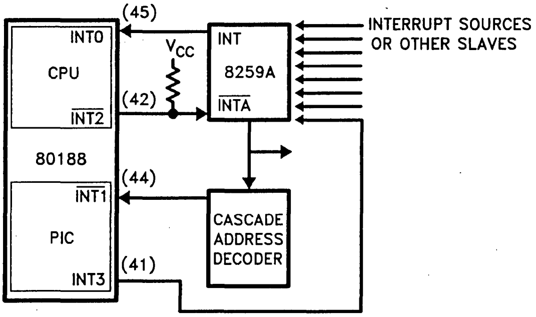


Figure 33. Slave Mode Interrupt Controller Connections

Correct master-slave interface requires decoding of the slave addresses (CAS0-2). Slave 8259As do this internally. Because of pin limitations, the 80188 slave address will have to be decoded externally. INT1 (pin 44) is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

INT2 (pin 42) is used as an acknowledge output, suitable to drive the INTA input of an 8259A.

Interrupt Nesting

Slave mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the Slave Mode

Vector generation in slave mode is exactly like that of an 8259A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In slave mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the Slave Mode

All control and command registers are located inside the internal peripheral control block. Figure 34 shows the offsets of these registers.

END-OF-INTERRUPT REGISTER

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 35. It initiates an EOI command when written by the 80188 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

IN-SERVICE REGISTER

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 36. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

INTERRUPT REQUEST REGISTER

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 36. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request. As in master mode, D0 and D1 are read/write, all other bits are read only.

MASK REGISTER

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 36. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

CONTROL REGISTERS

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 37. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 34. Interrupt Controller Registers (Slave Mode)

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	L2	L1	L0

Figure 35. Specific EOI Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 36. In-Service, Interrupt Request, and Mask Register Format

INTERRUPT VECTOR REGISTER

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 38. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

PRIORITY-LEVEL MASK REGISTER

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

INTERRUPT STATUS REGISTER

This register is defined as in master mode except that DHLT is not implemented. (See Figure 27).

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to master mode.

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	MSK	PR2	PR1	PR0

Figure 37. Control Word Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	t4	t3	t2	t1	t0	0	0	0

Figure 38. Interrupt Vector Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	m2	m1	m0

Figure 39. Priority Level Mask Register

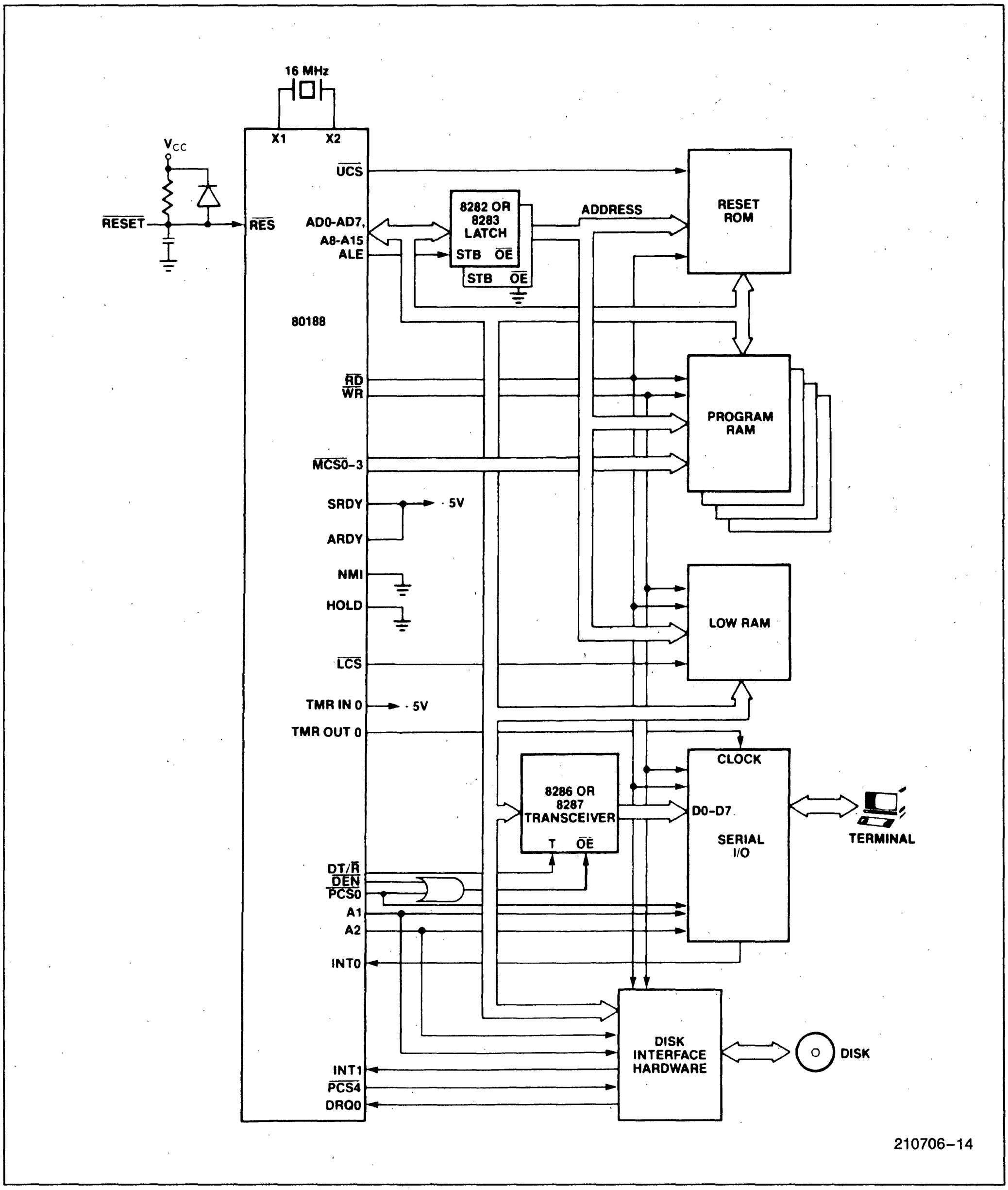


Figure 40. Typical 80188 Computer

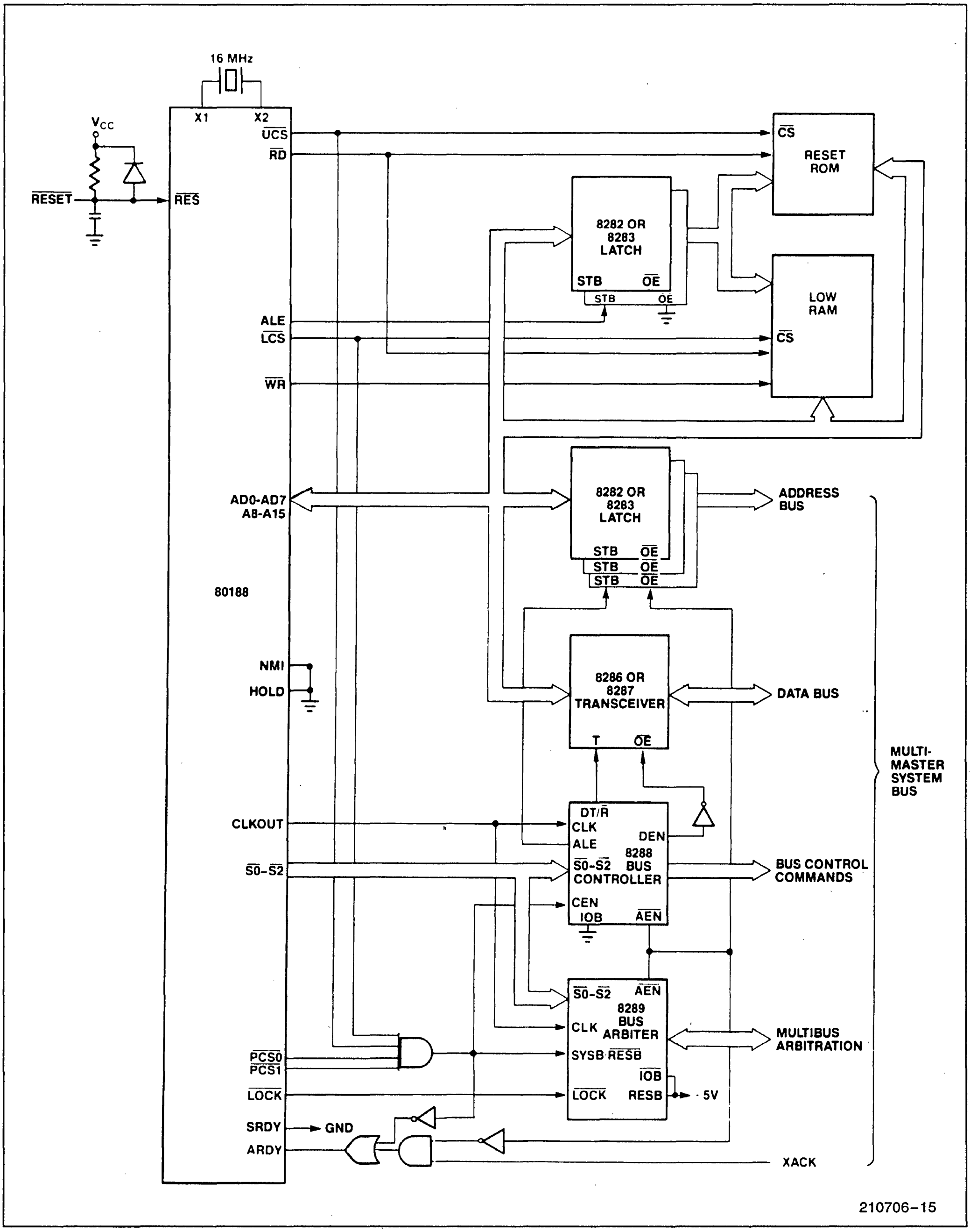


Figure 41. Typical 80188 Multi-Master Bus Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
Voltage on any Pin with Respect to Ground	−1.0V to +7V
Power Dissipation	3 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%)

Applicable to 80188 (8 MHz)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage (All except X1 and $\overline{RES}$)	2.0	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_a = 2.5 mA for $\overline{S0}$–$\overline{S2}$ I_a = 2.0 mA for all other outputs
V_{OH}	Output High Voltage	2.4		V	I_{oa} = −400 μA
I_{CC}	Power Supply Current		600*	mA	T_A = −40°C
			550	mA	T_A = 0°C
			415	mA	T_A = +70°C
I_{LI}	Input Leakage Current		±10	μA	0V < V_{IN} < V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V < V_{OUT} < V_{CC}
V_{CLO}	Clock Output Low		0.6	V	I_a = 4.0 mA
V_{CHO}	Clock Output High	4.0		V	I_{oa} = −200 μA
V_{CLI}	Clock Input Low Voltage	−0.5	0.6	V	
V_{CHI}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

*For extended temperature parts only.

PIN TIMINGS

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5V ±10%)

80188 Timing Requirements All Timings Measured At 1.5 Volts Unless Otherwise Noted

Symbol	Parameter	80188 (8 MHz)		Units	Test Conditions
		Min	Max		
T_{DVCL}	Data in Setup (A/D)	20		ns	
T_{CLDX}	Data in Hold (A/D)	10		ns	
T_{ARYHCH}	Asynchronous Ready (ARDY) active setup time*	20		ns	
T_{ARYLCL}	ARDY inactive setup time	35		ns	
T_{CLARX}	ARDY hold time	15		ns	
T_{ARYCHL}	Asynchronous Ready inactive hold time	15		ns	
T_{SRYCL}	Synchronous Ready (SRDY) Transition Setup Time	20		ns	
T_{CLSRY}	SRDY Transition Hold Time	15		ns	
T_{HVCL}	HOLD Setup*	25		ns	
T_{INVCH}	INTR, NMI, $\overline{TEST}$, TMR IN, Setup*	25		ns	
T_{INVCL}	DRQ0, DRQ1, Setup*	25		ns	

80188 Master Interface Timing Responses

T_{CLAV}	Address Valid Delay	5	55	ns	C_L = 20–200 pF all outputs (except T_{CLTMV}) @ 8 MHz
T_{CLAX}	Address Hold	10		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	35	ns	
T_{CHCZ}	Command Lines Float Delay		45	ns	
T_{CHCV}	Command Lines Valid Delay (after float)		55	ns	
T_{LHLL}	ALE Width	$T_{CLCL}-35$		ns	
T_{CHLH}	ALE Active Delay		35	ns	
T_{CHLL}	ALE Inactive Delay		35	ns	
T_{LLAX}	Address Hold to ALE Inactive	$T_{CHCL}-25$		ns	
T_{CLDV}	Data Valid Delay	10	44	ns	
T_{CLDOX}	Data Hold Time	10		ns	
T_{WHDX}	Data Hold after WR	$T_{CLCL}-40$		ns	
T_{CVCTV}	Control Active Delay 1	5	50	ns	
T_{CHCTV}	Control Active Delay 2	10	55	ns	
T_{CVCTX}	Control Inactive Delay	5	55	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)	10	70	ns	

*To guarantee recognition at next clock.

PIN TIMINGS (Continued)

A.C. CHARACTERISTICS

(T_A = 0°C to +70°C, V_{CC} = 5V ±10%) (Continued)

80188 Master Interface Timing Responses (Continued)

Symbol	Parameter	80188 (8 MHz)		Units	Test Conditions
		Min	Max		
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		ns	
T_{CLRL}	$\overline{RD}$ Active Delay	10	70	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	10	55	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active	$T_{CLCL}-40$		ns	
T_{CLHAV}	HLDA Valid Delay	5	50	ns	
T_{RLRH}	$\overline{RD}$ Width	$2T_{CLCL}-50$		ns	
T_{WLWH}	$\overline{WR}$ Width	$2T_{CLCL}-40$		ns	
T_{AVLL}	Address Valid to ALE Low	$T_{CLCH}-25$		ns	
T_{CHSV}	Status Active Delay	10	55	ns	
T_{CLSH}	Status Inactive Delay	10	65	ns	
T_{CLTMV}	Timer Output Delay		60	ns	100 pF max
T_{CLRO}	Reset Delay		60	ns	
T_{CHQSV}	Queue Status Delay		35	ns	
T_{CHDX}	Status Hold Time	10		ns	
T_{AVCH}	Address Valid to Clock High	10		ns	
T_{CLLV}	$\overline{LOCK}$ Valid/Invalid Delay	5	65	ns	

80188 Chip-Select Timing Responses

T_{CLCSV}	Chip-Select Active Delay		66	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	35		ns	
T_{CHCSX}	Chip-Select Inactive Delay	5	35	ns	

80188 CLKIN Requirements

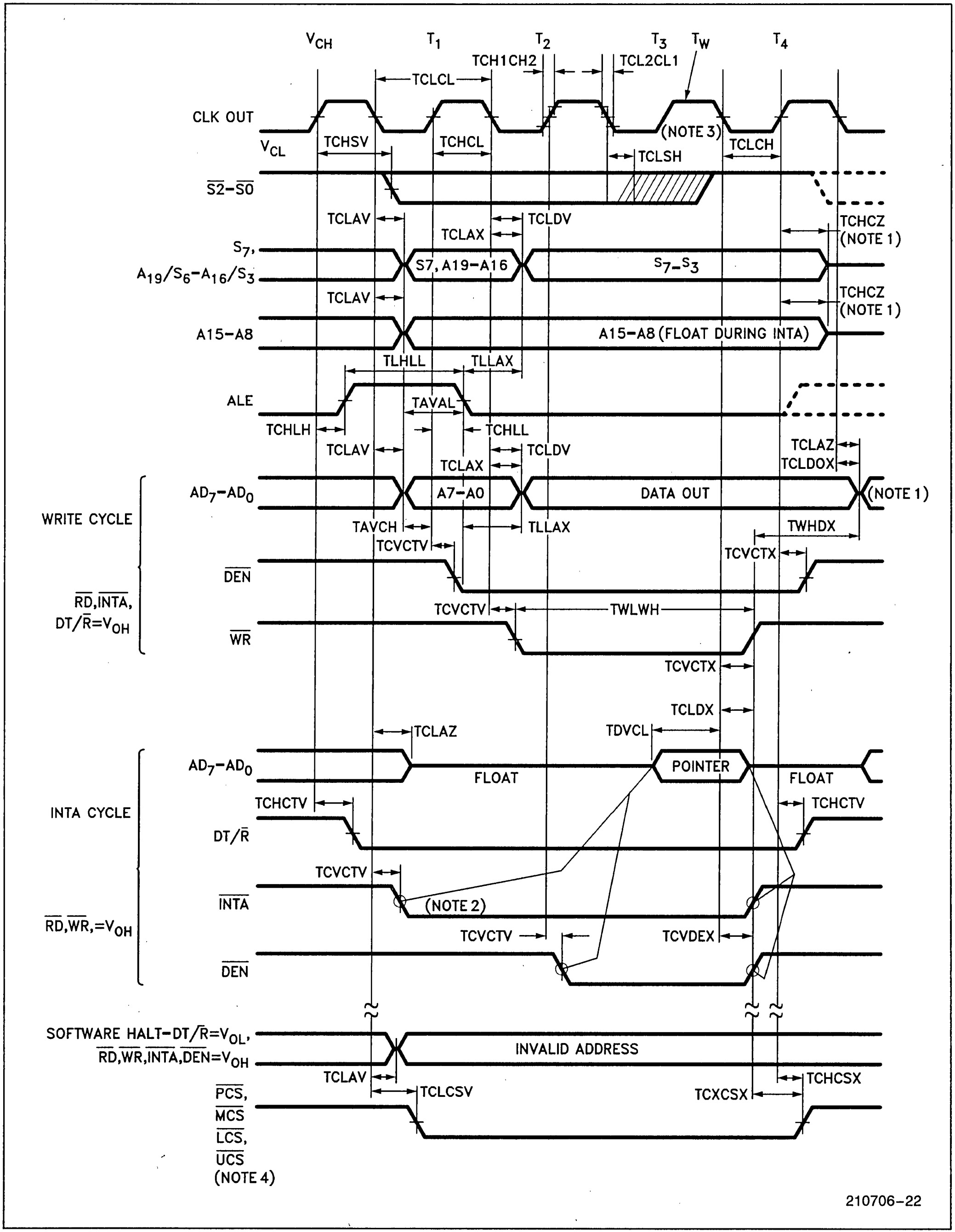
T_{CKIN}	CLKIN Period	62.5	250	ns	
T_{CKHL}	CLKIN Fall Time		10	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		10	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	25		ns	1.5V
T_{CHCK}	CLKIN High Time	25		ns	1.5V

80188 CLKOUT Timing (200 pF load)

T_{CICO}	CLKIN to CLKOUT Skew		50	ns	
T_{CLCL}	CLKOUT Period	125	500	ns	
T_{CLCH}	CLKOUT Low Time	$\frac{1}{2} T_{CLCL}-7.5$		ns	1.5V
T_{CHCL}	CLKOUT High Time	$\frac{1}{2} T_{CLCL}-7.5$		ns	1.5V
T_{CH1CH2}	CLKOUT Rise Time		15	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		15	ns	3.5 to 1.0V

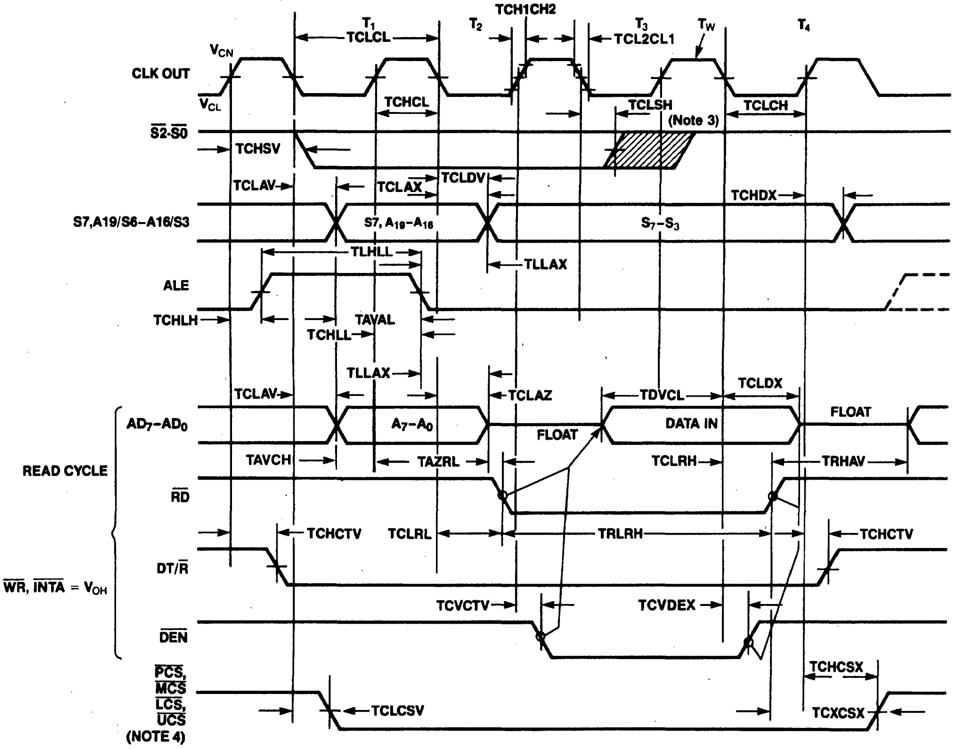
WAVEFORMS

Major Cycle Timing



WAVEFORMS (Continued)

Major Cycle Timing (Continued)

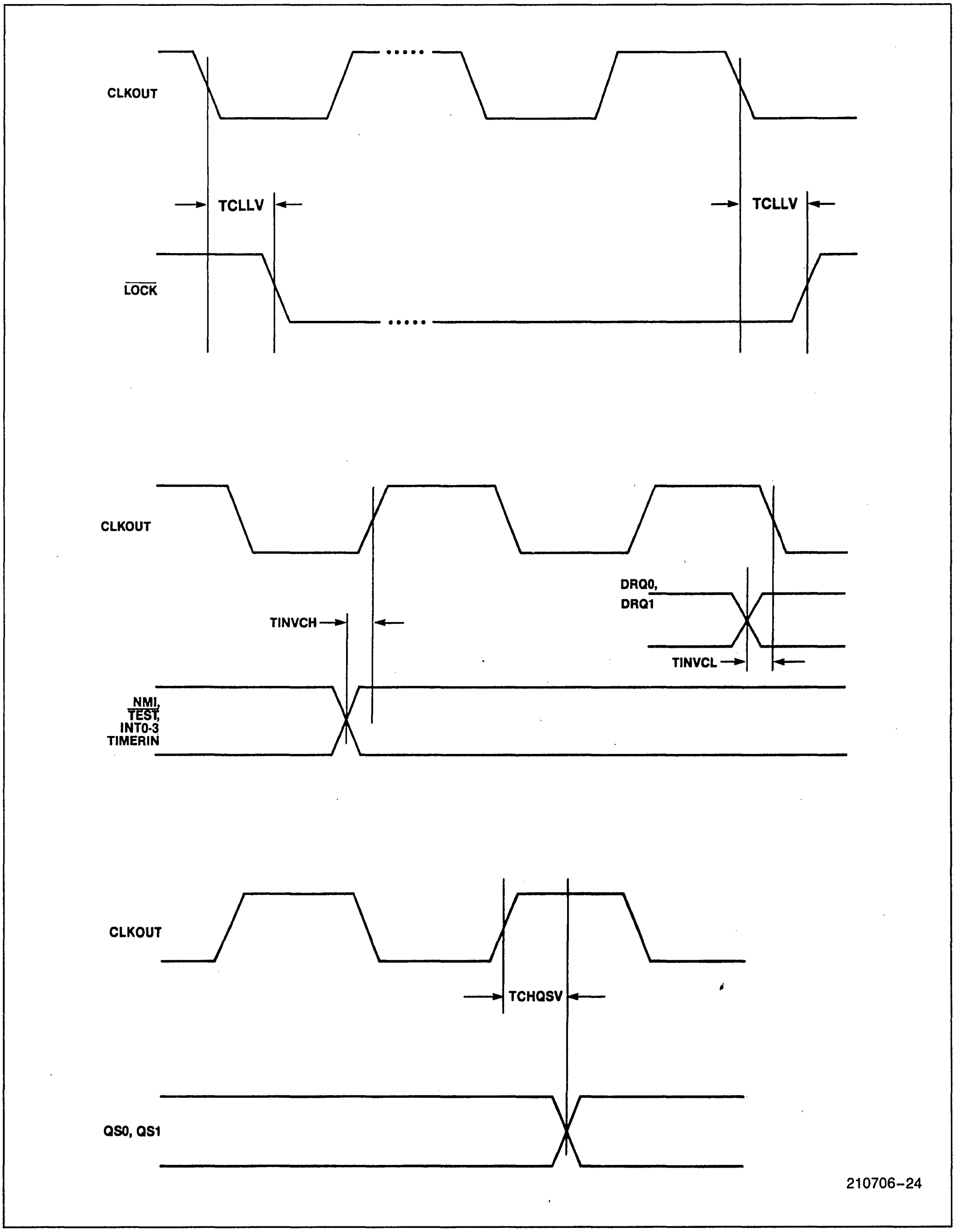


210706-23

NOTES:

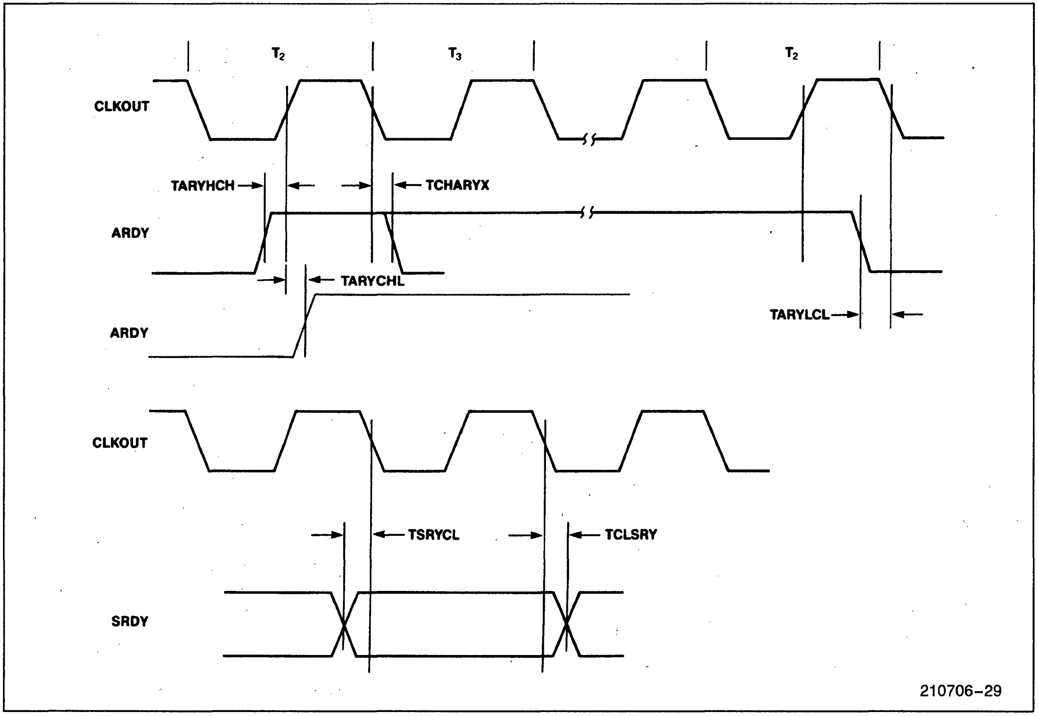
1. Following a Write cycle, the Local Bus is floated by the 80188 only when the 80188 enters a "Hold Acknowledge" state.
2. INTA occurs one clock later in Slave Mode.
3. Status inactive just prior to T_4.
4. Latched A1 and A2 have same timings as $\overline{PCS5}$ and $\overline{PCS6}$.

WAVEFORMS (Continued)

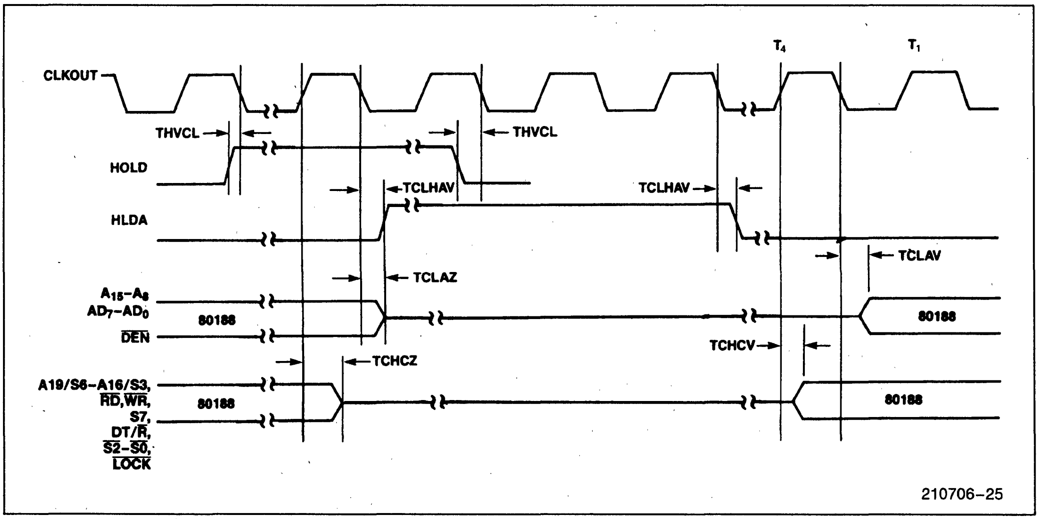


WAVEFORMS (Continued)

READY TIMING

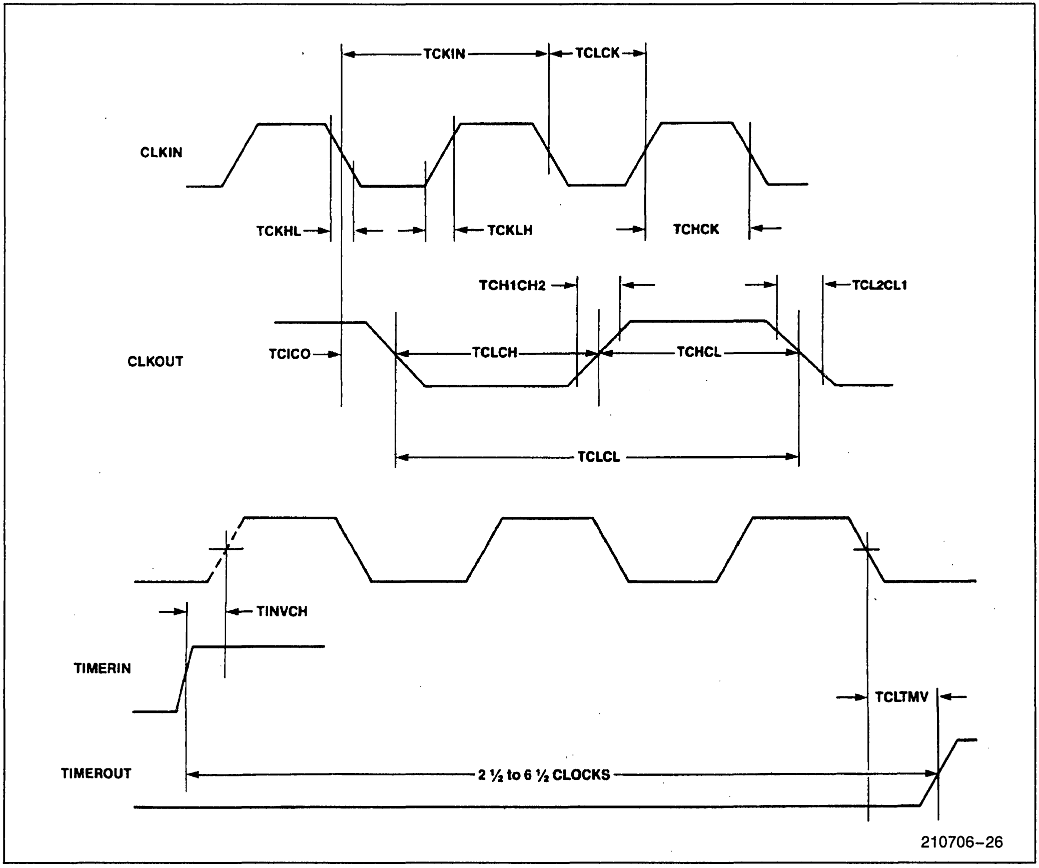


HOLD-HLDA TIMING



WAVEFORMS (Continued)

Timer On 80188



80188 EXECUTION TIMINGS

Since the bus interface unit and execution unit operate independently, a determination of 80188 program execution timing must consider both the bus cycles necessary to prefetch instructions as well as the number of execution unit cycles necessary to execute instructions. The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDs occur.

All instructions which involve memory accesses can also require one or two additional clocks above the minimum timings shown due to the asynchronous handshake between the BIU and execution unit.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

The 80188 8-bit BIU is noticeably limited in its performance relative to the execution unit. A sufficient number of prefetched bytes may not reside in the prefetch queue much of the time. Therefore, actual program execution may be substantially greater than that derived from adding the instruction timings shown.

INSTRUCTION SET SUMMARY

Function	Format				Clock Cycles	Comments
DATA TRANSFER						
MOV = Move:						
Register to register/memory	1 0 0 0 1 0 0 w	mod reg r/m			2/12*	
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m			2/9*	
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w=1	12/13*	8/16-bit
Immediate to register	1 0 1 1 w reg	data	data if w=1		3/4	8/16-bit
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high		8*	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high		9*	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m			2/13	
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m			2/15	
PUSH = Push:						
Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m			20	
Register	0 1 0 1 0 reg				14	
Segment register	0 0 0 reg 1 1 0				13	
Immediate	0 1 1 0 1 0 s 0	data	data if s=0		14	
PUSHA = Push All	0 1 1 0 0 0 0 0				68	
POP = Pop:						
Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m			24	
Register	0 1 0 1 1 reg				14	
Segment register	0 0 0 reg 1 1 1	(reg≠01)			12	
POPA = Pop All	0 1 1 0 0 0 0 1				83	
XCHG = Exchange:						
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m			4/17*	
Register with accumulator	1 0 0 1 0 reg				3	
IN = Input from:						
Fixed port	1 1 1 0 0 1 0 w	port			10*	
Variable port	1 1 1 0 1 1 0 w				8*	
OUT = Output to:						
Fixed port	1 1 1 0 0 1 1 w	port			9*	
Variable port	1 1 1 0 1 1 1 w				7*	
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1				15	
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m			6	
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m	(mod≠11)		26	
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m	(mod≠11)		26	
LAHF = Load AH with flags	1 0 0 1 1 1 1 1				2	
SAHF = Store AH into flags	1 0 0 1 1 1 1 0				3	
PUSHF = Push flags	1 0 0 1 1 1 0 0				13	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
DATA TRANSFER (Continued)						
POPF = Pop flags	10011101				12	
SEGMENT = **Segment Override:**						
CS	00101110				2	
SS	00110110				2	
DS	00111110				2	
ES	00100110				2	
ARITHMETIC						
ADD = **Add:**						
Reg/memory with register to either	000000dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 0 0 0 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0000010w	data	data if w=1		3/4	8/16-bit
ADC = **Add with carry:**						
Reg/memory with register to either	000100dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 0 1 0 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0001010w	data	data if w=1		3/4	8/16-bit
INC = **Increment:**						
Register/memory	1111111w	mod 0 0 0 r/m			3/15*	
Register	01000 reg				3	
SUB = **Subtract:**						
Reg/memory and register to either	001010dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 1 0 1 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0010110w	data	data if w=1		3/4	8/16-bit
SBB = **Subtract with borrow:**						
Reg/memory and register to either	000110dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 0 1 1 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0001110w	data	data if w=1		3/4	8/16-bit
DEC = **Decrement:**						
Register/memory	1111111w	mod 0 0 1 r/m			3/15*	
Register	01001 reg				3	
CMP = **Compare:**						
Register/memory with register	0011101w	mod reg r/m			3/10*	
Register with register/memory	0011100w	mod reg r/m			3/10*	
Immediate with register/memory	100000sw	mod 1 1 1 r/m	data	data if s w=01	3/10*	
Immediate with accumulator	0011110w	data	data if w=1		3/4	8/16-bit
NEG = Change sign register/memory	1111011w	mod 0 1 1 r/m			3/10*	
AAA = ASCII adjust for add	00110111				8	
DAA = Decimal adjust for add	00100111				4	
AAS = ASCII adjust for subtract	00111111				7	
DAS = Decimal adjust for subtract	00101111				4	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
ARITHMETIC (Continued)						
MUL = Multiply (unsigned):	1111011w	mod 100 r/m				
Register-Byte					26–28	
Register-Word					35–37	
Memory-Byte					32–34	
Memory-Word					41–43*	
IMUL = Integer multiply (signed):	1111011w	mod 101 r/m				
Register-Byte					25–28	
Register-Word					34–37	
Memory-Byte					31–34	
Memory-Word					40–43*	
IMUL = Integer Immediate multiply (signed)	011010s1	mod reg r/m	data	data if s=0	22–25/ 29–32	
DIV = Divide (unsigned):	1111011w	mod 110 r/m				
Register-Byte					29	
Register-Word					38	
Memory-Byte					35	
Memory-Word					44*	
IDIV = Integer divide (signed):	1111011w	mod 111 r/m				
Register-Byte					44–52	
Register-Word					53–61	
Memory-Byte					50–58	
Memory-Word					59–67*	
AAM = ASCII adjust for multiply	11010100	00001010			19	
AAD = ASCII adjust for divide	11010101	00001010			15	
CBW = Convert byte to word	10011000				2	
CWD = Convert word to double word	10011001				4	
LOGIC **Shift/Rotate Instructions:**						
Register/Memory by 1	1101000w	mod TTT r/m			2/15	
Register/Memory by CL	1101001w	mod TTT r/m			5+n/17+n	
Register/Memory by Count	1100000w	mod TTT r/m	count		5+n/17+n	
		TTT Instruction				
		000 ROL				
		001 ROR				
		010 RCL				
		011 RCR				
		100 SHL/SAL				
		101 SHR				
		111 SAR				
AND = And:						
Reg/memory and register to either	001000dw	mod reg r/m			3/10*	
Immediate to register/memory	1000000w	mod 100 r/m	data	data if w=1	4/16*	
Immediate to accumulator	0010010w	data	data if w=1		3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
LOGIC (Continued)						
TEST = And function to flags, no result:						
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m			3/10*	
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1	4/10*	
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w = 1		3/4	8/16-bit
OR = Or:						
Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m			3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1	4/16*	
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w = 1		3/4	8/16-bit
XOR = Exclusive or:						
Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m			3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1	4/16*	
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w = 1		3/4	8/16-bit
NOT = Invert register/memory	1 1 1 1 0 1 1 w	mod 0 1 0 r/m			3/10*	
STRING MANIPULATION:						
MOVS = Move byte/word	1 0 1 0 0 1 0 w				14*	
CMPS = Compare byte/word	1 0 1 0 0 1 1 w				22*	
SCAS = Scan byte/word	1 0 1 0 1 1 1 w				15*	
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w				12*	
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w				10*	
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w				14	
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w				14	
Repeated by count in CX						
MOVS = Move string	1 1 1 1 0 0 1 0	1 0 1 0 0 1 0 w			8+8n*	
CMPS = Compare string	1 1 1 1 0 0 1 z	1 0 1 0 0 1 1 w			5+22n*	
SCAS = Scan string	1 1 1 1 0 0 1 z	1 0 1 0 1 1 1 w			5+15n*	
LODS = Load string	1 1 1 1 0 0 1 0	1 0 1 0 1 1 0 w			6+11n*	
STOS = Store string	1 1 1 1 0 0 1 0	1 0 1 0 1 0 1 w			6+9n*	
INS = Input string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 0 w			8+8n*	
OUTS = Output string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 1 w			8+8n*	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
CONTROL TRANSFER						
CALL = Call:						
Direct within segment	11101000	disp-low	disp-high		19	
Register/memory indirect within segment	11111111	mod 010 r/m			17/27	
Direct intersegment	10011010	segment offset			31	
		segment selector				
Indirect intersegment	11111111	mod 011 r/m		(mod ≠ 11)	54	
JMP = Unconditional jump:						
Short/long	11101011	disp-low			14	
Direct within segment	11101001	disp-low	disp-high		14	
Register/memory indirect within segment	11111111	mod 100 r/m			11/21	
Direct intersegment	11101010	segment offset			14	
		segment selector				
Indirect intersegment	11111111	mod 101 r/m		(mod ≠ 11)	34	
RET = Return from CALL:						
Within segment	11000011				20	
Within seg adding immed to SP	11000010	data-low	data-high		22	
Intersegment	11001011				30	
Intersegment adding immediate to SP	11001010	data-low	data-high		33	
JE/JZ = Jump on equal/zero	01110100	disp			4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100	disp			4/13	
JLE/JNG = Jump on less or equal/not greater	01111110	disp			4/13	
JB/JNAE = Jump on below/not above or equal	01110010	disp			4/13	
JBE/JNA = Jump on below or equal/not above	01110110	disp			4/13	
JP/JPE = Jump on parity/parity even	01111010	disp			4/13	
JO = Jump on overflow	01110000	disp			4/13	
JS = Jump on sign	01111000	disp			4/13	
JNE/JNZ = Jump on not equal/not zero	01110101	disp			4/13	
JNL/JGE = Jump on not less/greater or equal	01111101	disp			4/13	
JNLE/JG = Jump on not less or equal/greater	01111111	disp			4/13	
JNB/JAE = Jump on not below/above or equal	01110011	disp			4/13	
JNBE/JA = Jump on not below or equal/above	01110111	disp			4/13	
JNP/JPO = Jump on not par/par odd	01111011	disp			4/13	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
CONTROL TRANSFER (Continued)			
JNO = Jump on not overflow	01110001 disp	4/13	
JNS = Jump on not sign	01111001 disp	4/13	
JCXZ = Jump on CX zero	11100011 disp	5/15	
LOOP = Loop CX times	11100010 disp	6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001 disp	6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000 disp	6/16	
ENTER = Enter Procedure	11001000 data-low data-high L		
L = 0		15	
L = 1		25	
L > 1		22+16(n−1)	
LEAVE = Leave Procedure	11001001	8	
INT = **Interrupt:**			
Type specified	11001101 type	47	
Type 3	11001100	45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	11001110	48/4	
IRET = Interrupt return	11001111	28	
BOUND = Detect value out of range	01100010 mod reg r/m	33-35	
PROCESSOR CONTROL			
CLC = Clear carry	11111000	2	
CMC = Complement carry	11110101	2	
STC = Set carry	11111001	2	
CLD = Clear direction	11111100	2	
STD = Set direction	11111101	2	
CLI = Clear interrupt	11111010	2	
STI = Set interrupt	11111011	2	
HLT = Halt	11110100	2	
WAIT = Wait	10011011	6	if $\overline{\text{test}}$ = 0
LOCK = Bus lock prefix	11110000	2	
ESC = Processor Extension Escape	11011TTT mod LLL r/m (TTT LLL are opcode to processor extension)	6	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field

if mod = 00 then DISP = 0*, disp-low and disp-high are absent

if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent

if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP

if r/m = 001 then EA = (BX) + (DI) + DISP

if r/m = 010 then EA = (BP) + (SI) + DISP

if r/m = 011 then EA = (BP) + (DI) + DISP

if r/m = 100 then EA = (SI) + DISP

if r/m = 101 then EA = (DI) + DISP

if r/m = 110 then EA = (BP) + DISP*

if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	**Segment Register**
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	**8-Bit (w = 0)**
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

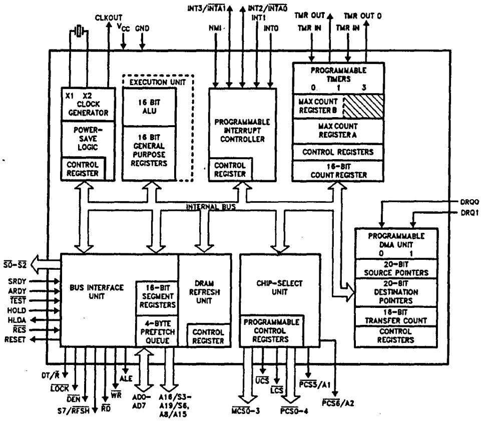
ADVANCE INFORMATION

80C188
CHMOS HIGH INTEGRATION 16-BIT MICROPROCESSOR

- **Operation Modes Include:**
  - **—Enhanced Mode Which Has**
    - **—DRAM Refresh**
    - **—Power-Save Logic**
  - **—Compatible Mode**
    - **—NMOS 80188 Pin for Pin Replacement for Non-Numerics Applications**
- **Integrated Feature Set**
  - **—Enhanced 80C86/C88 CPU**
  - **—Clock Generator**
  - **—2 Independent DMA Channels**
  - **—Programmable Interrupt Controller**
  - **—3 Programmable 16-Bit Timers**
  - **—Dynamic RAM Refresh Control Unit**
  - **—Programmable Memory and Peripheral Chip Select Logic**
  - **—Programmable Wait State Generator**
  - **—Local Bus Controller**
  - **—Power Save Logic**
  - **—System-Level Testing Support (High Impedance Test Mode)**
- **Available in 16 MHz (80C188-16), 12.5 MHz (80C188-12) and 10 MHz (80C188-10) Versions**
- **Direct Addressing Capability to 1 MByte and 64 KByte I/O**
- **Completely Object Code Compatible with All Existing 8086/8088 Software and Also Has 10 Additional Instructions over 8086/8088**
- **Complete System Development Support**
  - **—All 8088 and NMOS 80188 Software Development Tools Can Be Used for 80C186 System Development**
    - **—Assembler, PL/M, Pascal, Fortran, and System Utilities**
    - **—In-Circuit-Emulator (ICE™-188)**
- **Available in 68 Pin:**
  - **—Plastic Leaded Chip Carrier (PLCC)**
  - **—Ceramic Pin Grid Array (PGA)**
  - **—Ceramic Leadless Chip Carrier (JEDEC A Package)**

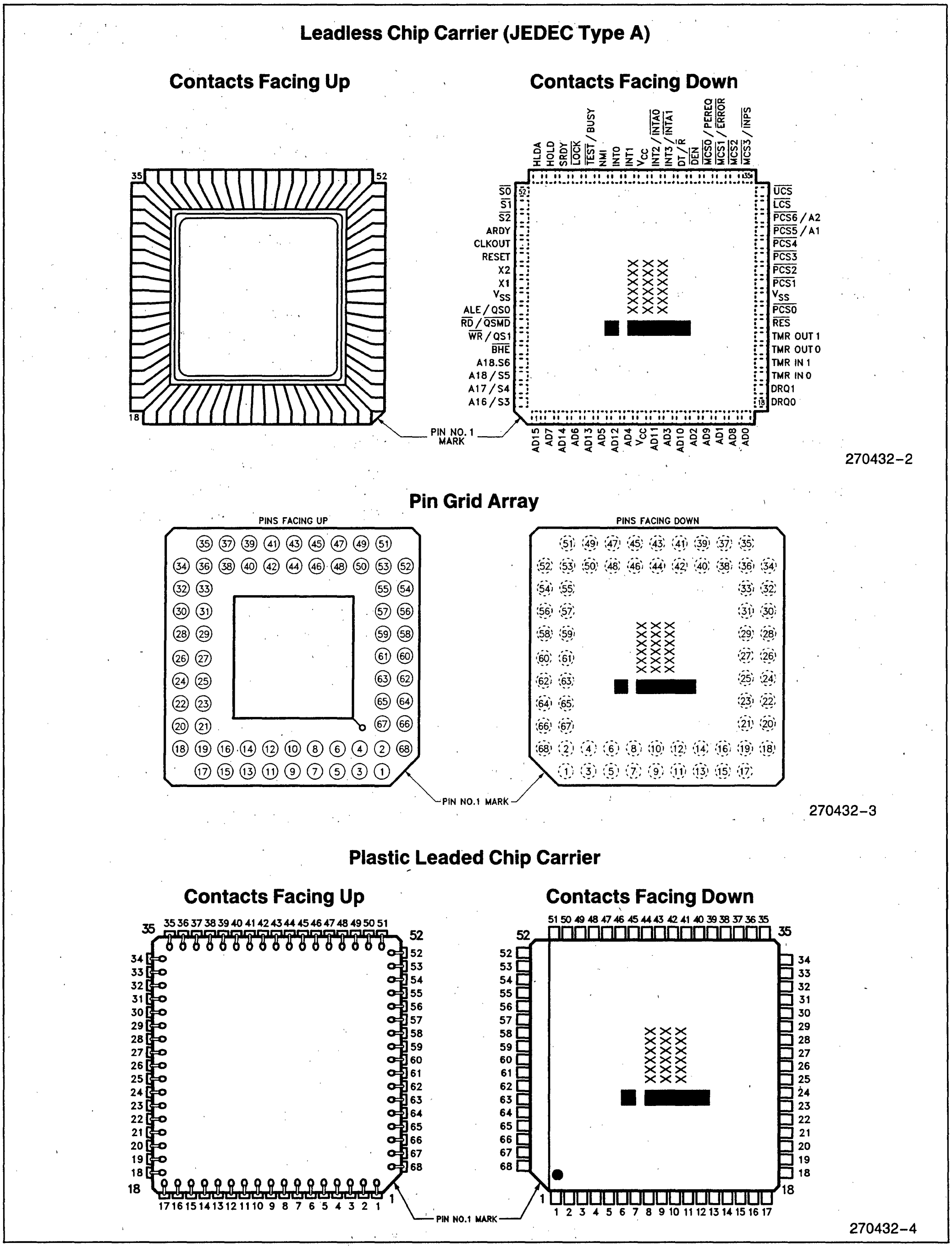
  (See Packaging Outlines and Dimensions, Order Number 231369)
- **Available in EXPRESS:**
  - **—Standard Temperature with Burn-In**
  - **—Extended Temperature Range (−40°C to +85°C)**

The Intel 80C188 is a CHMOS high integration microprocessor. It has features which are new to the 80186 family which include a DRAM refresh control unit, power-save mode and a direct numerics interface. When used in "compatible" mode, the 80C188 is 100% pin-for-pin compatible with the NMOS 80188 (except for 8087 applications). The "enhanced" mode of operation allows the full feature set of the 80C188 to be used. The 80C188 is upward compatible with 8086 and 8088 software and fully compatible with 80186 and 80188 software.



270432-1

Figure 1. 80C188 Block Diagram



Leadless Chip Carrier (JEDEC Type A)

Contacts Facing Up

Contacts Facing Down

270432-2

Pin Grid Array

270432-3

Plastic Leaded Chip Carrier

Contacts Facing Up

Contacts Facing Down

270432-4

Figure 2. 80C188 Pinout Diagrams

Table 1. 80C188 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}, V_{CC}	9, 43	I	System Power: +5 volt power supply.
V_{SS}, V_{SS}	26, 60	I	System Ground.
RESET	57	O	Reset Output indicates that the 80C188 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{\text{RES}}$ signal. Reset goes inactive 2 clockout periods after $\overline{\text{RES}}$ goes inactive. When tied to the $\overline{\text{TEST}}$ pin, Reset forces the 80C188 into enhanced mode.
X1, X2	59, 58	I	Crystal Inputs X1 and X2 provide external connections for a fundamental mode or third overtone parallel resonant crystal for the internal oscillator. X1 can connect to an external clock instead of a crystal. In this case, minimize the capacitance on X2 or drive X2 with complemented X1. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	Clock Output provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT.
$\overline{\text{RES}}$	24	I	System Reset causes the 80C188 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80C188 clock. The 80C188 begins fetching instructions approximately 7 clock cycles after $\overline{\text{RES}}$ is returned HIGH. For proper initialization, V_{CC} must be within specifications and the clock signal must be stable for more than 4 clocks with $\overline{\text{RES}}$ held LOW. $\overline{\text{RES}}$ is internally synchronized. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{\text{RES}}$ generation via an RC network. When $\overline{\text{RES}}$ occurs, the 80C188 will drive the status lines to an inactive level for one clock, and then float them.
$\overline{\text{TEST}}$	47	I	The $\overline{\text{TEST}}$ pin is sampled during and after reset to determine whether the 80C188 is to enter Compatible or Enhanced Mode. Enhanced Mode requires $\overline{\text{TEST}}$ to be HIGH on the rising edge of $\overline{\text{RES}}$ and LOW four clocks later. Any other combination will place the 80C188 in Compatible Mode. A weak internal pullup insures a HIGH state when the pin is not driven. This pin is examined by the WAIT instruction. If the $\overline{\text{TEST}}$ input is HIGH when WAIT execution begins, instruction execution will suspend. $\overline{\text{TEST}}$ will be resampled every five clocks until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80C188 is waiting for $\overline{\text{TEST}}$, interrupts will be serviced.
TMR IN 0, TMR IN 1	20 21	I I	Timer Inputs are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0, TMR OUT 1	22 23	O O	Timer outputs are used to provide single pulse or continous waveform generation, depending upon the timer mode selected.

Table 1. 80C188 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>DRQ0
DRQ1</td><td>18
19</td><td>I
I</td><td>DMA Request is driven HIGH by an external device when it desires that a DMA channel (Channel 0 or 1) perform a transfer. These signals are active HIGH, level-triggered, and internally synchronized.</td></tr>
<tr><td>NMI</td><td>46</td><td>I</td><td>Non-Maskable Interrupt is an edge-triggered input which causes a type 2 interrupt. NMI is not maskable internally. A transition from a LOW to HIGH initiates the interrupt at the next instruction boundary. NMI is latched internally. An NMI duration of one clock or more will guarantee service. This input is internally synchronized.</td></tr>
<tr><td>INT0, INT1
INT2/$\overline{\text{INTA0}}$
INT3/$\overline{\text{INTA1}}$</td><td>45, 44
42
41</td><td>I
I/O
I/O</td><td>Maskable Interrupt Requests can be requested by activating one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured via software to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured via software to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When slave mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).</td></tr>
<tr><td>A19/S6,
A18/S5,
A17/S4,
A16/S3</td><td>65
66
67
68</td><td>O
O
O
O</td><td>Address Bus Outputs (16–19) and Bus Cycle Status (3–6) reflect the four most significant address bits during T$_1$. These signals are active HIGH. During T$_2$, T$_3$, T$_W$, and T$_4$, status information is available on these lines as encoded below:
<table>
<tr><th></th><th>Low</th><th>High</th></tr>
<tr><td>S6</td><td>Processor Cycle</td><td>DMA Cycle</td></tr>
</table>
S3, S4, and S5 are defined as LOW during T$_2$–T$_4$.</td></tr>
<tr><td>A15–A8</td><td>1, 3, 5, 7,
10, 12, 14, 16</td><td>O</td><td>Address-Only Bus (15–8) contains valid addresses from T$_1$–T$_4$. The bus is active high.</td></tr>
<tr><td>AD7–AD0</td><td>2, 4, 6, 8,
11, 13, 16, 17</td><td>I/O</td><td>Address/Data Bus (7–0) signals constitute the time multiplexed memory or I/O address (T$_1$) and data (T$_2$, T$_3$, T$_W$, and T$_4$) bus. The bus is active high.</td></tr>
<tr><td>S7/$\overline{\text{RFSH}}$</td><td>64</td><td>O</td><td>In compatible mode, S7 is high to signify that the 80C188 has an 8-bit bus except during bus HOLD at which time the pin floats.
In Enhanced Mode, S7 will become S7/$\overline{\text{RFSH}}$ in order to signify DRAM refresh cycles. A refresh cycle is indicated by S7/$\overline{\text{RFSH}}$ being LOW.</td></tr>
</table>

Table 1. 80C186 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>ALE/QS0</td><td>61</td><td>O</td><td>Address Latch Enable/Queue Status 0 is provided by the 80C188 to latch the address. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the standard 8088. The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 8088. Note that ALE is never floated.</td></tr>
<tr><td>$\overline{\text{WR}}$/QS1</td><td>63</td><td>O</td><td>Write Strobe/Queue Status 1 indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{\text{WR}}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during "HOLD." It is driven HIGH for one clock during Reset, and then floated. When the 80C188 is in queue status mode, the ALE/QS0 and $\overline{\text{WR}}$/QS1 pins provide information about processor/instruction queue interaction.
<table>
<tr><th>QS1</th><th>QS0</th><th>Queue Operation</th></tr>
<tr><td>0</td><td>0</td><td>No queue operation</td></tr>
<tr><td>0</td><td>1</td><td>First opcode byte fetched from the queue</td></tr>
<tr><td>1</td><td>1</td><td>Subsequent byte fetched from the queue</td></tr>
<tr><td>1</td><td>0</td><td>Empty the queue</td></tr>
</table>
</td></tr>
<tr><td>$\overline{\text{RD}}$/$\overline{\text{QSMD}}$</td><td>62</td><td>O</td><td>Read Strobe indicates that the 80C188 is performing a memory or I/O read cycle. $\overline{\text{RD}}$ is active LOW for T_2, T_3, and T_W of any read cycle. It is guaranteed not to go LOW in T_2 until after the Address Bus is floated. $\overline{\text{RD}}$ is active LOW, and floats during "HOLD". $\overline{\text{RD}}$ is driven HIGH for one clock during Reset, and then the output driver is floated. A weak internal pull-up mechanism of the $\overline{\text{RD}}$ line holds it HIGH when the line is not driven. During RESET the pin is sampled to determine whether the 80C188 should provide ALE, $\overline{\text{WR}}$ and $\overline{\text{RD}}$, or if the Queue-Status should be provided. $\overline{\text{RD}}$ should be connected to GND to provide Queue-Status data.</td></tr>
<tr><td>ARDY</td><td>55</td><td>I</td><td>Asynchronous Ready informs the 80C188 that the addressed memory space or I/O device will complete a data transfer. The ARDY input pin will accept an asynchronous input, and is active HIGH. Only the rising edge is internally synchronized by the 80C188. This means that the falling edge of ARDY must be synchronized to the 80C188 clock. If connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active to terminate a bus cycle. If unused, this line should be tied LOW to yield control to the SRDY pin.</td></tr>
<tr><td>SRDY</td><td>49</td><td>I</td><td>Synchronous Ready must be synchronized externally to the 80C188. The use of SRDY provides a relaxed system-timing specification on the Ready input. This is accomplished by eliminating the one-half clock cycle which is required for internally resolving the signal level when using the ARDY input. This line is active HIGH. If this line is connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active before a bus cycle is terminated. If unused, this line should be tied LOW to yield control to the ARDY pin.</td></tr>
</table>

Table 1. 80C188 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$\overline{\text{LOCK}}$</td><td>48</td><td>O</td><td>$\overline{\text{LOCK}}$ output indicates that other system bus masters are not to gain control of the system bus while $\overline{\text{LOCK}}$ is active LOW. The $\overline{\text{LOCK}}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of the instruction following the LOCK prefix. No prefetches will occur while $\overline{\text{LOCK}}$ is asserted. $\overline{\text{LOCK}}$ is active LOW, is driven HIGH for one clock during RESET, and then floated.</td></tr>
<tr><td>$\overline{\text{S0}}$, $\overline{\text{S1}}$, $\overline{\text{S2}}$</td><td>52–54</td><td>O</td><td>Bus cycle status $\overline{\text{S0}}$–$\overline{\text{S2}}$ are encoded to provide bus-transaction information:
<table>
<tr><th colspan="4">80C188 Bus Cycle Status Information</th></tr>
<tr><th>$\overline{\text{S2}}$</th><th>$\overline{\text{S1}}$</th><th>$\overline{\text{S0}}$</th><th>Bus Cycle Initiated</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Interrupt Acknowledge</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Read I/O</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Write I/O</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Halt</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Instruction Fetch</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Read Data from Memory</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Write Data to Memory</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Passive (no bus cycle)</td></tr>
</table>
The status pins float during HOLD/HLDA.
$\overline{\text{S2}}$ may be used as a logical M/$\overline{\text{IO}}$ indicator, and $\overline{\text{S1}}$ as a DT/$\overline{\text{R}}$ indicator.
The status lines are driven HIGH for one clock during Reset, and then floated until a bus cycle begins.</td></tr>
<tr><td>HOLD (input)
HLDA (output)</td><td>50
51</td><td>I
O</td><td>HOLD indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80C188 clock. The 80C188 will issue a HLDA (HIGH) in response to a HOLD request at the end of T_4 or T_i. Simultaneous with the issuance of HLDA, the 80C188 will float the local bus and control lines. After HOLD is detected as being LOW, the 80C188 will lower HLDA. When the 80C188 needs to run another bus cycle, it will again drive the local bus and control lines.
In Enhanced Mode, HLDA will go low when a DRAM refresh cycle is pending in the 80C188 and an external bus master has control of the bus. It will be up to the external master to relinquish the bus by lowering HOLD so that the 80C188 may execute the refresh cycle. Lowering HOLD for four clocks and returning HIGH will insure only one refresh cycle to the external master. HLDA will immediately go active after the refresh cycle has taken place.</td></tr>
<tr><td>$\overline{\text{UCS}}$</td><td>34</td><td>O</td><td>Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{\text{UCS}}$ is software programmable.
$\overline{\text{UCS}}$ and $\overline{\text{LCS}}$ are sampled upon the rising edge of $\overline{\text{RES}}$. If both pins are held low, the 80C188 will enter ONCE™ Mode. In ONCE Mode all pins assume a high impedance state and remain so until a subsequent RESET. $\overline{\text{UCS}}$ has a weak internal pullup for normal operation.</td></tr>
</table>

Table 1. 80C188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{\text{LCS}}$	33	O	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K−256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{\text{LCS}}$ is software programmable. $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$ are sampled upon the rising edge of $\overline{\text{RES}}$. If both pins are held low, the 80C186 will enter ONCE Mode. In ONCE Mode all pins assume a high impedance state and remain so until a subsequent RESET. $\overline{\text{LCS}}$ has a weak internal pullup for normal operation.
$\overline{\text{MCS0}}$−3	38, 37, 36, 35	O	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined mid-range portion of memory (8K−512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{\text{MCS0}}$−3 are software programmable.
$\overline{\text{PCS0}}$	25	O	Peripheral Chip Select signals 0−4 are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{\text{PCS0}}$−4 are software programmable.
$\overline{\text{PCS1}}$–4	27, 28, 29, 30	O	
$\overline{\text{PCS5}}$/A1	31	O	Peripheral Chip Select 5 or Latched A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{\text{PCS5}}$ is software programmable. When programmed to provide latched. A1, rather than $\overline{\text{PCS5}}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{\text{PCS6}}$/A2	32	O	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{\text{PCS6}}$ is software programmable. When programmed to provide latched A2, rather than $\overline{\text{PCS6}}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{\text{R}}$	40	O	Data Transmit/Receive controls the direction of data flow through the external 8286/8287 data bus transceiver. When LOW, data is transferred to the 80C188. When HIGH the 80C188 places write data on the data bus.
$\overline{\text{DEN}}$	39	O	Data Enable is provided as an 8286/8287 data bus transceiver output enable. $\overline{\text{DEN}}$ is active LOW during each memory and I/O access. $\overline{\text{DEN}}$ is HIGH whenever DT/$\overline{\text{R}}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80C188. This architecture is common to the 8086, 8088, 80186 and 80286 microprocessor families as well. The 80C188 is a very high integration 16-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip. The 80C188 is object code compatible with the 8086/8088 microprocessors and adds 10 new instruction types to the existing 8086/8088 instruction set.

The 80C188 has two major modes of operation, Compatible and Enhanced. In Compatible Mode the 80C188 is completely compatible with NMOS 80188, with the exception of 8087 support. All pin functions, timings, and drive capabilities are identical. The Enhanced mode adds two new features to the system design. These are Power-Save control and Dynamic RAM refresh.

80C188 BASE ARCHITECTURE

The 8086, 8088, 80186, and 80286 families all contain the same basic set of registers, instructions, and addressing modes. The 80C188 processor is upward compatible with the 8086, 8088, and 80286 CPUs.

Register Set

The 80C188 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

General Registers

Eight 16-bit general purpose registers may be used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

Segment Registers

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

Status and Control Registers

Two 16-bit special purpose registers record or alter certain aspects of the 80C188 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

Status Word Description

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80C186 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

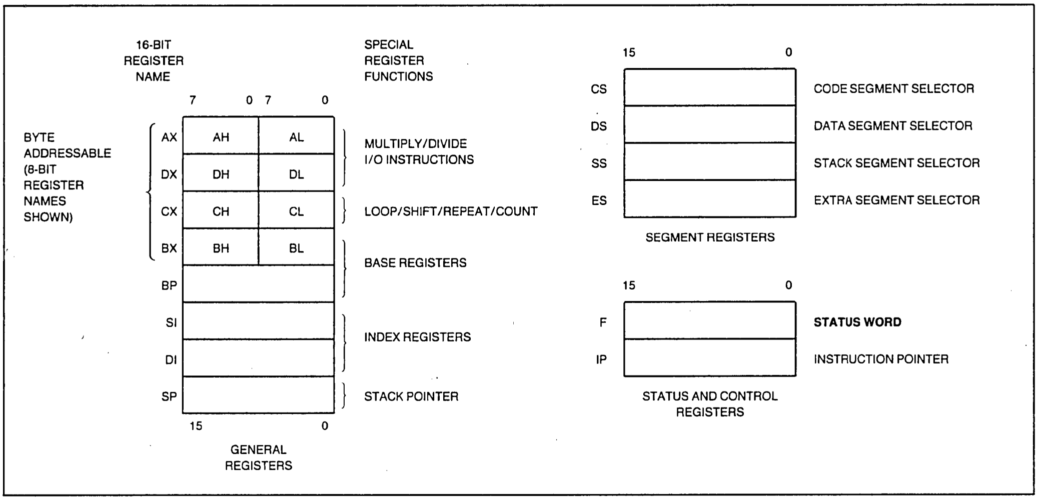


Figure 3a. 80C188 Register Set

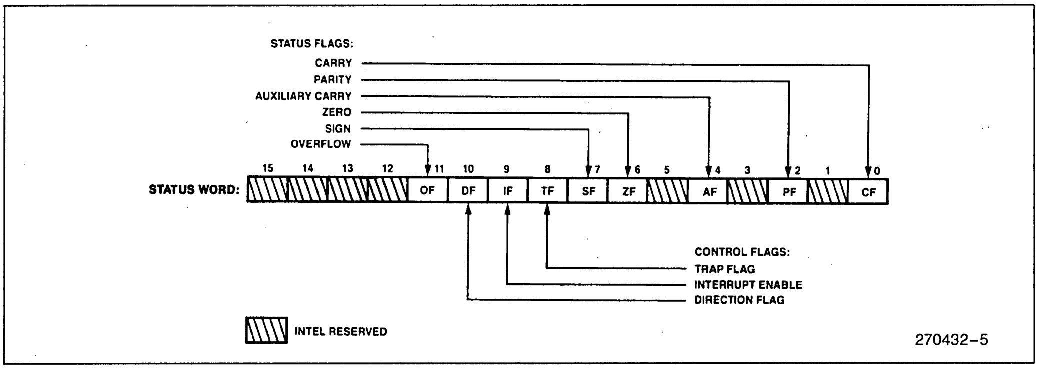


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80C188 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment register (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. 80C188 Instruction Set

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign

JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4. 80C188 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

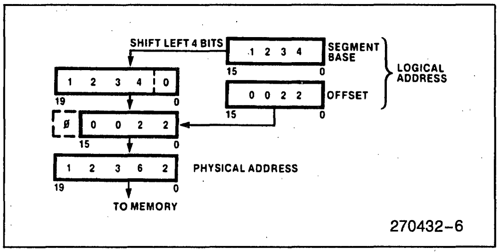


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

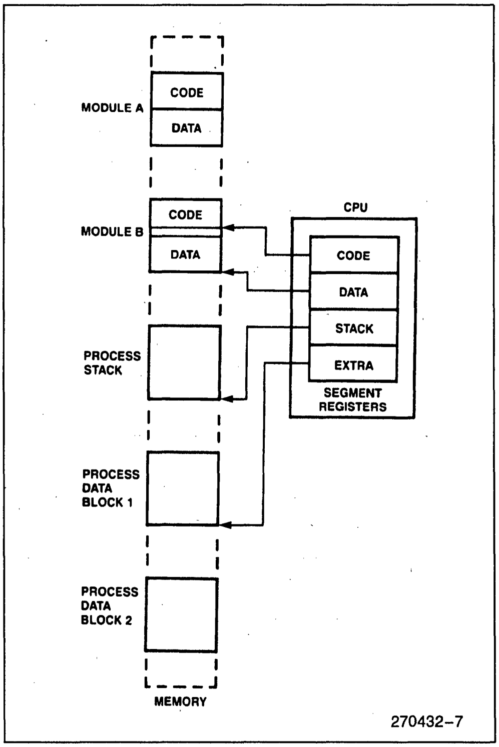


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80C188 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an Index register.
- *Based indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80C188 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.

In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80C188.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by attempted execution of an ESC instruction, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding

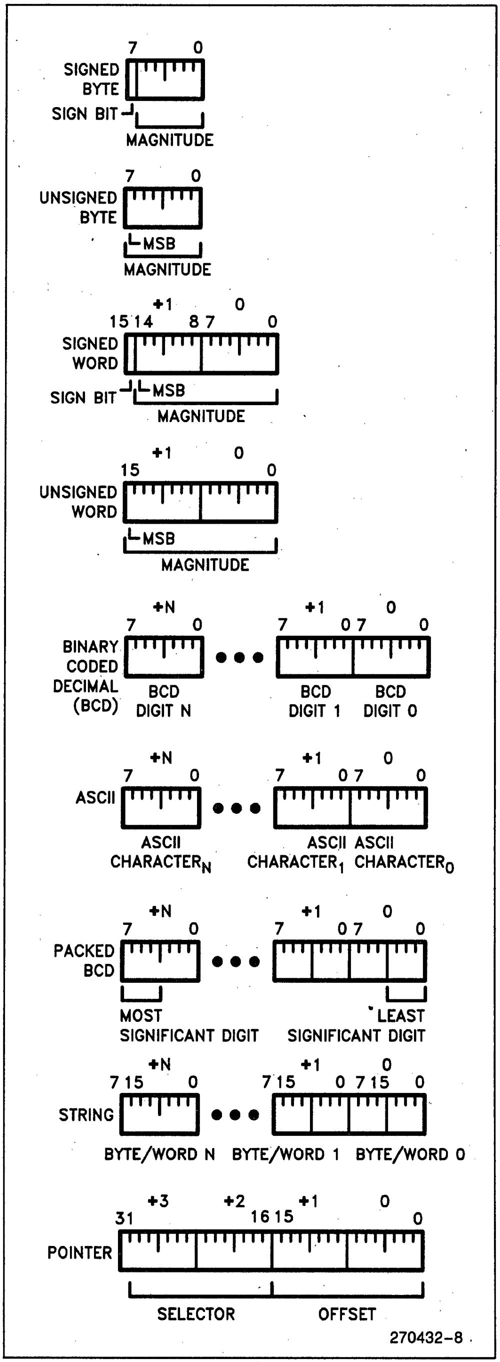


Figure 7. 80C188 Supported Data Types

the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80C188 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80C186 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80C188 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INT0, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80C188 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80C188 Interrupt Vectors

Interrupt Name	Vector Type	Default Priority	Related Instructions
Divide Error Exception	0	*1	DIV, IDIV
Single Step Interrupt	1	12**	All
NMI	2	1	All
Breakpoint Interrupt	3	*1	INT
INT0 Detected Overflow Exception	4	*1	INTO
Array Bounds Exception	5	*1	BOUND
Unused-Opcode Exception	6	*1	Undefined Opcodes
ESC Opcode Exception	7	*1***	ESC Opcodes
Timer 0 Interrupt	8	2A****	
Timer 1 Interrupt	18	2B****	
Timer 2 Interrupt	19	2C****	
Reserved	9	3	
DMA 0 Interrupt	10	4	
DMA 1 Interrupt	11	5	
INT0 Interrupt	12	6	
INT1 Interrupt	13	7	
INT2 Interrupt	14	8	
INT3 Interrupt	15	9	

NOTES:
*1. These are generated as the result of an instruction execution.
**2. This is handled as in the 8088.
****3. All three timers constitute one source of request to the interrupt controller. The Timer interrupts all have the same default priority level with respect to all other interrupt sources. However, they have a defined priority ordering amongst themselves. (Priority 2A is higher priority than 2B.) Each Timer interrupt has a separate vector type number.
4. Default priorities for the interrupt sources are used only if the user does not program each source into a unique priority level.
***5. An escape opcode will cause a trap regardless of the 80C188 operating mode.

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INT0 DETECTED OVERFLOW EXCEPTION (TYPE4)

Generated during an INT0 instruction if the 0F bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H–DFH). The 80C188 does not check an escape opcode trap bit as does the 80C186. On the 80C188, ESC traps occcur in both compatible and enhanced operating modes. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80C188 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80C188 integrated DMA controller and the integrated timer unit. The vector types for these interrupts is shown in Table 4. Software enables these inputs by setting the interrupt flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80C188 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80C188 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{\text{RES}}$ input pin LOW. $\overline{\text{RES}}$ forces the 80C188 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{\text{RES}}$ is active. After $\overline{\text{RES}}$ becomes inactive and an internal processing interval elapses, the 80C188 begins execution with the instruction at physical location FFFF0(H). $\overline{\text{RES}}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80C188 Initial Register State after RESET

Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

THE 80C188 COMPARED TO THE 80C186

The 80C188 is an 8-bit processor designed based on the 80C188 internal structure. Most internal functions of the 80C188 are identical to the equivalent 80C186 functions. The 80C188 handles the external bus the same way the 80C186 does with the distinction of handling only 8 bits at a time. Sixteen-bit operands are fetched or written in two consecutive bus cycles. The processors will look the same to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions except numerics instructions have the same end result. Internally, there are four differences between the 80C188 and the 80C186. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 80C188, whereas the 80C186 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 80C188 BIU will fetch a new instruction to load into the queue each time there is a 1-byte hole (space available) in the queue. The 80C186 waits until a 2-byte space is available.
- The internal execution time of an instruction is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU may also be limited by the rate of instruction fetches when a series of simple operations occur. When the more sophisticated instructions of the 80C188 are being used, the queue has more time to fill and the execution proceeds more closely to the speed at which the execution unit will allow.
- The 80C188 does not have a numerics interface, since the 80C186 numerics interface inherently requires 16-bit communication with the numerics coprocessor.

The 80C188 and 80C186 are completely software compatible (except for numerics instructions) by virtue of their identical execution units. However, software that is system dependent may not be completely transferable.

The bus interface and associated control signals vary somewhat between the two processors. The pin assignments are nearly identical, with the following functional changes:

- A8–A15—These pins are only address outputs on the 80C188. These address lines are latched internally and remain valid throughout the bus cycle.
- $\overline{\text{BHE}}$ has no meaning on the 80C188. However, it was necessary to designate this pin the S7/$\overline{\text{RFSH}}$ pin in order to provide an indication of DRAM refresh bus cycles.

80C188 CLOCK GENERATOR

The 80C188 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The 80C188 oscillator circuit is designed to be used either with a parallel resonant fundamental or third-overtone mode crystal, depending upon the frequency range of the application as shown in Figure 8c. This is used as the time base for the 80C188. The crystal frequency chosen should be twice the required processor frequency. Use of an LC or RC circuit is not recommended.

The output of the oscillator is not directly available outside the 80C188. The two recommended crystal configurations are shown in Figure 8a. When used in third-overtone mode the tank circuit shown in Figure 8b is recommended for stable operation. The sum of the stray capacitances and loading capacitors should equal the values shown. It is advisable to limit stray capacitance between the X1 and X2 pins to less than 10 pF. While a fundamental-mode circuit will require approximately 1 ms for start-up, the third-overtone arrangement may require 1 ms to 3 ms to stabilize.

Alternately the oscillator pins may be driven from an external source in a configuration shown in Figure 8d or Figure 8e. The configuration shown in Figure 8f is not recommended.

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	40Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pF max
C_1 (Load Capacitance):	20 pF ± 2 pF
Drive Level:	1 mW max

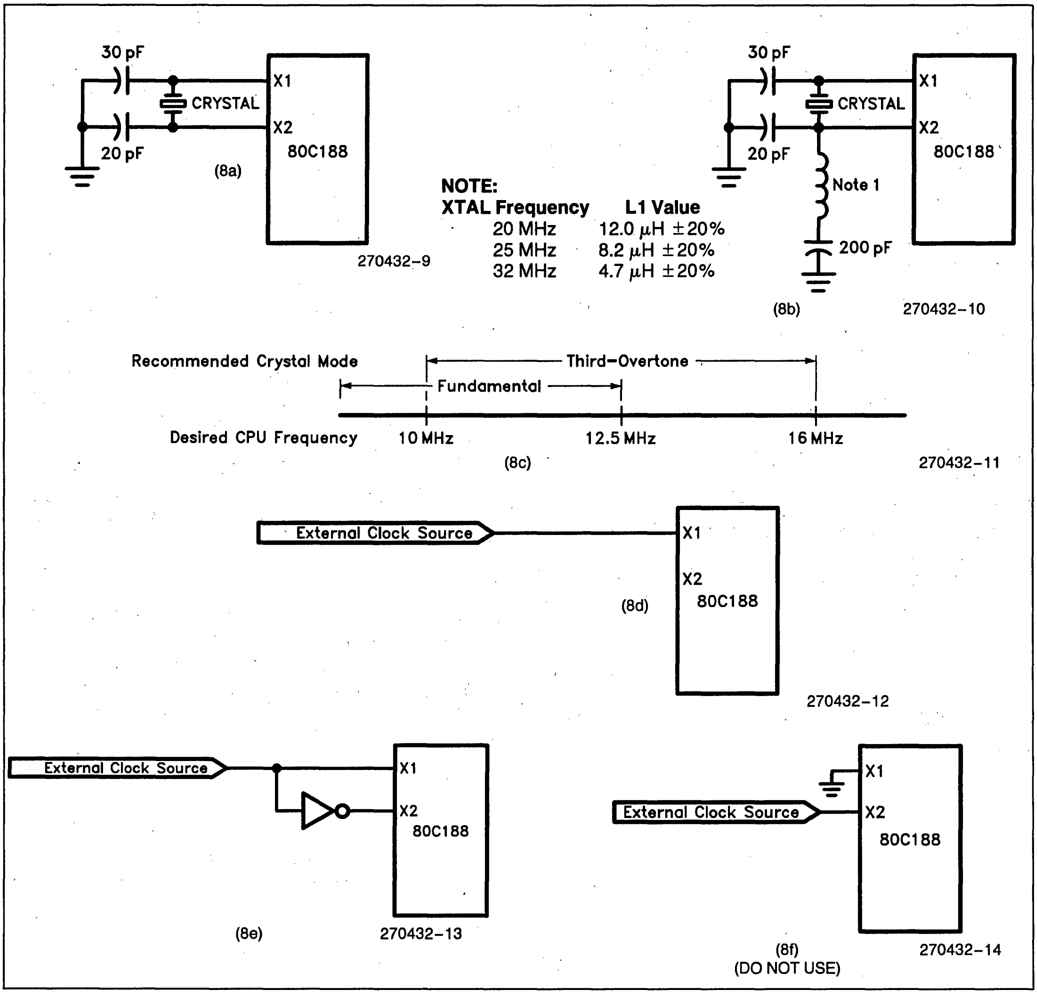
Clock Generator

The 80C188 clock generator provides the 50% duty cycle processor clock for the 80C188. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80C188. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80C188 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used. Full synchronization is performed only on the rising edge of ARDY, i.e., the falling edge of ARDY must be synchronized to the CLKOUT signal if it will occur during T_2, T_3, or T_W. High-to-LOW transitions of ARDY must be performed synchronously to the CPU clock.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.



NOTE:

XTAL Frequency	L1 Value
20 MHz	12.0 μH ±20%
25 MHz	8.2 μH ±20%
32 MHz	4.7 μH ±20%

Figure 8. 80C188 Oscillator Configurations (see text)

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80C188, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The 80C188 provides both a $\overline{\text{RES}}$ input pin and a synchronized RESET pin for use with other system components. The $\overline{\text{RES}}$ input pin on the 80C188 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{\text{RES}}$ input of at least six clocks. RESET may be delayed up to two and one-half clocks behind $\overline{\text{RES}}$.

Multiple 80C188 processors may be synchronized through the $\overline{\text{RES}}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{\text{RES}}$ must satisfy a 25 ns setup time before the falling edge of the 80C188 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 15 ns setup time before the rising edge of the CLKOUT signal of all the processors.

LOCAL BUS CONTROLLER

The 80C188 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides control lines that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80C188 provides ALE, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ bus control signals. The $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals are used to strobe data from memory to the 80C188 or to strobe data from the 80C188 to memory. The ALE line provides a strobe to address latches for the multiplexed address/data bus. The 80C188 local bus controller does not provide a memory/$\overline{\text{IO}}$ signal. If this is required, the user will have to use the $\overline{\text{S2}}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80C188 generates two control signals to be connected to external transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{\text{R}}$ and $\overline{\text{DEN}}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{\text{DEN}}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{\text{R}}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80C188 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80C188 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. This requires external circuitry to arbitrate which external device will gain control of the bus from the 80C188 when there is more than one alternate local bus master. When the 80C188 relinquishes control of the local bus, it floats $\overline{\text{DEN}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{S0}}$–$\overline{\text{S2}}$, $\overline{\text{LOCK}}$, AD0–AD7, A8–A19, S7/$\overline{\text{RFSH}}$, and DT/$\overline{\text{R}}$ to allow another master to drive these lines directly.

The 80C188 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is the highest-priority activity request which the processor may receive: higher than instruction fetching or internal DMA cycles. However, if a DMA cycle is in progress, the 80C188 will complete the transfer before relinquishing the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency time can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd ad-

dress to an odd address. This is a total of 16 clocks or more, if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

Local Bus Controller and Reset

Upon receipt of a RESET pulse from the $\overline{RES}$ input, the local bus controller will perform the following action:

- Drive $\overline{DEN}$, $\overline{RD}$, and $\overline{WR}$ HIGH for one clock cycle, then float.

NOTE:
$\overline{RD}$ is also provided with an internal pull-up device to prevent the processor from inadvertently entering Queue Status mode during reset.

- Drive $\overline{S0}$–$\overline{S2}$ to the passive state (all HIGH) and then float.
- Drive $\overline{LOCK}$ HIGH and then float.
- Float AD0–AD7, A8–A19, S7/$\overline{RFSH}$, DT/$\overline{R}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80C188 integrated peripherals are controlled via 16-bit registers contained within an internal 256-byte control block. This control block may be mapped into either memory or I/O space. Internal logic will recognize the address and respond to the bus cycle. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{15-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80C188 CPU at any time. The location of any register contained within the 256-byte control block is determined by the current base address of the control block.

The control block base address is programmed via a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. The control block is effectively an internal chip select range and must abide by all the rules concerning chip selects (the chip select circuitry is discussed later in this data sheet). Any access to the 256 bytes of the control block activates an internal chip select. Other chip selects may overlap the control block only if they are programmed to zero wait states and ignore external ready. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space, whereas if the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into slave mode. At RESET, the relocation register is set to 20FFH. This causes the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

The integrated 80C188 peripherals operate semi-autonomously from the CPU. Access to them for the most part is via software read/write of the control block. Most of these registers can be both read and written. A few dedicated lines, such as interrupts and DMA request provide real-time communication between the CPU and peripherals as in a more conventional system utilizing discrete peripheral blocks. The overall interaction and function of the peripheral blocks has not substantially changed.

CHIP-SELECT/READY GENERATION LOGIC

The 80C188 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also povide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The 80C188 provides 6 memory chip select outputs for 3 address areas; upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: FEH	X	SLAVE/$\overline{\text{MASTER}}$	X	M/$\overline{\text{IO}}$	Relocation Address Bits R19–R8											

M/$\overline{\text{IO}}$ = Register block located in Memory / I/O Space (1/0)
SLAVE/$\overline{\text{MASTER}}$ = Configures interrupt controller for Slave/Master Mode (1/0)

Figure 9. Relocation Register

	OFFSET
Relocation Register	FEH
DMA Descriptors Channel 1	DAH D0H
DMA Descriptors Channel 0	CAH C0H
Chip-Select Control Registers	A8H A0H
Time 2 Control Registers	66H 60H
Time 1 Control Registers	5EH 58H
Time 0 Control Registers	56H 50H
Interrupt Controller Registers	3EH 20H

Figure 10. Internal Register Map

of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes, whereas 80C188 memory is arranged in words. This means that if, for example, 16 64K x 1 memories are used, the memory block size will be 128K, not 64K.

Upper Memory $\overline{\text{CS}}$

The 80C188 provides a chip select, called $\overline{\text{UCS}}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80C188 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0=R1=R2=0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

Any internally generated 20-bit address whose upper 16 bits are greater than or equal to UMCS (with bits 0–5 "0") will cause UCS to be activated. UMCS bits R2–R0 are used to specify READY mode for the area of memory defined by this chip-select register, as explained below.

Lower Memory $\overline{\text{CS}}$

The 80C188 provides a chip select for low memory called $\overline{\text{LCS}}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is also defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12). This register is at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{\text{LCS}}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will cause $\overline{\text{LCS}}$ to be active. LMCS register bits R2–R0 are used to specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{\text{CS}}$

The 80C188 provides four $\overline{\text{MCS}}$ lines which are active within a user-locatable memory block. This block can be located within the 80C188 1M byte memory address space exclusive of the areas defined by $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{\text{MCS}}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. Thus, if the total block size is 32K, each chip select is active for 8K of memory with $\overline{\text{MCS0}}$ being active for the first range and $\overline{\text{MCS3}}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionally as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{\text{MCS}}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{\text{MCS}}$ lines will be active until both the MMCS and MPCS registers are accessed.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A0H	1	1	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 11. UMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A2H	0	0	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19								A11							

Figure 12. LMCS Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A8H	1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

Figure 13. MPCS Register

	15						9						3			0
OFFSET: A6H	U	U	U	U	U	U	U	1	1	1	1	1	1	R2	R1	R0
	A19						A13									

Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all mid-range chip selects. All devices in mid-range memory must use the same number of WAIT states.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{\text{LCS}}$ line was programmed, there would be an internal conflict between the $\overline{\text{LCS}}$ ready generation logic and the $\overline{\text{MCS}}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{\text{UCS}}$ ready generation logic. Since the $\overline{\text{LCS}}$ chip-select line does not become active until programmed, while the $\overline{\text{UCS}}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{\text{LCS}}$ range must not be programmed.

Peripheral Chip Selects

The 80C188 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. This base address may be located in either memory or I/O space.

Seven $\overline{\text{CS}}$ lines called $\overline{\text{PCS0}}$–6 are generated by the 80C188. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$ can also be programmed to provide latched address bits A1, A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0, A1 pins used for selecting internal registers of 8-bit peripheral chips. This scheme simplifies the hardware interface because the 8-bit registers of peripherals are simply treated as 16-bit registers located on even boundaries in I/O space or memory space where only the lower 8-bits of the register are significant: the upper 8-bits are "don't cares."

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). This register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

	15									6	5		3			0
OFFSET: A4H	U	U	U	U	U	U	U	U	U	U	1	1	1	R2	R1	R0
	A19									A10						

Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for $\overline{\text{PCS0}}$–$\overline{\text{PCS3}}$.

Table 10. PCS Address Ranges

PCS Line	Active between Locations
PCS0	PBA —PBA+127
PCS1	PBA+128—PBA+255
PCS2	PBA+256—PBA+383
PCS3	PBA+384—PBA+511
PCS4	PBA+512—PBA+639
PCS5	PBA+640—PBA+767
PCS6	PBA+768—PBA+895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 13). This register is located at offset A8H in the internal control block. Bit 7 is used to select the function of $\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space. 0 = Peripherals mapped into I/O space.
EX	0 = 5 $\overline{\text{PCS}}$ lines. A1, A2 provided. 1 = 7 $\overline{\text{PCS}}$ lines. A1, A2 are not provided.

MPCS bits 0–2 are used to specify READY mode for $\overline{\text{PCS4}}$–$\overline{\text{PCS6}}$ as outlined below.

READY Generation Logic

The 80C188 can generate a "READY" signal internally for each of the memory or peripheral $\overline{\text{CS}}$ lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the 80C188 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each $\overline{\text{CS}}$ line or group of lines generated by the 80C188. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). This means, for example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the $\overline{\text{PCS0}}$–3 READY mode, R2–R0 of MPCS set the $\overline{\text{PCS4}}$–6 READY mode.

Chip Select/Ready Logic and Reset

Upon reset, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the $\overline{\text{UCS}}$ line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to allow the maximum number of internal wait states in conjunction with external Ready consideration (i.e., UMCS resets to FFFBH).
- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the $\overline{\text{PCS}}$ lines will become active.

DMA CHANNELS

The 80C188 DMA controller provides two independent DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer. Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data.

DMA Operation

Each channel has six registers in the control block which define each channel's specific operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit destination pointer (2 words), a 16-bit Transfer Counter, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte or word transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 17). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

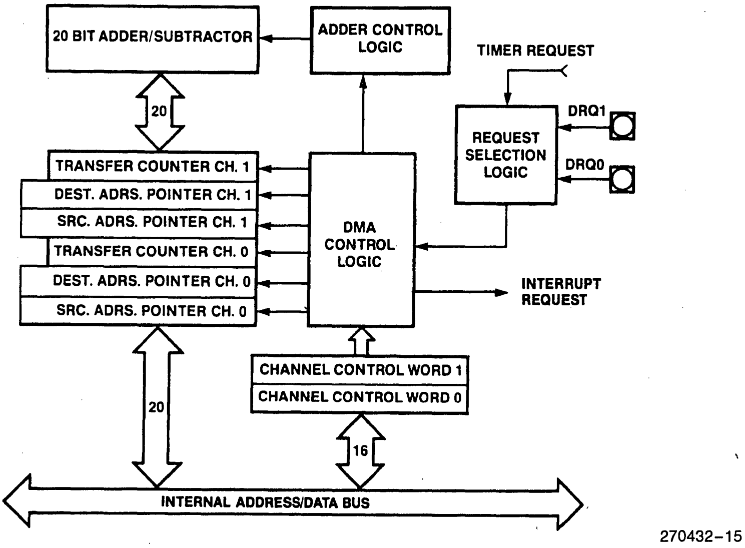


Figure 16. DMA Unit Block Diagram

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M/ $\overline{\text{IO}}$	DESTINATION DEC	 INC	M/ $\overline{\text{IO}}$	SOURCE DEC	 INC	TC	INT	SYN		P	T D R Q	X	CHG/ $\overline{\text{NOCHG}}$	ST/ $\overline{\text{STOP}}$	X

X = DON'T CARE.

Figure 17. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 80C188 DMA channel. This register specifies:

- the mode of synchronization;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

ST/$\overline{\text{STOP}}$: Start/stop (1/0) Channel.

CHG/$\overline{\text{NOCHG}}$: Change/Do not change (1/0) ST/$\overline{\text{STOP}}$ bit. If this bit is set when writing to the control word, the ST/$\overline{\text{STOP}}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{\text{STOP}}$ bit will not be altered. This bit is not stored; it will always be a 0 on read.

INT: Enable Interrupts to CPU on Transfer Count termination.

TC: If set, DMA will terminate when the contents of the Transfer Count register reach zero. The ST/$\overline{\text{STOP}}$ bit will also be reset at this point if TC is set. If this bit is cleared, the DMA unit will decrement the transfer count register for each DMA cycle, but the DMA transfer will not stop when the contents of the TC register reach zero.

SYN (2 bits)

00 No synchronization.

NOTE:

When unsynchronized transfers are specified, the TC bit will be ignored and the ST bit will be cleared upon the transfer count reaching zero, stopping the channel.

01 Source synchronization.

10 Destination synchronization.

11 Unused.

SOURCE:INC Increment source pointer by 1 after each transfer.

M/$\overline{\text{IO}}$ Source pointer is in M/IO space (1/0).

DEC Decrement source pointer by 1 after each transfer.

DEST: INC Increment destination pointer by 1 after each transfer.

M/$\overline{\text{IO}}$ Destination pointer is in M/IO space (1/0).

DEC Decrement destination pointer by 1 after each transfer.

P Channel priority—relative to other channel.

0 low priority.

1 high priority.

Channels will alternate cycles if both set at same priority level.

TDRQ 0: Disable DMA requests from timer 2.

1: Enable DMA requests from timer 2.

Bit 3 Bit 3 is not used.

If both INC and DEC are specified for the same pointer, the pointer will remain constant after each cycle.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18). These pointers may be individually incremented or decremented after each transfer. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers. Higher transfer rates can be obtained if all word transfers are performed to even addresses, since this will allow data to be accessed in a single memory access.

DMA Transfer Count Register

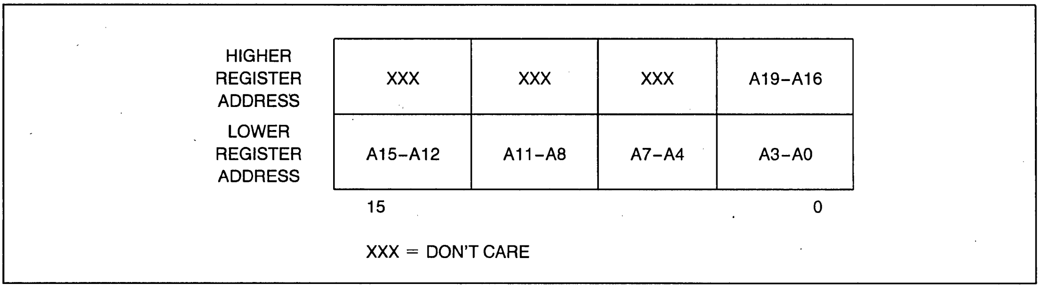
Each DMA channel maintains a 16-bit transfer count register (TC). This register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or if unsynchronized transfers are programmed, however, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). No prefetching occurs when destination synchronization is performed, however. Data will not be fetched from the source address until the destination device signals that it is ready to receive it. When destination synchronized transfers are requested, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another DMA cycle will begin after two processor clocks. This is done to allow the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. These lead to the maximum DMA transfer rates shown in Table 14.

Table 14. Maximum DMA Transfer Rates at 16 MHz

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	2.0 MBytes/sec	2.0 MBytes/sec
Source Synch	2.0 MBytes/sec	2.0 MBytes/sec
Destination Synch	1.3 MBytes/sec	1.6 MBytes/sec



	15			0
HIGHER REGISTER ADDRESS	XXX	XXX	XXX	A19–A16
LOWER REGISTER ADDRESS	A15–A12	A11–A8	A7–A4	A3–A0

XXX = DON'T CARE

Figure 18. DMA Memory Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed such that one channel is always given priority over the other, or they may be programmed such as to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses; however, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/STOP bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also have been generated. Therefore the source and destination transfer pointers, and the transfer count register (if used) must be programmed before this bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/STOP bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

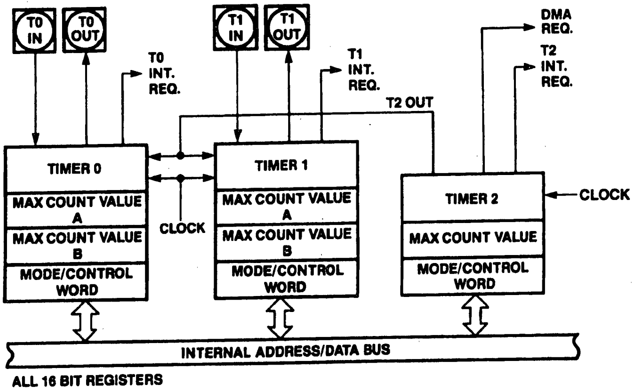
DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The Start/Stop bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80C188 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, this third timer can be used as a prescaler to the other two, or as a DMA request source.



270432-16

Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the internal peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values programmed by the user. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 1 clock after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate. Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

15	14	13	12	11		5	4	3	2	1	0
EN	$\overline{\text{INH}}$	INT	RIU	0	...	MC	RTG	P	EXT	ALT	CONT

Figure 20. Timer Mode/Control Register

ALT:

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT:

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If COUNT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

EXT:

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80C188 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

P:

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

RTG:

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80C188 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

EN:

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transistions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

$\overline{\text{INH}}$:

The inhibit bit allows for selective updating of the enable (EN) bit. If $\overline{\text{INH}}$ is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If $\overline{\text{INH}}$ is a zero during the write, the EN bit will be unaffected by the operation. This bit is not stored; it will always be a 0 on a read.

INT:

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller).

MC:

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set

regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts.

Programmer intervention is required to clear this bit.

RIU:

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The current contents of this register may be read or written by the processor at any time. If the register is written into while the timer is counting,the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. The condition which causes a timer to reset is equivalent between the current count value and the max count being used. This means that if the count is changed to be above the max count value, or if the max count value is changed to be below the current value, the timer will not reset to zero, but rather will count to its maximum value, "wrap around" to zero, then count until the max count is reached.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- All SEL (Select) bits are reset to zero. This selects MAX COUNT register A, resulting in the Timer Out pins going HIGH upon RESET.

INTERRUPT CONTROLLER

The 80C188 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80C188 interrupt controller has its own control register that set the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may themselves be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The 80C188 has a special slave mode in which the internal interrupt controller acts as a slave to an external master. The controller is programmed into this mode by setting bit 14 in the peripheral control block relocation register. (See Slave Mode section.)

MASTER MODE OPERATION

Interrupt Controller External Interface

For external interrupt sources, five dedicated pins are provided. One of these pins is dedicated to NMI, non-maskable interrupt. This is typically used for power-fail interrupts, etc. The other four pins may function either as four interrupt input lines with internally generated interrupt vectors, as an interrupt line and an interrupt acknowledge line (called the "cascade mode") along with two other input lines with internally generated interrupt vectors, or as two interrupt input lines and two dedicated interrupt acknowledge output lines. When the interrupt lines are configured in cascade mode, the 80C188 interrupt controller will not generate internal interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80C188 on the second cycle. The capability to interface to external 82C59A programmable interrupt controllers is thus provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in master mode are similar to the 82C59A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

Fully Nested Mode

When in the fully nested mode four pins are used as direct interrupt requests as in Figure 22. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled without being themselves interrupted by lower-priority interrupts. Since interrupts are enabled, higher-priority interrupts will be serviced.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is issued at the end of the service routine just before the issuance of the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

Cascade Mode

The 80C188 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 23. INT0 is an interrupt input interfaced to an 82C59A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 82C59As. Three levels of priority are created, requiring priority resolution in the 80C188 interrupt controller, the master 82C59As, and the slave 82C59As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

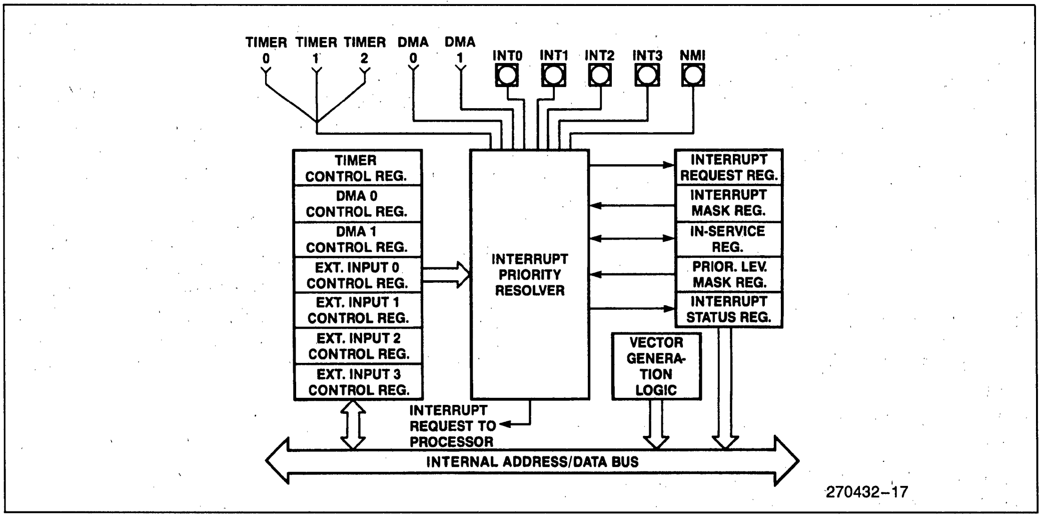


Figure 21. Interrupt Controller Block Diagram

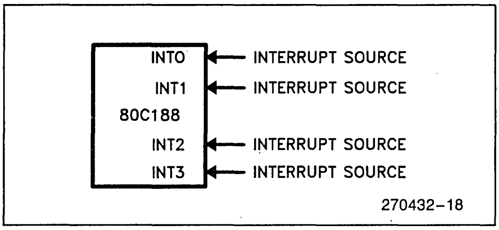


Figure 22. Fully Nested (Direct) Mode Interrupt Controller Connections

Special Fully Nested Mode

This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 82C59A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80C188 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80C188 controller until the 80C188 in-service bit is reset. In special fully nested mode, the 80C188 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80C188 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the external master's IS register is required to determine if there is more than one bit set. If so, the IS bit in the 80C188 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 32). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80C188 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Master Mode Features

Programmable Priority

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, it allows other requests to be serviced.

End-of-Interrupt Command

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

Trigger Mode

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the

80C188 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

Interrupt Vectoring

The 80C186 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 24. It contains 15 registers. All registers can both be read or written unless specified otherwise.

In-Service Register

This register can be read from or written into. The format is shown in Figure 25. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command issued by the CPU.

Interrupt Request Register

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 25. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits show exactly when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Writes to the interrupt request register will affect the D0 and D1 interrupt request bits. Setting either bit will cause the corresponding interrupt request while clearing either bit will remove the corresponding interrupt request. All other bits in the register are read-only.

Mask Register

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 25. A one in a bit position corre-

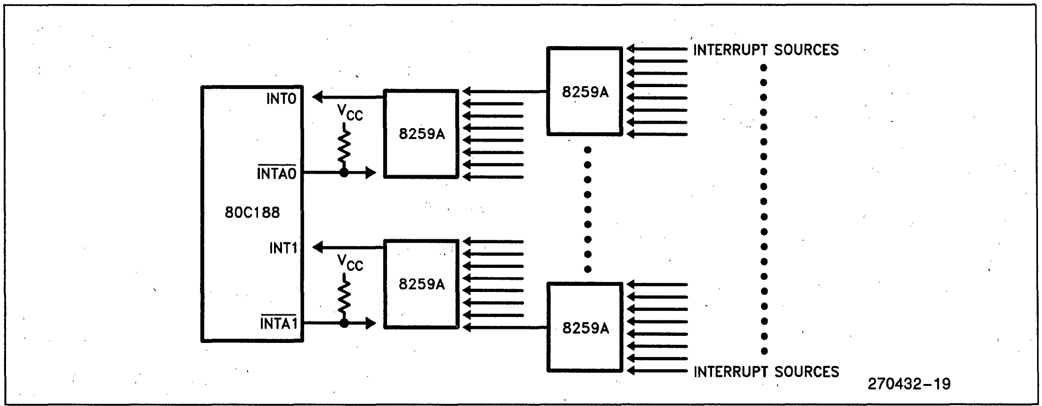


Figure 23. Cascade and Special Fully Nested Mode Interrupt Controller Connections

sponding to a particular source serves to mask the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER.	22H

Figure 24. Interrupt Controller Registers (Master Mode)

Priority Mask Register

This register is used to mask all interrupts below particular interrupt priority levels. The format of this register is shown in Figure 26. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so no interrupts are masked due to priority number.

Interrupt Status Register

This register contains general interrupt controller status information. The format of this register is shown in Figure 27. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. The purpose of this bit is to allow prompt service of all non-maskable interrupts. This bit may also be set by the programmer.

IRTx: These three bits represent the individual timer interrupt request bits. These bits are used to differentiate the timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt request. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

15	14				10	9	8	7	6	5	4	3	2	1	0
0	0	•	•	•	0	0	0	13	12	I1	I0	D1	D0	0	TMR

Figure 25. In-Service, Interrupt Request, and Mask Register Formats

15	14											3	2	1	0
0	0	•	•	•	•	•	•	•	•	•		0	PRM2	PRM1	PRM0

Figure 26. Priority Mask Register Format

15	14						7	6	5	4	3	2	1	0
DHLT	0	•	•	•	•	•	0	0	0	0	0	IRT2	IRT1	IRT0

Figure 27. Interrupt Status Register Format (Master Mode)

Timer, DMA 0, 1; Control Register

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 28. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 Control Registers

These registers are the control words for the four external input pins. Figure 29 shows the format of the INT0 and INT1 Control registers; Figure 30 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest Priority = 000, Lowest Priority = 111

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is preceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct

SFNM: Special fully nested mode bit, 1 = SFNM

EOI Register

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 30. It initiates an EOI command when written to by the 80C188 CPU.

The bits in the EOI register are encoded as follows:

S_X: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10.

NOTE:

To reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

NSPEC/: A bit that determines the type of EOI com-
SPEC mand. Nonspecific = 1, Specific = 0.

15	14		4	3	2	1	0
0	0	• • • • • • • •	0	MSK	PR2	PR1	PRO

Figure 28. Timer/DMA Control Registers Formats

15	14		7	6	5	4	3	2	1	0
0	0	• • • • •	0	SFNM	C	LTM	MSK	PR2	PR1	PRO

Figure 29. INT0/INT1 Control Register Formats

15	14		5	4	3	2	1	0
0	0	• • • • • • •	0	LTM	MSK	PR2	PR1	PRO

Figure 30. INT2/INT3 Control Register Formats

Poll and Poll Status Registers

These registers contain polling information. The format of these registers is shown in Figure 32. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_X: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

SLAVE MODE OPERATION

When slave mode is used, the internal 80C188 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80C188 resources will be monitored by the internal interrupt controller, while the external controller functions as the system master interrupt controller.

Upon reset, the 80C188 will be in master mode. To provide for slave mode operation bit 14 of the relocation register should be set.

Because of pin limitations caused by the need to interface to an external 82C59A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80C188 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

In slave mode each peripheral must be assigned a unique priority to ensure proper interrupt controller operation. Therefore, it is the programmer's responsibility to assign correct priorities and initialize interrupt control registers before enabling interrupts.

Slave Mode External Interface

The configuration of the 80C188 with respect to an external 82C59A master is shown in Figure 33. The INT0 (Pin 45) input is used as the 80C188 CPU interrupt input. INT3 (Pin 41) functions as an output to send the 80C188 slave-interrupt-request to one of the 8 master-PIC-inputs.

15	14	13								5	4	3	2	1	0
SPEC/ NSPEC	0	0	•	•	•	•	•	•	•	0	S4	S3	S2	S1	S0

Figure 31. EOI Register Format

15	14	13								5	4	3	2	1	0
INT REQ	0	0	•	•	•	•	•	•	•	0	S4	S3	S2	S1	S0

Figure 32. Poll and Poll Status Register Format

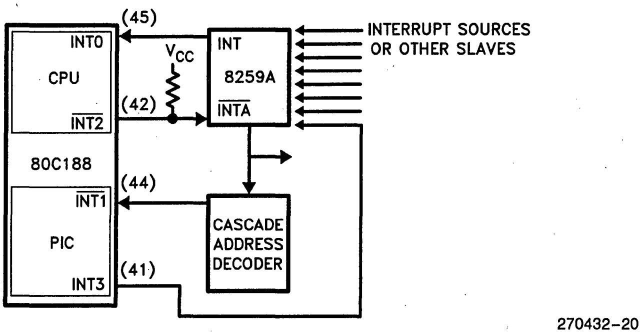


Figure 33. Slave Mode Interrupt Controller Connections

Correct master-slave interface requires decoding of the slave addresses (CAS0-2). Slave 82C59As do this internally. Because of pin limitations, the 80C188 slave address will have to be decoded externally. $\overline{\text{INT1}}$ (Pin 44) is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{\text{INT2}}$ (Pin 42) is used as an acknowledge output, suitable to drive the $\overline{\text{INTA}}$ input of an 82C59A.

Interrupt Nesting

Slave mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the Slave Mode

Vector generation in slave mode is exactly like that of an 82C59A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In slave mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the Slave Mode

All control and command registers are located inside the internal peripheral control block. Figure 34 shows the offsets of these registers.

End-of-Interrupt Register

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 35. It initiates an EOI command when written by the 80C188 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

In-Service Register

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 36. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

Interrupt Request Register

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 36. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request. As in master mode, D0 and D1 are read/write; all other bits are read only.

Mask Register

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 36. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

Control Registers

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 37. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT-REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 34. Interrupt Controller Registers (Slave Mode)

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	L2	L1	L0

Figure 35. Specific EOI Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 36. In-Service, Interrupt Request, and Mask Register Format

Interrupt Vector Register

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 38. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

Priority-Level Mask Register

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

Interrupt Status Register

This register is defined as in master mode except that DHLT is not implemented (see Figure 27).

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to master mode.

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	MSK	PR2	PR1	PR0

Figure 37. Control Word Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	t4	t3	t2	t1	t0	0	0	0

Figure 38. Interrupt Vector Register Format

15	14	13					8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	m2	m1	m0

Figure 39. Priority Level Mask Register

Enhanced Mode Operation

In Compatible Mode the 80C188 operates with all the features of the NMOS 80188, with the exception of 8087 support (i.e. no numeric coprocessing is possible). Queue-Status information is still available for design purposes other than 8087 support.

All the Enhanced Mode features are completely masked when in Compatible Mode. A write to any of the Enhanced Mode registers will have no effect, while a read will not return any valid data.

In Enhanced Mode, the 80C188 will operate with Power-Save and DRAM refresh, in addition to all the Compatible Mode features.

Entering Enhanced Mode

Enhanced mode can be entered by tying the RESET output signal from the 80C188 to the TEST/BUSY input.

Queue-Status Mode

The queue-status mode is entered by strapping the $\overline{RD}$ pin low. $\overline{RD}$ is sampled at RESET and if LOW, the 80C188 will reconfigure the ALE and $\overline{WR}$ pins to be QS0 and QS1 respectively. This mode is available on the 80C188 in both Compatible and Enhanced Modes and is identical to the NMOS 80188.

DRAM Refresh Control Unit Description

The Refresh Control Unit (RCU) automatically generates DRAM refresh bus cycles. The RCU operates only in Enhanced Mode. After a programmable period of time, the RCU generates a memory read request to the BIU. If the address generated during a refresh bus cycle is within the range of a properly programmed chip select, that chip select will be activated when the BIU executes the refresh bus cycle. The ready logic and wait states programmed for that region will also be in force. If no chip select is activated, then external ready is automatically required to terminate the refresh bus cycle.

If the HLDA pin is active when a DRAM refresh request is generated (indicating a bus hold condition), then the 80C188 will deactivate the HLDA pin in order to perform a refresh cycle. The circuit external to the 80C188 must remove the HOLD signal in order to execute the refresh cycle. The sequence of HLDA going inactive while HOLD is being held active can be used to signal a pending refresh request.

All registers controlling DRAM refresh may be read and written in Enhanced Mode. When the processor is operating in Compatible Mode, they are deselected and are therefore inaccessible. Some fields of these registers cannot be written and are always read as zeros.

DRAM Refresh Addresses

The address generated during a refresh cycle is determined by the contents of the MDRAM register (see Figure 40) and the contents of a 9-bit counter. Figure 41 illustrates the origin of each bit.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MDRAM: Offset E0H	M6	M5	M4	M3	M2	M1	M0	0	0	0	0	0	0	0	0	0

Bits 0–8: Reserved, read back as 0.

Bits 9–15: M0–M6, are address bits A13–A19 of the 20-bit memory refresh address. These bits should correspond to the chip select address to be activated for the DRAM partition. These bits are set to 0 on RESET.

Figure 40. Memory Partition Register

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
M6	M5	M4	M3	M2	M1	M0	0	0	0	CA8	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0	1

M6–M0: Bits defined by MDRAM Register

CA8–CA0: Bits defined by refresh address counter

Figure 41. Addresses Generated by RCU

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CDRAM: Offset E2H	0	0	0	0	0	0	0	C8	C7	C6	C5	C4	C3	C2	C1	C0

Bits 0–8: C0–C8, clock divisor register, holds the number of CLKOUT cycles between each refresh request.

Bits 9–15: Reserved, read back as 0.

Figure 42. Clock Pre-Scaler Register

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EDRAM: Offset E4H	E	0	0	0	0	0	0	T8	T7	T6	T5	T4	T3	T2	T1	T0

Bits 0–8: T0–T8, refresh clock counter outputs. Read only.

Bits 9–14: Reserved, read back as 0.

Bit 15: Enable RCU, set to 0 on RESET.

Figure 43. Enable RCU Register

Refresh Control Unit Programming and Operation

After programming the MDRAM and the CDRAM registers (Figures 40 and 42), the RCU is enabled by setting the "E" bit in the EDRAM register (Figure 43). The clock counter (T0–T8 of EDRAM) will be loaded from C0–C8 of CDRAM during T_3 of instruction cycle that sets the "E" bit. The clock counter is then decremented at each subsequent CLKOUT.

A refresh is requested when the value of the counter has reached 1 and the counter is reloaded from CDRAM. In order to avoid missing refresh requests, the value in the CDRAM register should always be at least 18 (12H). Clearing the "E" bit at anytime will clear the counter and stop refresh requests, but will not reset the refresh address counter.

POWER-SAVE CONTROL

Power Save Operation

The 80C188, when in Enhanced Mode, can enter a power saving state by internally dividing the clock-in frequency by a programmable factor. This divided frequency is also available at the CLKOUT pin. The PDCON register contains the two-bit fields for selecting the clock division factor and the enable bit.

All internal logic, including the Refresh Control Unit and the timers, will have their clocks slowed down by the division factor. To maintain a real time count or a fixed DRAM refresh rate, these peripherals must be re-programmed when entering and leaving the power-save mode.

The power-save mode is exited whenever an interrupt is processed by automatically resetting the enable bit. If the power-save mode is to be re-entered after serving the interrupt, the enable bit will need to be reset in software before returning from the interrupt routine.

The internal clocks of the 80C188 will begin to be divided during the T_3 state of the instruction cycle that sets the enable bit. Clearing the enable bit will restore full speed in the T_3 state of that instruction.

At no time should the internal clock frequency be allowed to fall below 0.5 MHz. This is the minimum operational frequency of the 80C188. For example, an 80C188 running with a 12 MHz crystal (6 MHz CLOCKOUT) should never have a clock divisor greater than eight.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDCON: Offset F0H	E	0	0	0	0	0	0	0	0	0	0	0	0	0	F1	F0

Bits 0–1: Clock Divisor Select

F1	F0	Division Factor
0	0	divide by 1
0	1	divide by 4
1	0	divide by 8
1	1	divide by 16

Bits 2–14: Reserved, read back as zero.

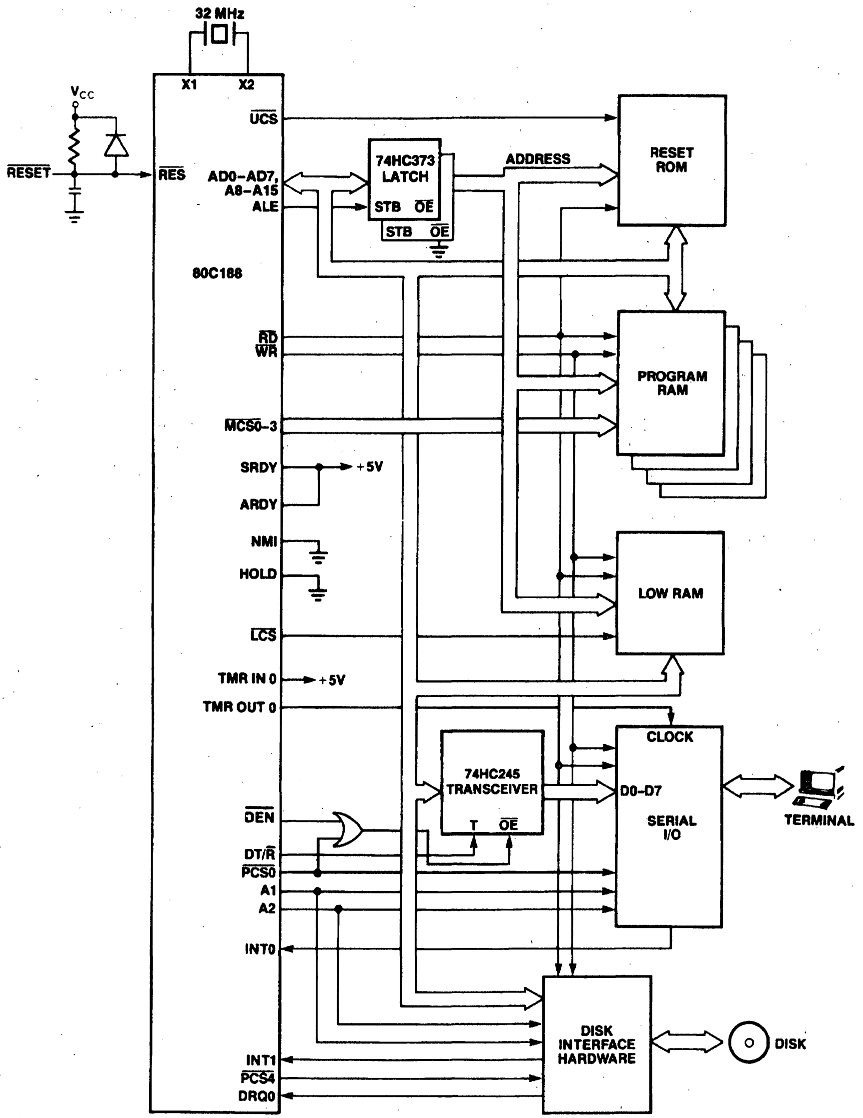
Bit 15: Enable Power Save Mode. Set to zero on RESET.

Figure 44. Power-Save Control Register

ONCE™ Test Mode

To facilitate testing and inspection of devices when fixed into a target system, the 80C188 has a test mode available which allows all pins to be placed in a high-impedance state. "ONCE" stands for "ON Circuit Emulation". When placed in this mode, the 80C188 will put all pins in the high-impedance state until RESET.

The ONCE mode is selected by tying the $\overline{UCS}$ and the $\overline{LCS}$ LOW during RESET. These pins are sampled on the low-to-high transition of the $\overline{RES}$ pin. The $\overline{UCS}$ and the $\overline{LCS}$ pins have weak internal pull-up resistors similar to the $\overline{RD}$ and $\overline{TEST}$/BUSY pins to guarantee proper normal operation.



270432–21

Figure 45. Typical 80C188 Computer

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ....0°C to +70°C
Storage Temperature ....−65°C to +150°C
Voltage on Any Pin with
Respect to Ground ....−1.0V to +7.0V
Package Power Dissipation 3W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

D.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ± 5% at 16 MHz

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.2 V_{CC} − 0.3	V	
V_{IH}	Input High Voltage (All except X1 and $\overline{RES}$)	0.2 V_{CC} + 0.9	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.5 mA (S0, 1, 2) I_{OL} = 2.0 mA (others)
V_{OH}	Output High Voltage	2.4	V_{CC}	V	I_{OH} = −2.4 mA @ 2.4V
		0.8 V_{CC}	V_{CC}	V	I_{OH} = −200 μA @ 0.8 V_{CC}
I_{CC}	Power Supply Current		150	mA	@ 12.5 MHz, 0°C V_{CC} = 5.5V
I_{PS}	Power Save Current	10 mA per MHz + 20		mA	Typical @25°C, V_{CC} = 5.0V
I_{LI}	Input Leakage Current		±10	μA	0.45V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}(1)
V_{CLO}	Clock Output Low		0.5	V	I_{CLO} = 4.0 mA
V_{CHO}	Clock Output High	0.8 V_{CC}		V	I_{CHO} = −500 μA
V_{CLI}	Clock Input Low Voltage (X1)	−0.5	0.6	V	
V_{CHI}	Clock Input High Voltage (X1)	3.9	V_{CC} + 0.5	V	
C_{IN}	Input Capacitance		10	pF	@ 1 MHz(2)
C_{IO}	I/O Capacitance		20	pF	@ 1 MHz(2)

NOTES:
1. Pins being floated during HOLD or by invoking the ONCE Mode.
2. Characterization conditions are a) Frequency = 1 MHz; b) Unmeasured pins at GND; c) V_{IN} at +5.0V or 0.45V. This parameter is not tested.

PIN TIMINGS

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at 16 MHz

All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
Input V_{IL} = 0.45V and V_{IH} = 2.4V for A.C. tests.

Symbol	Parameter	80C188-10		80C188-12		80C188-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C186 TIMING REQUIREMENTS									
T_{DVCL}	Data In Setup (A/D)	15		15		10		ns	
T_{CLDX}	Data In Hold (A/D)	5		5		5		ns	
T_{ARYCH}	ARDY Resolution Transition Setup Time(1)	15		15		15		ns	
T_{ARYLCL}	Asynchronous Ready (ARDY) Setup Time	25		25		25		ns	
T_{CLARX}	ARDY Active Hold Time	15		15		15		ns	
T_{ARYCHL}	ARDY Inactive Hold Time	15		15		15		ns	
T_{SRYCL}	Synchronous Ready (SRDY) Transition Setup Time(1)	15		15		15		ns	
T_{CLSRY}	SRDY Transition Hold Time	15		15		15		ns	
T_{HVCL}	HOLD Setup(1)	15		15		15		ns	
T_{INVCH}	INTR, NMI, TEST, TMR IN Setup Time(1)	15		15		15		ns	
T_{INVCL}	DRQ0, DRQ1, Setup Time(1)	15		15		15		ns	
80C188 MASTER INTERFACE TIMING RESPONSES									
T_{CLAV}	Address Valid Delay	5	50	5	36	5	33	ns	C_L = 50 pF −200 pF all outputs (except T_{CLTMV}) @ 10 MHz
T_{CLAX}	Address Hold	0		0		0		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	30	T_{CLAX}	25	T_{CLAX}	20	ns	
T_{CHCZ}	Command Lines Float Delay		40		33		28	ns	
T_{CHCV}	Command Lines Valid Delay (after Float)		45		37		32	ns	
T_{LHLL}	ALE Width (min)	$T_{CLCL} - 30$		$T_{CLCL} - 30$		$T_{CLCL} - 30$		ns	C_L = 50 pF −100 pF all outputs @ 12.5 & 16 MHz
T_{CHLH}	ALE Active Delay		30		25		20	ns	
T_{CHLL}	ALE Inactive Delay		30		25		20	ns	
T_{LLAX}	Address Hold to ALE Inactive (min)	$T_{CHCL} - 20$		$T_{CHCL} - 15$		$T_{CHCL} - 15$		ns	
T_{CLDV}	Data Valid Delay	5	40	5	36	5	33	ns	
T_{CLDOX}	Data Hold Time	5		5		5		ns	
T_{WHDX}	Data Hold after $\overline{WR}$ (min)	$T_{CLCL} - 34$		$T_{CLCL} - 20$		$T_{CLCL} - 20$		ns	
T_{CVCTV}	Control Active Delay 1	5	56	5	47	5	31	ns	
T_{CHCTV}	Control Active Delay 2	5	44	5	37	5	31	ns	
T_{CVCTX}	Control Inactive Delay	5	44	5	37	5	31	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)	5	56	5	47	5	35	ns	

NOTE:
1. To guarantee recognition at next clock.

PIN TIMINGS (Continued)

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at 16 MHz

All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
Input V_{IL} = 0.45V and V_{IH} = 2.4V for A.C. tests.

Symbol	Parameter	80C188-10		80C188-12		80C188-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C188 MASTER INTERFACE TIMING RESPONSES (Continued)									
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		0		ns	C_L = 50–200 pF all outputs (except T_{CLTMV}) @ 10 MHz
T_{CLRL}	$\overline{RD}$ Active Delay	5	44	5	37	5	31	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	5	44	5	37	5	31	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active (min)	$T_{CLCL} - 40$		$T_{CLCL} - 20$		$T_{CLCL} - 20$		ns	C_L = 50–100 pF all outputs @ 12.5 & 16 MHz
T_{CLHAV}	HLDA Valid Delay	5	40	5	33	5	25	ns	
T_{RLRH}	$\overline{RD}$ Pulse Width (min)	$2T_{CLCL} - 46$		$2T_{CLCL} - 40$		$2T_{CLCL} - 30$		ns	
T_{WLWH}	$\overline{WR}$ Pulse Width (min)	$2T_{CLCL} - 34$		$2T_{CLCL} - 30$		$2T_{CLCL} - 25$		ns	
T_{AVLL}	Address Valid to ALE Low (min)	$T_{CLCH} - 19$		$T_{CLCH} - 15$		$T_{CLCH} - 15$		ns	Equal Loading
T_{CHSV}	Status Active Delay	5	45	5	35	5	31	ns	
T_{CLSH}	Status Inactive Delay	5	50	5	35	5	30	ns	
T_{CLTMV}	Timer Output Delay		48		40		30	ns	100 pF max @ 10 MHz
T_{CLRO}	Reset Delay		48		40		30	ns	C_L = 50–200 pF All outputs (except T_{CLTMV}) @ 10 MHz
T_{CHQSV}	Queue Status Delay		28		28		25	ns	
T_{CHDX}	Status Hold Time	5		5		5		ns	
T_{AVCH}	Address Valid to Clock High	0		0		0		ns	C_L = 50–100 pF All outputs @ 12.5 & 16 MHz
T_{CLLV}	$\overline{LOCK}$ Valid/Invalid Delay	5	45	5	40	5	35	ns	
T_{DXDL}	$\overline{DEN}$ Inactive to DT/$\overline{R}$ Low	0		0		0		ns	Equal Loading
80C188 CHIP-SELECT TIMING RESPONSES									
T_{CLCSV}	Chip-Select Active Delay		45		33		30	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	$T_{CLCH} - 10$		$T_{CLCH} - 10$		$T_{CLCH} - 10$		ns	Equal Loading
T_{CHCSX}	Chip-Select Inactive Delay	5	32	5	28	5	23	ns	

PIN TIMINGS (Continued)

ADVANCE INFORMATION—SEE INTEL FOR DESIGN-IN INFORMATION

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±10% except V_{CC} = 5V ±5% at 16 MHz

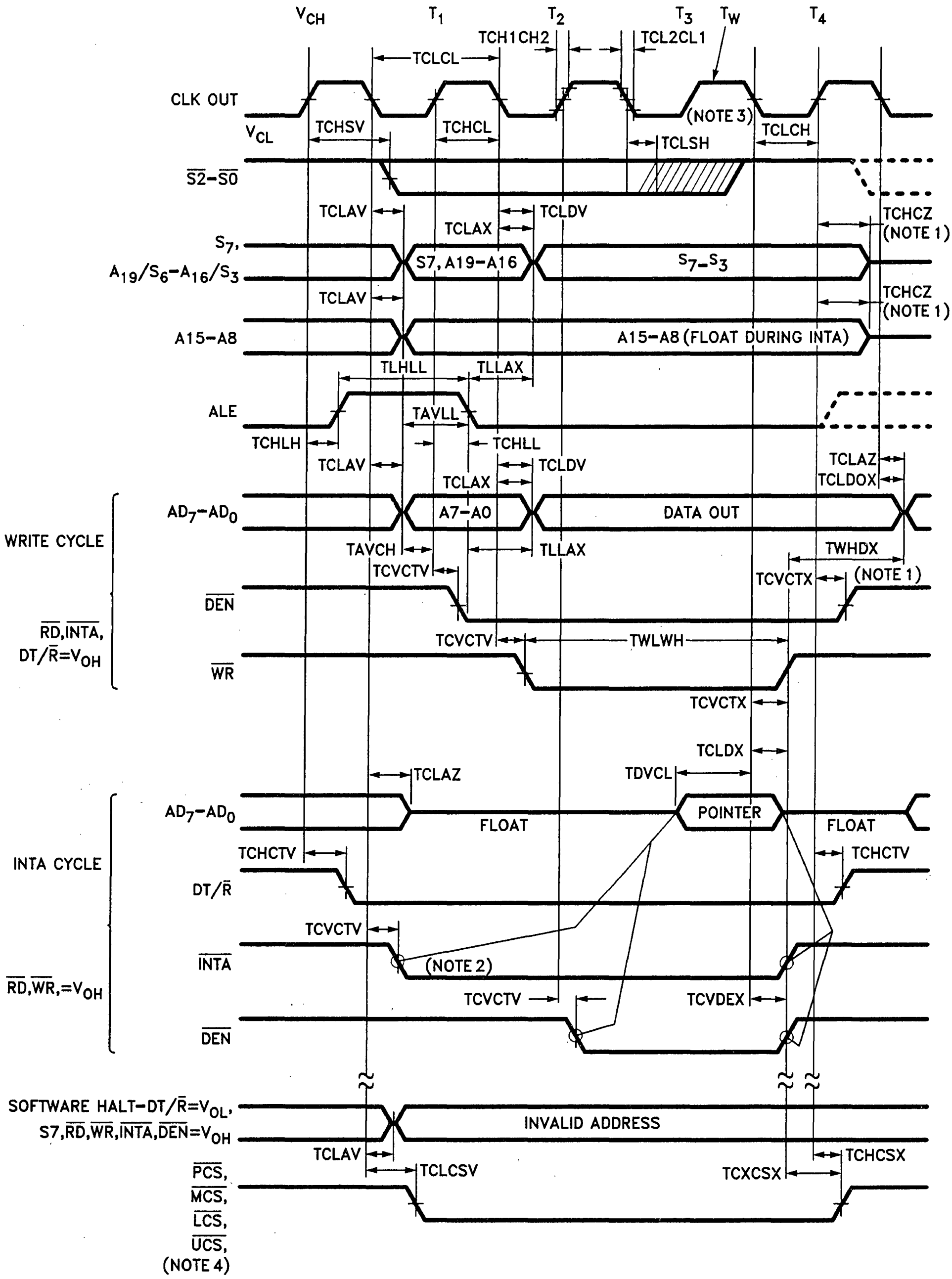
All timings are measured at 1.5V and 100 pF loading on CLKOUT unless otherwise noted.
All output test conditions are with C_L = 50–200 pF (10 MHz) and C_L = 50–100 pF (12.5–16 MHz).
Input V_{IL} = 0.45V and V_{IH} = 2.4V for A.C. tests.

Symbol	Parameter	80C188-10		80C188-12		80C188-16		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max		
80C188 CLKIN REQUIREMENTS Measurements taken with following conditions: External clock input to X1 and X2 not connected (float)									
T_{CKIN}	CLKIN Period	50	1000	40	1000	31.25	1000	ns	
T_{CKHL}	CLKIN Fall Time		5		5		5	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		5		5		5	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	20		15		13		ns	1.5V(2)
T_{CHCK}	CLKIN High Time	20		15		13		ns	1.5V(2)
80C188 CLKOUT TIMING 200 pF load maximum for 10 MHz or less, 100 pF load maximum above 10 MHz									
T_{CICO}	CLKIN to CLKOUT Skew		25		21		17	ns	
T_{CLCL}	CLKOUT Period	100	2000	80	2000	62.5	2000	ns	
T_{CLCH}	CLKOUT Low Time (min)	0.5 $T_{CLCL}-6$		0.5 $T_{CLCL}-5$		0.5 $T_{CLCL}-5$		ns	1.5V
T_{CHCL}	CLKOUT High Time (min)	0.5 $T_{CLCL}-6$		0.5 $T_{CLCL}-5$		0.5 $T_{CLCL}-5$		ns	1.5V
T_{CH1CH2}	CLKOUT Rise Time		10		10		8	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		10		10		8	ns	3.5 to 1.0V

NOTE:
2. T_{CLCK} and T_{CHCK} (CLKIN Low and High times) should not have a duration less than 40% of T_{CKIN}.

WAVEFORMS

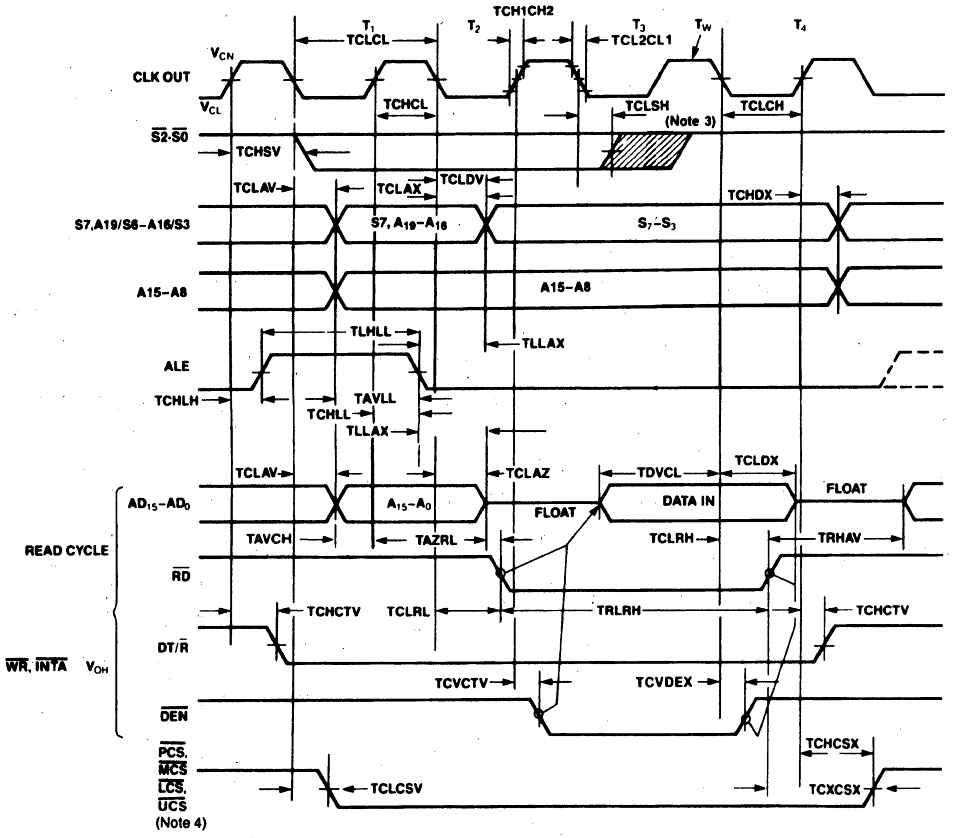
MAJOR CYCLE TIMING



270432-22

WAVEFORMS (Continued)

MAJOR CYCLE TIMING (Continued)

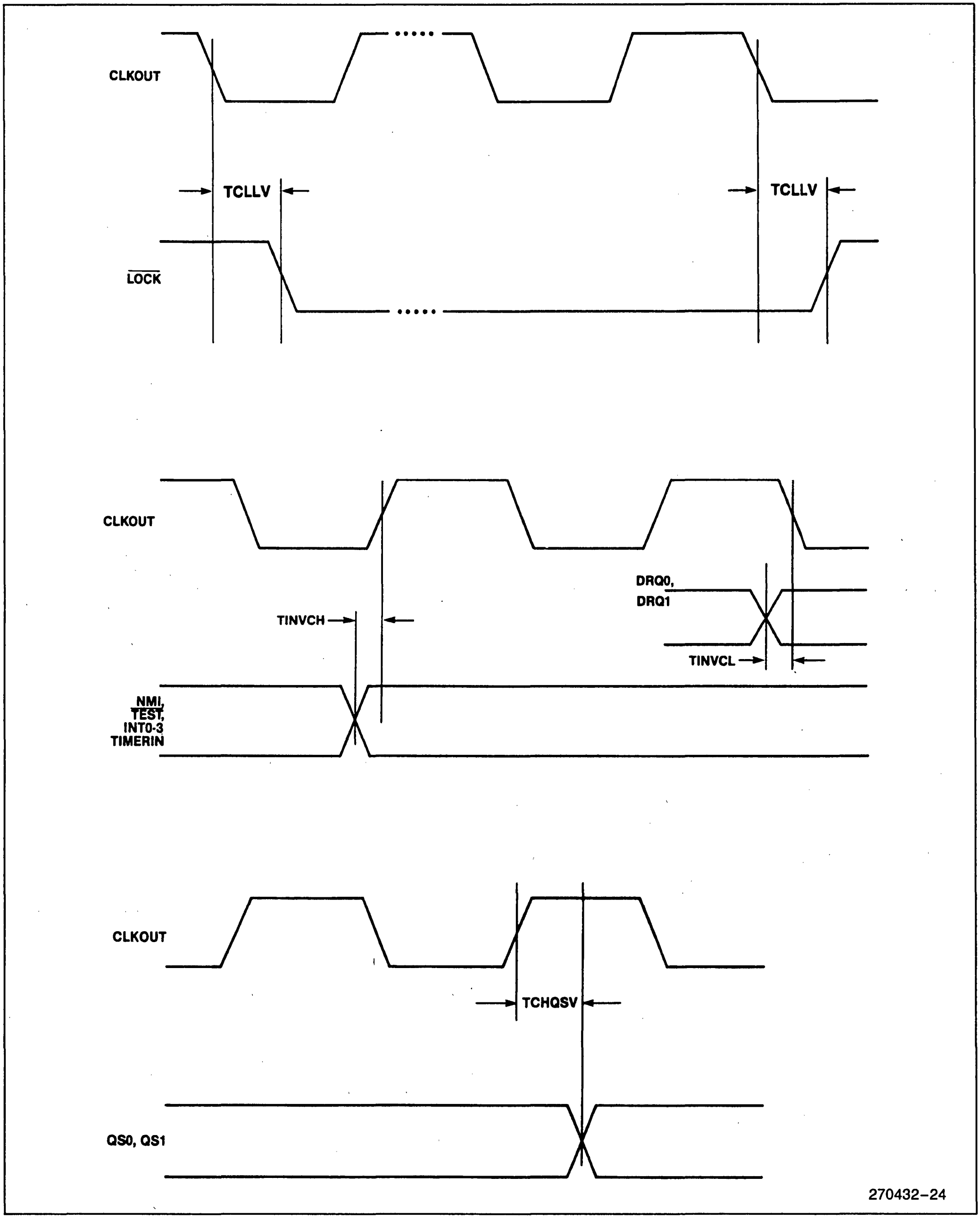


270432–23

NOTES:

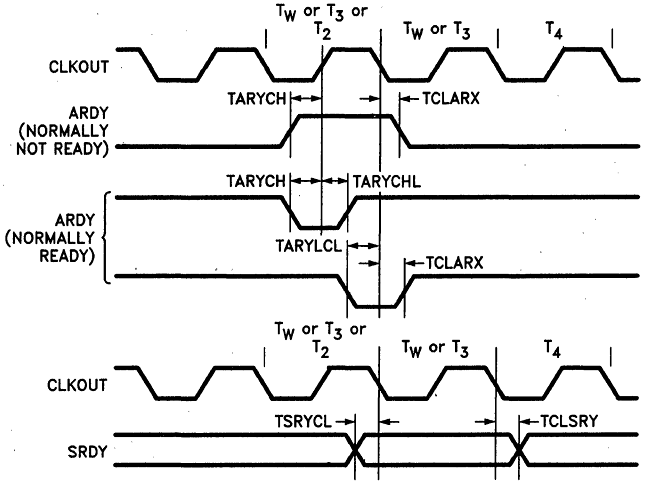
1. Following a Write cycle, the Local Bus is floated by the 80C188 only when the 80C188 enters a "Hold Acknowledge" state.
2. INTA occurs one clock later in slave mode.
3. Status inactive just prior to T4.
4. Latched A1 and A2 have the same timings as $\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$.

WAVEFORMS (Continued)



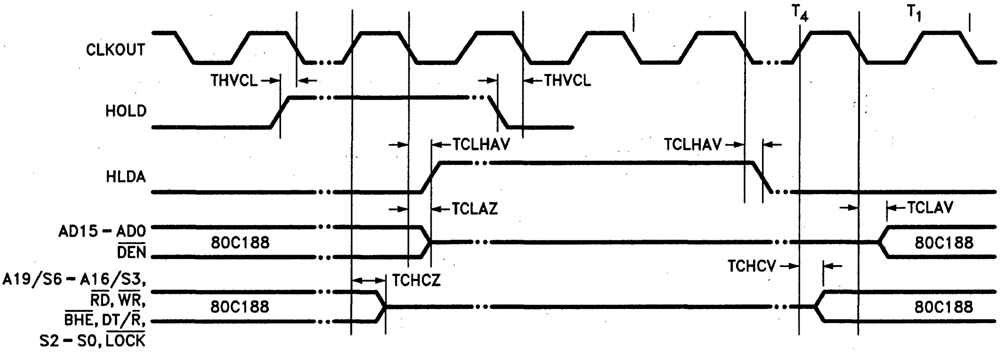
WAVEFORMS (Continued)

READY TIMING



270432–25

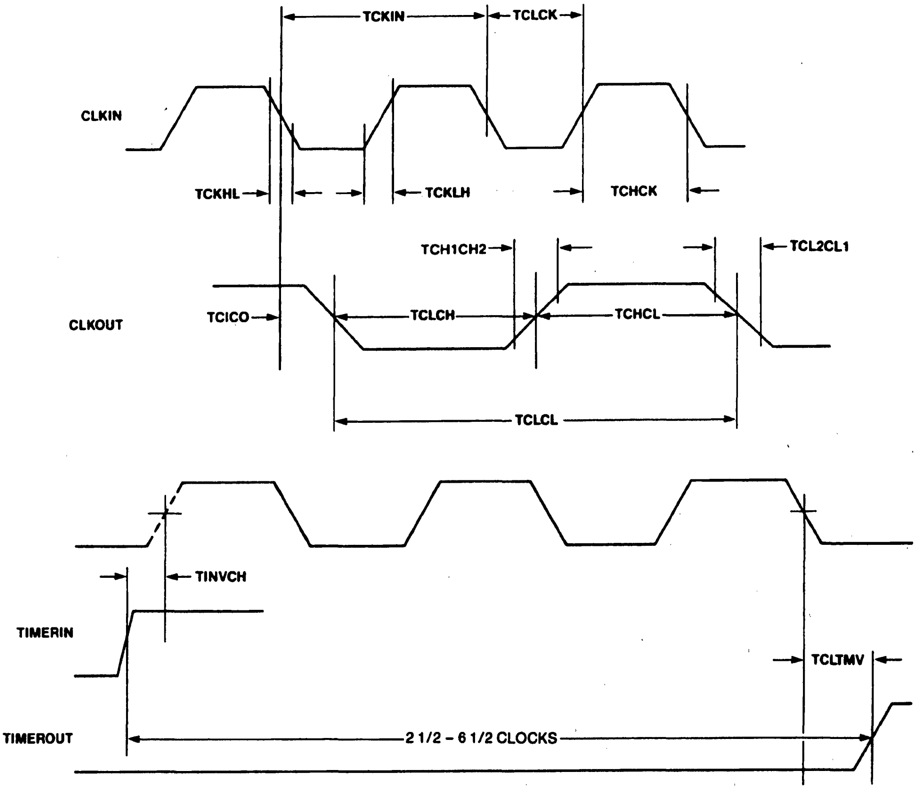
HOLD-HLDA TIMING



270432–26

WAVEFORMS (Continued)

TIMER ON 80C186



270432-27

80C188 EXECUTION TIMINGS

A determination of 80C188 program execution timing must consider both the bus cycles necessary to prefetch instructions as well as the number of execution unit cycles necessary to execute instructions. The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDs occur.

All instructions which involve memory accesses can require one or two additional clocks above the minimum timings shown due to the asynchronous handshake between the BIU and execution unit.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

The 80C188 8-bit BIU is noticeably limited in its performance relative to the execution unit. A sufficient number of prefetched bytes may not reside in the prefetch queue much of the time. Therefore, actual program execution will be substantially greater than that derived from adding the instruction timings shown.

INSTRUCTION SET SUMMARY

Function	Format				Clock Cycles	Comments
DATA TRANSFER **MOV = Move:**						
Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m			2/12*	
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m			2/9*	
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 000 r/m	data	data if w = 1	12/13	8/16-bit
Immediate to register	1 0 1 1 w reg	data	data if w = 1		3/4	8/16-bit
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high		8*	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high		9*	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m			2/13	
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m			2/15	
PUSH = Push:						
Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m			20	
Register	0 1 0 1 0 reg				14	
Segment register	0 0 0 reg 1 1 0				13	
Immediate	0 1 1 0 1 0 s 0	data	data if s = 0		14	
PUSHA = Push All	0 1 1 0 0 0 0 0				68	
POP = Pop:						
Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m			24	
Register	0 1 0 1 1 reg				14	
Segment register	0 0 0 reg 1 1 1	(reg≠01)			12	
POPA = Pop All	0 1 1 0 0 0 0 1				83	
XCHG = Exchange:						
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m			4/17*	
Register with accumulator	1 0 0 1 0 reg				3	
IN = Input from:						
Fixed port	1 1 1 0 0 1 0 w	port			10*	
Variable port	1 1 1 0 1 1 0 w				8*	
OUT = Output to:						
Fixed port	1 1 1 0 0 1 1 w	port			9*	
Variable port	1 1 1 0 1 1 1 w				7*	
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1				15	
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m			6	
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m	(mod≠11)		26	
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m	(mod≠11)		26	
LAHF = Load AH with flags	1 0 0 1 1 1 1 1				2	
SAHF = Store AH into flags	1 0 0 1 1 1 1 0				3	
PUSHF = Push flags	1 0 0 1 1 1 0 0				13	
POPF = Pop flags	1 0 0 1 1 1 0 1				12	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

***NOTE:**
Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
DATA TRANSFER (Continued) **SEGMENT = Segment Override:**						
CS	00101110				2	
SS	00110110				2	
DS	00111110				2	
ES	00100110				2	
ARITHMETIC **ADD = Add:**						
Reg/memory with register to either	000000dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 000 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0000010w	data	data if w=1		3/4	8/16-bit
ADC = Add with carry:						
Reg/memory with register to either	000100dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 010 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0001010w	data	data if w=1		3/4	8/16-bit
INC = Increment:						
Register/memory	1111111w	mod 000 r/m			3/15*	
Register	01000 reg				3	
SUB = Subtract:						
Reg/memory and register to either	001010dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 101 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0010110w	data	data if w=1		3/4	8/16-bit
SBB = Subtract with borrow:						
Reg/memory and register to either	000110dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 011 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0001110w	data	data if w=1		3/4	8/16-bit
DEC = Decrement						
Register/memory	1111111w	mod 001 r/m			3/15*	
Register	01001 reg				3	
CMP = Compare:						
Register/memory with register	0011101w	mod reg r/m			3/10*	
Register with register/memory	0011100w	mod reg r/m			3/10*	
Immediate with register/memory	100000sw	mod 111 r/m	data	data if s w=01	3/10*	
Immediate with accumulator	0011110w	data	data if w=1		3/4	8/16-bit
NEG = Change sign register/memory	1111011w	mod 011 r/m			3/10*	
AAA = ASCII adjust for add	00110111				8	
DAA = Decimal adjust for add	00100111				4	
AAS = ASCII adjust for subtract	00111111				7	
DAS = Decimal adjust for subtract	00101111				4	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

***NOTE:**
Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
ARITHMETIC (Continued)						
MUL = Multiply (unsigned):	1 1 1 1 0 1 1 w	mod 100 r/m				
Register-Byte					26–28	
Register-Word					35–37	
Memory-Byte					32–34	
Memory-Word					41–43*	
IMUL = Integer multiply (signed):	1 1 1 1 0 1 1 w	mod 1 0 1 r/m				
Register-Byte					25–28	
Register-Word					34–37	
Memory-Byte					31–34	
Memory-Word					40–43*	
IMUL = Integer immediate multiply (signed)	0 1 1 0 1 0 s 1	mod reg r/m	data	data if s=0	22–25/ 29–32	
DIV = Divide (unsigned):	1 1 1 1 0 1 1 w	mod 1 1 0 r/m				
Register-Byte					29	
Register-Word					38	
Memory-Byte					35	
Memory-Word					44*	
IDIV = Integer divide (signed):	1 1 1 1 0 1 1 w	mod 1 1 1 r/m				
Register-Byte					44–52	
Register-Word					53–61	
Memory-Byte					50–58	
Memory-Word					59–67*	
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0			19	
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0			15	
CBW = Convert byte to word	1 0 0 1 1 0 0 0				2	
CWD = Convert word to double word	1 0 0 1 1 0 0 1				4	
LOGIC						
Shift/Rotate Instructions:						
Register/Memory by 1	1 1 0 1 0 0 0 w	mod TTT r/m			2/15	
Register/Memory by CL	1 1 0 1 0 0 1 w	mod TTT r/m			5+n/17+n	
Register/Memory by Count	1 1 0 0 0 0 0 w	mod TTT r/m	count		5+n/17+n	
	TTT	**Instruction**				
	0 0 0	ROL				
	0 0 1	ROR				
	0 1 0	RCL				
	0 1 1	RCR				
	1 0 0	SHL/SAL				
	1 0 1	SHR				
	1 1 1	SAR				
AND = And:						
Reg/memory and register to either	0 0 1 0 0 0 d w	mod reg r/m			3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w=1	4/16*	
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w=1		3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

***NOTE:**
Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
LOGIC (Continued)						
TEST = And function to flags, no result:						
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m			3/10*	
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1	4/10*	
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w = 1		3/4	8/16-bit
OR = Or:						
Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m			3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1	4/16*	
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w = 1		3/4	8/16-bit
XOR = Exclusive or:						
Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m			3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1	4/16*	
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w = 1		3/4	8/16-bit
NOT = Invert register/memory	1 1 1 1 0 1 1 w	mod 0 1 0 r/m			3/10*	
STRING MANIPULATION						
MOVS = Move byte/word	1 0 1 0 0 1 0 w				14*	
CMPS = Compare byte/word	1 0 1 0 0 1 1 w				22*	
SCAS = Scan byte/word	1 0 1 0 1 1 1 w				15*	
LODS = Load byte/wd to ALAX	1 0 1 0 1 1 0 w				12*	
STOS = Stor byte/wd from ALA	1 0 1 0 1 0 1 w				10*	
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w				14	
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w				14	
Repeated by count in CX						
MOVS = Move string	1 1 1 1 0 0 1 0	1 0 1 0 0 1 0 w			8 + 8n*	
CMPS = Compare string	1 1 1 1 0 0 1 z	1 0 1 0 0 1 1 w			5 + 22n*	
SCAS = Scan string	1 1 1 1 0 0 1 z	1 0 1 0 1 1 1 w			5 + 15n*	
LODS = Load string	1 1 1 1 0 0 1 0	1 0 1 0 1 1 0 w			6 + 11n*	
STOS = Store string	1 1 1 1 0 0 1 0	1 0 1 0 1 0 1 w			6 + 9n*	
INS = Input string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 0 w			8 + 8n*	
OUTS = Output string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 1 w			8 + 8n*	
CONTROL TRANSFER						
CALL = **Call:**						
Direct within segment	1 1 1 0 1 0 0 0	disp-low	disp-high		19	
Register/memory indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m			17/27	
Direct intersegment	1 0 0 1 1 0 1 0	segment offset			31	
		segment selector				
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	(mod ≠ 11)		54	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

***NOTE:**
Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
CONTROL TRANSFER (Continued)			
JMP = **Unconditional jump:**			
Short/long	11101011 disp-low	14	
Direct within segment	11101001 disp-low disp-high	14	
Register/memory indirect within segment	11111111 mod 100 r/m	11/21	
Direct intersegment	11101010 segment offset segment selector	14	
Indirect intersegment	11111111 mod 101 r/m (mod ≠ 11)	34	
RET = **Return from CALL:**			
Within segment	11000011	20	
Within seg adding immed to SP	11000010 data-low data-high	22	
Intersegment	11001011	30	
Intersegment adding immediate to SP	11001010 data-low data-high	33	
JE/JZ = Jump on equal/zero	01110100 disp	4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100 disp	4/13	
JLE/JNG = Jump on less or equal/not greater	01111110 disp	4/13	
JB/JNAE = Jump on below/not above or equal	01110010 disp	4/13	
JBE/JNA = Jump on below or equal/not above	01110110 disp	4/13	
JP/JPE = Jump on parity/parity even	01111010 disp	4/13	
JO = Jump on overflow	01110000 disp	4/13	
JS = Jump on sign	01111000 disp	4/13	
JNE/JNZ = Jump on not equal/not zero	01110101 disp	4/13	
JNL/JGE = Jump on not less/greater or equal	01111101 disp	4/13	
JNLE/JG = Jump on not less or equal/greater	01111111 disp	4/13	
JNB/JAE = Jump on not below/above or equal	01110011 disp	4/13	
JNBE/JA = Jump on not below or equal/above	01110111 disp	4/13	
JNP/JPO = Jump on not par/par odd	01111011 disp	4/13	
JNO = Jump on not overflow	01110001 disp	4/13	
JNS = Jump on not sign	01111001 disp	4/13	
JCXZ = Jump on CX zero	11100011 disp	5/15	
LOOP = Loop CX times	11100010 disp	6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001 disp	6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000 disp	6/16	
ENTER = Enter Procedure	11001000 data-low data-high L		
L = 0		15	
L = 1		25	
L > 1		22+16(n−1)	
LEAVE = Leave Procedure	11001001	8	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
CONTROL TRANSFER (Continued) **INT = Interrupt:**			
Type specified	11001101 | type	47	

Type 3	11001100	45	if INT. taken/
			if INT. not taken

INTO = Interrupt on overflow	11001110	48/4	
IRET = Interrupt return	11001111	28	
BOUND = Detect value out of range	01100010 | mod reg r/m	33-35	
PROCESSOR CONTROL			
CLC = Clear carry	11111000	2	
CMC = Complement carry	11110101	2	
STC = Set carry	11111001	2	
CLD = Clear direction	11111100	2	
STD = Set direction	11111101	2	
CLI = Clear interrupt	11111010	2	
STI = Set interrupt	11111011	2	
HLT = Halt	11110100	2	
WAIT = Wait	10011011	6	if $\overline{\text{test}}$ = 0
LOCK = Bus lock prefix	11110000	2	
ESC = Processor Extension Escape	11011TTT | mod LLL r/m (TTT LLL are opcode to processor extension)	6	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

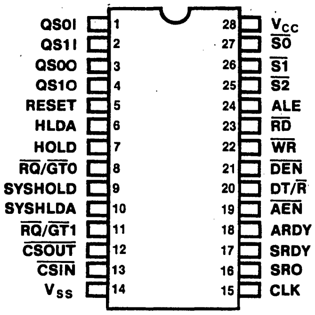
The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

PRELIMINARY

82188
INTEGRATED BUS CONTROLLER FOR 8086, 8088, 80186, 80188 PROCESSORS

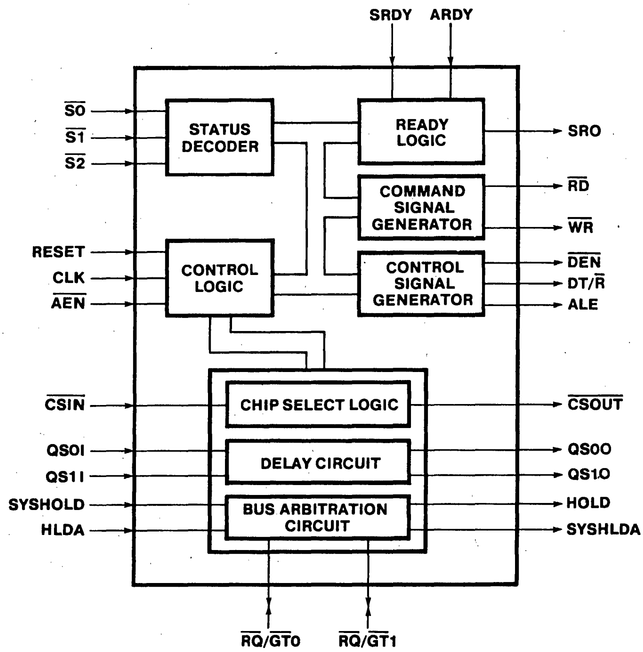
- **Provides Flexibility in System Configurations**
  - **— Supports 8087 Numerics Coprocessor in 8 MHz 80186 and 80188 Systems**
  - **— Provides a Low-cost Interface for 8086, 8088 Systems to an 82586 LAN Coprocessor or 82730 Text Coprocessor**
- **Facilitates Interface to one or more Multimaster Busses**
- **Supports Multiprocessor, Local Bus Systems**
- **Allows use of 80186, 80188 High-Integration Features**
- **3-State, Command Output Drivers**
- **Available in EXPRESS**
  - **— Standard Temperature Range**
  - **— Extended Temperature Range**
- **Available in Plastic DIP or Cerdip Package**
  (See Packaging Spec., Order #231369)

The 82188 Integrated Bus Controller (IBC) is a 28-pin HMOS III component for use with 80186, 80188, 8086 and 8088 systems. The IBC provides command and control timing signals plus a configurable $\overline{RQ}/\overline{GT}$ ⟷ HOLD-HLDA converter. The device may be used to interface an 8087 Numerics Coprocessor with an 80186 or 80188 Processor. Also, an 82586 Local Area Network (LAN) Coprocessor or 82730 Text Coprocessor may be interfaced to an 8086 or 8088 with the IBC.



231051-1

Figure 1. 82188 Pin Configuration



231051-2

Figure 2. 82188 Block Diagram

October 1986
Order Number: 231051-004

PIN DESCRIPTIONS

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$\overline{S0}$
$\overline{S1}$
$\overline{S2}$</td><td>27
26
25</td><td>I</td><td><b>Status Input Pins</b>
$\overline{S0}$–$\overline{S2}$ correspond to the status pins of the CPU.
The 82188 uses the status lines to detect and identify the processor bus cycles. The 82188 decodes $\overline{S0}$–$\overline{S2}$ to generate the command and control signals. $\overline{S0}$–$\overline{S2}$ are also used to insert 3 wait states into the SRO line during the first 256 80186 bus cycles after RESET. A HIGH input on all three lines indicates that no bus activity is taking place. The status input lines contain weak internal pull-up devices.
<table>
<tr><th>$\overline{S2}$</th><th>$\overline{S1}$</th><th>$\overline{S0}$</th><th>Bus Cycle Initiated</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>interrupt acknowledge</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>read I/O</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>write I/O</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>halt</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>instruction fetch</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>read data from memory</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>write data to memory</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>passive (no bus cycle)</td></tr>
</table></td></tr>
<tr><td>CLK</td><td>15</td><td>I</td><td><b>CLOCK</b>
CLK is the clock signal generated by the CPU or clock generator device. CLK edges establish when signals are sampled and generated.</td></tr>
<tr><td>RESET</td><td>5</td><td>I</td><td><b>RESET</b>
RESET is a level triggered signal that corresponds to the system reset signal. The signal initializes an internal bus cycle counter, thus enabling the 82188 to insert internally generated wait states into the SRO signal during system initialization. The 82188 mode is also determined during RESET. $\overline{RD}$, $\overline{WR}$, and $\overline{DEN}$ are driven HIGH during RESET regardless of $\overline{AEN}$. RESET is active HIGH.</td></tr>
<tr><td>$\overline{AEN}$</td><td>19</td><td>I</td><td><b>Address Enable</b>
This signal enables the system command lines when active. If $\overline{AEN}$ is inactive (HIGH), $\overline{RD}$, $\overline{WR}$, and $\overline{DEN}$ will be tri-stated and ALE will be driven LOW (DT/$\overline{R}$ will not be effected). $\overline{AEN}$ is an asynchronous signal and is active LOW.</td></tr>
<tr><td>ALE</td><td>24</td><td>O</td><td><b>Address Latch Enable</b>
This signal is used to strobe an address into address latches. ALE is active HIGH and latch should occur on the HIGH to LOW transition. ALE is intended for use with transparent D-type latches.</td></tr>
<tr><td>$\overline{DEN}$</td><td>21</td><td>O</td><td><b>Data Enable</b>
This signal is used to enable data transceivers located on either the local or system data bus. The signal is active LOW. $\overline{DEN}$ is tri-stated when $\overline{AEN}$ is inactive.</td></tr>
<tr><td>DT/$\overline{R}$</td><td>20</td><td>O</td><td><b>Data TRANSMIT/RECEIVE</b>
This signal establishes the direction of data flow through the data transceivers. A HIGH on this line indicates TRANSMIT (write to I/O or memory) and a LOW indicates RECEIVE (Read from I/O or memory).</td></tr>
</table>

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{\text{RD}}$	23	O	**READ** This signal instructs an I/O or memory device to drive its data onto the data bus. The $\overline{\text{RD}}$ signal is similiar to the $\overline{\text{RD}}$ signal of the 80186(80188) in Non-Queue-Status Mode. $\overline{\text{RD}}$ is active LOW and is tri-stated when $\overline{\text{AEN}}$ is inactive.
$\overline{\text{WR}}$	22	O	**WRITE** This signal instructs an I/O or memory device to record the data presented on the data bus. The $\overline{\text{WR}}$ signal is similiar to the $\overline{\text{WR}}$ signal of the 80186(80188) in Non-Queue-Status Mode. $\overline{\text{WR}}$ is active LOW and is tri-stated when $\overline{\text{AEN}}$ is inactive.
HOLD	7	O	**HOLD** The HOLD signal is used to request bus control from the 80186 or 80188. The request can come from either the 8087 ($\overline{\text{RQ}}/\overline{\text{GTO}}$) or from the third processor (SYSHOLD). The signal is active HIGH.
HLDA	6	I	**HOLD Acknowledge** 80186 MODE–This line serves to translate the HLDA output of the 80186(80188) to the appropriate signal of the device requesting the bus. HLDA going active (HIGH) indicates that the 80186 has relinquished the bus. If the requesting device is the 8087, HLDA will be translated into the grant pulse of the $\overline{\text{RQ}}/\overline{\text{GTO}}$ line. If the requesting device is the optional third processor, HLDA will be routed into the SYSHLDA line. This pin also determines the mode in which the 82188 will operate. If this line is HIGH during the falling edge of RESET, the 82188 will enter the 8086 mode. If LOW, the 82188 will enter the 80186 mode. For 8086 mode, this pin should be strapped to V_{CC}.
$\overline{\text{RQ}}/\overline{\text{GTO}}$	8	I/O	**Request/Grant O** $\overline{\text{RQ}}/\overline{\text{GTO}}$ is connected to $\overline{\text{RQ}}/\overline{\text{GTO}}$ of the 8087 Numeric Coprocessor. When initiated by the 8087, $\overline{\text{RQ}}/\overline{\text{GTO}}$ will be translated to HOLD-HLDA to acquire the bus from the 80186(80188). This line is bidirectional, and is active LOW. $\overline{\text{RQ}}/\overline{\text{GTO}}$ has a weak internal pull-up device to prevent erroneous request/grant signals.
$\overline{\text{RQ}}/\overline{\text{GT1}}$	11	I/O	**Request/Grant 1** 80186 Mode–In 80186 Mode, $\overline{\text{RQ}}/\overline{\text{GT1}}$ allows a third processor to take control of the local bus when the 8087 has bus control. For a HOLD-HLDA type third processor, the 82188's $\overline{\text{RQ}}/\overline{\text{GT1}}$ line should be connected to the $\overline{\text{RQ}}/\overline{\text{GT1}}$ line of the 8087. 8086 MODE–In 8086 Mode, $\overline{\text{RQ}}/\overline{\text{GT1}}$ is connected to either $\overline{\text{RQ}}/\overline{\text{GTO}}$ or $\overline{\text{RQ}}/\overline{\text{GT1}}$ of the 8086. $\overline{\text{RQ}}/\overline{\text{GT1}}$ will start its request/grant sequence when the SYSHOLD line goes active. In 8086 Mode, $\overline{\text{RQ}}/\overline{\text{GT1}}$ is used to gain bus control from the 8086 or 8088. $\overline{\text{RQ}}/\overline{\text{GT1}}$ is a bidirectional line and is active LOW. This line has a weak internal pull-up device to prevent erroneous request/grant signals.

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
SYSHOLD	9	I	**System Hold** 80186 MODE–SYSHOLD serves as a hold input for an optional third processor in an 80186(80188)-8087 system. If the 80186(80188) has bus control, SYSHOLD will be routed to HOLD to gain control of the bus. If the 8087 has bus control, SYSHOLD will be translated to $\overline{RQ}/\overline{GT}1$ to gain control of the bus. 8086 MODE–SYSHOLD serves as a hold input for a coprocessor in an 8086 or 8088 system. SYSHOLD is translated to $\overline{RQ}/\overline{GT}1$ of the 82188 to allow the coprocessor to take control of the bus. SYSHOLD may be an asynchronous signal.
SYSHLDA	10	O	**System Hold Acknowledge** SYSHLDA serves as a hold acknowledge line to the processor or coprocessor connected to it. The device connected to the SYSHOLD-SYSHLDA lines is allowed the bus when SYSHLDA goes active (HIGH).
SRDY	17	I	**Synchronous Ready** The SRDY input serves the same function as SRDY of the 80186(80188). The 82188 combines SRDY with ARDY to form a synchronized ready output signal (SRO). SRDY must be synchronized external to the 82188 and is active HIGH. If tied to V_{CC}, SRO will remain active (HIGH) after the first 256 80186 cycles following RESET. If only ARDY is to be used, SRDY should be tied LOW.
ARDY	18	I	**Asynchronous Ready** The ARDY input serves the same function as ARDY of the 80186(80188). ARDY may be an asynchronous input, and is active HIGH. Only the rising edge of ARDY is synchronized by the 82188. The falling edge must be synchronized external to the 82188. If connected to V_{CC}, SRO will remain active (HIGH) after the first 256 80186 bus cycles following RESET. If only SRDY is to be used, ARDY should be connected LOW.
SRO	16	O	**Synchronous READY Output** SRO provides a synchronized READY signal which may be interfaced directly with the SRDY of the 80186(80188) and READY of the 8087. The SRO signal is an accumulation of the synchronized ARDY signal, the SRDY signal, and the internally generated wait state signal.
QS0I QS1I	1 2	I	**Queue-Status Inputs** QS0I, QS1I are connected to the Queue-Status lines of the 80186(80188) to allow synchronization of the queue-status signals to 8087 timing requirements.
QS0O QS1O	3 4	O	**Queue-Status Outputs** QS0O, QS1O are connected to the queue-status pins of the 8087. The signals produced meet 8087 Queue-Status input requirements.

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
CSIN	13	I	**Chip-Select Input** CSIN is connected to one of the chip-select lines of the 80186(80188). CSIN informs the 82188 that a bank select is taking place. The 82188 routes this signal to the chip-select output (CSOUT). CSIN is active LOW. This line is not used when memory and I/O device addresses are decoded external to the 80186(80188).
CSOUT	12	O	**Chip-Select Output** This signal is used as a chip-select line for a bank of memory devices. It is active when CSIN is active or when the 8087 has bus control. CSOUT is active LOW.

FUNCTIONAL DESCRIPTION

BUS CONTROLLER

The 82188 Integrated Bus Controller (IBC) generates system control and command signals. The signals generated are determined by the Status Decoding Logic. The bus controller logic interprets status lines S0-S2 to determine what type of bus cycle is taking place. The appropriate signals are then generated by the Command and Control Signal Generators.

The Address Enable (AEN) line allows the command and control signals to be disabled. When AEN is inactive (HIGH), the command signals and DEN will be tri-stated, and ALE will be held low (DT/R will be uneffected). AEN inactive will allow other systems to take control of the bus. Control and command signals respond to a change in the AEN signal within 40 ns.

The command signals consist of RD and WR. The 82188's RD and WR signals are similiar to RD and WR of the 80186(80188) in the non-Queue-Status Mode. These command signals do not differentiate between memory and I/O devices. RD and WR can be conditioned by S2 of the 80186(80188) to obtain separate signals for I/O and memory devices.

The control commands consist of Data Enable (DEN), Data Transmit/Receive (DT/R), and Address Latch Enable (ALE). The control commands are similiar to those generated by the 80186(80188). DEN determines when the external bus should be enabled onto the local bus. DT/R determines the direction of the data transfer, and ALE determines when the address should be strobed into the latches (used for demultiplexing the address bus).

MODE SELECT

The 82188 Integrated Bus Controller (IBC) is configurable. The device has two modes: 80186 Mode and 8086 Mode. Selecting the mode of the device configures the Bus Arbitration Logic (see BUS ARBITRATION section for details). In 80186 Mode, the 82188 IBC may be used as a bus controller/interface device for an 80186(81088), 8087, and optional third processor system. In 8086 Mode, the 82188 IBC may be used as an interface device allowing a maximum mode 8086(8088) to interface with a coprocessor that uses a HOLD-HLDA bus exchange protocol.

The mode of the 82188 is determined during RESET. If the HLDA line is LOW at the falling edge of RESET (as in the case when tied to the HLDA line of the 80186 or 80188), the 82188 will enter into 80186 Mode. If the HLDA line is HIGH at the falling edge of RESET, the 82188 will enter 8086 Mode. In 8086 Mode, only the Bus Arbitration Logic is used. The eight pins used in 8086 Mode are: SYSHOLD, SYSHLDA, HLDA, CLK, RESET, RQ/GT1, V_{CC}, and V_{SS}. The other pins may be left unconnected.

BUS ARBITRATION

The Bus Exchange Logic interfaces up to three sets of bus exchange signals:

- HOLD-HLDA
- SYSHOLD-SYSHLDA
- RQ/GT0 (RQ/GT1)

This logic executes translating, routing, and arbitrating functions. The logic translates HOLD-HLDA signals to RQ/GT signals and RQ/GT signals to HOLD-HLDA signals. The logic also determines which set of bus exchange signals are to be interfaced. The mode of the 82188 and the priority of the devices requesting the bus determine the routing of the bus exchange signals.

80186 MODE

In 80186 Mode, a system may have three potential bus masters: the 80186 or 80188 CPU, the 8087 Numerics Coprocessor, and a third processor (such as the 82586 LAN or 82730 Text Coprocessor). The third processor may have either a HOLD-HLDA or $\overline{\text{RQ}}/\overline{\text{GT}}$ bus exchange protocol. The possible bus exchange signal connections and paths for 80186 Mode are shown in Figures 3 & 4 and Tables 1 & 2, respectively. If no HOLD-HLDA type third processor is used, SYSHOLD should be tied LOW to prevent an erroneous SYSHOLD signal. In 80186 mode, the bus priorities are:

Highest Priority	Third Processor
Second Highest Priority	8087
Default Priority	80186

— THREE-PROCESSOR SYSTEM OPERATION (HOLD-HLDA TYPE THIRD PROCESSOR)

In the configuration shown in Figure 3, the third processor requests the bus by sending SYSHOLD HIGH. The 82188 will route (and translate if necessary) the request to the current bus master. This includes routing the request to HOLD if the 80186(80188) is the current bus master or routing and translating the request to $\overline{\text{RQ}}/\overline{\text{GT}}$1 if the 8087 is in control of the bus. The third processor's request is not passed through the 8087 if the 80186 is the bus master (see Table 1).

The 8087 requests the bus using $\overline{\text{RQ}}/\overline{\text{GT}}$0. The request pulse from the 8087 will be translated and routed to HOLD if the 80186 is the bus master. If the third processor has control of the bus, the grant pulse to the 8087 will be delayed until the third processor relinquishes the bus (sending SYSHOLD LOW). In this case, HOLD will remain HIGH during the third processor-to-8087 bus control transfer. The 80186 will not be granted the bus until both coprocessors have released it.

Table 1. Bus Exchange Paths (80186 Mode) (HOLD-HLDA Type 3rd Proc)

Requesting Device	Current Bus Master		
	80186	**8087**	**3rd Proc**
80186	n/a	n/a	n/a
8087	$\overline{\text{RQ}}/\overline{\text{GT}}0 \longleftrightarrow \frac{\text{HOLD}}{\text{HLDA}}$	n/a	n/a
3rd Proc	$\frac{\text{SYSHOLD}}{\text{SYSHLDA}} \longleftrightarrow \frac{\text{HOLD}}{\text{HLDA}}$	$\frac{\text{SYSHOLD}}{\text{SYSHLDA}} \longleftrightarrow \overline{\text{RQ}}/\overline{\text{GT}}1$	n/a

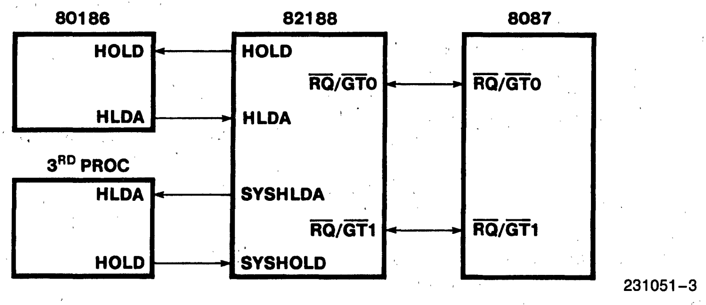


Figure 3. Bus Exchange Signal Connections (80186 Mode) for a Three Local Processor System (HOLD-HLDA Type 3rd Proc)

Table 2. Bus Exchange Paths (80186 Mode) ($\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}$ Type 3rd Proc)

Requesting Device	Current Bus Master		
	80186	**8087**	**3rd Proc**
80186	n/a	n/a	n/a
8087	$\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}0 \longleftrightarrow \frac{\mathrm{HOLD}}{\mathrm{HLDA}}$	n/a	n/a
3rd Proc	$\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}1 \longleftrightarrow \overline{\mathrm{RQ}}/\overline{\mathrm{GT}}0 \longleftrightarrow \frac{\mathrm{HOLD}}{\mathrm{HLDA}}$	$\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}1$	n/a

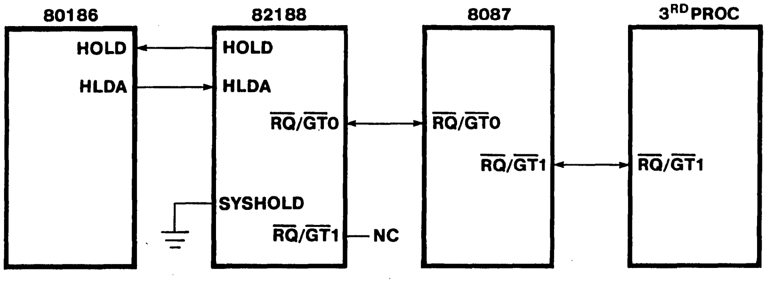


Figure 4.
Bus Exchange Signal Connections (80186 Mode) for a Three Local Processor System ($\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}$ Type 3rd Proc)

When the bus is requested from the 80186(80188), a bus priority decision is made. This decision is made when the HLDA line goes active. Upon receipt of the HLDA signal, the highest-priority requesting device will be acknowledged the bus. For example, if the 8087 initially requested the bus, the bus will be granted to the third processor if SYSHOLD became active before HLDA was received by the 82188. In this case, the grant pulse to the 8087 will be delayed until the third processor relinquishes the bus.

— THREE-PROCESSOR SYSTEM OPERATION ($\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}$ TYPE THIRD PROCESSOR)

In the configuration shown in Figure 4, the third processor requests the bus by initiating a request/grant sequence with the 8087's $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}1$ line. The 8087 will grant the bus if it is the current bus master or will pass the request on if the 80186 is the current bus master (see Table 2). In this configuration, the 82188's Bus Arbitration Logic translates $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}0$ to HOLD-HLDA. The 8087 provides the bus arbitration in this configuration.

8086 MODE

The 8086 Mode allows an 8086, 8088 system to contain both $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}$ and HOLD-HLDA type coprocessors simultaneously. In 8086 Mode, two possible bus masters may be interfaced by the 82188; an 8086 or 8088 CPU and a coprocessor which uses a HOLD-HLDA bus exchange protocol (typically an 82586 LAN Coprocessor or an 82730 Text Coprocessor). The bus exchange signal connections for 8086 Mode are shown in Figure 5. Bus arbitration signals used in the 8086 Mode are:

- $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}1$
- SYSHOLD
- SYSHLDA

In 8086 Mode, no arbitration is necessary since only two devices are interfaced. The coprocessor has bus priority over the 8086(8088). SYSHOLD-SYSHLDA are routed and translated directly to $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}1$. $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}1$ of the 82188 may be tied to either $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}0$ or $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}1$ of the 8086(8088).

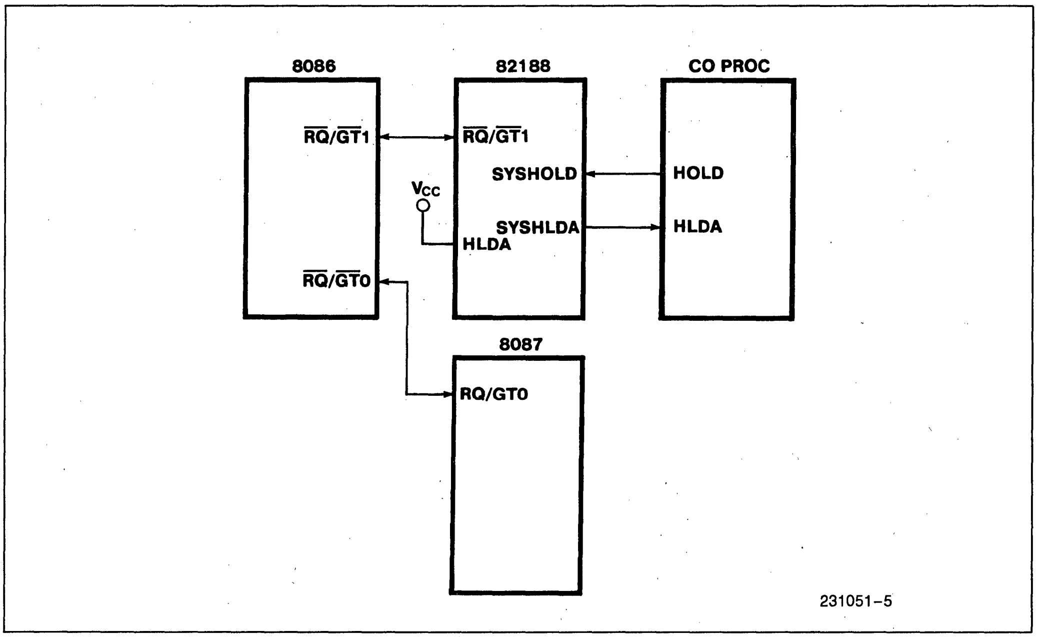


Figure 5. Bus Exchange Signal Connections (8086 Mode)

QUEUE-STATUS DELAY

The Queue-Status Delay logic is used to delay the queue-status signals from the 80186(80188) to meet 8087 queue-status timing requirements. QS0I, QS1I correspond to the queue-status lines of the 80186(80188). The 82188 delays these signals by one clock phase. The delayed signals are interfaced to the 8087 queue-status lines by QS0O, QS1O.

CHIP-SELECT

The Chip-Select Logic allows the utilization of the chip select circuitry of the 80186(80188). Normally, this circuitry could not be used in an 80186(80188)-8087 system since the 8087 contains no chip select circuitry. The Chip-Select Logic contains two external connections: Chip-Select Input ($\overline{\text{CSIN}}$) and Chip-Select Output ($\overline{\text{CSOUT}}$). $\overline{\text{CSOUT}}$ is active when either $\overline{\text{CSIN}}$ is active or when the 8087 has control of the bus.

By using $\overline{\text{CSOUT}}$ to select memory containing data structures, no external decoding is necessary. The 80186 may gain access to this memory bank through the $\overline{\text{CSIN}}$ line while the 8087 will automatically obtain access when it becomes the bus master. Note that this configuration limits the amount of memory accessible by the 8087 to the physical memory bank selected by $\overline{\text{CSOUT}}$. Systems where the 8087 must access the full 1 Megabyte address space must use an external decoding scheme.

READY

The Ready logic allows two types of Ready signals: a Synchronous Ready Signal (SRDY) and an Asynchronous Ready Signal (ARDY). These signals are similiar to SRDY and ARDY of the 80186. Wait states will be inserted when both SRDY and ARDY are LOW. Inserting wait states allows slower memory and I/O devices to be interfaced to the 80186(80188)-8087 system.

ARDY's LOW-to-HIGH transition is synchronized to the CPU clock by the 82188. The 82188 samples ARDY at the beginning of T2, T3 and Tw until sampled HIGH. Note that ARDY of the 82188 is sampled one phase earlier than ARDY of the 80186. ARDY's falling edge must be synchronous to the CPU clock. ARDY allows an easy interface with devices that emit an asynchronous ready signal.

The SRDY signal allows direct interface to devices that emit a synchronized ready signal. SRDY must be synchronized to the CPU clock for both of its transitions. SRDY is sampled in the middle of T2, T3 and in the middle of each Tw. An 82188-80186(80188)'s SRDY setup time is 30 ns longer than the 80186(80188)'s SRDY setup time. SRDY eliminates the half-clock cycle penalty necessary for ARDY to be internally sychronized.

The sychronized ready output (SRO) is the accumulation of SRDY, ARDY, and the internal wait-state generator. SRO should be connected to SRDY of the 80186(80188) (with 80186(80188)'s ARDY tied LOW), and READY of the 8087.

SRDY	ARDY	SRO
0	0	0
1	X	1
X	1	1

The internal wait state generator allows for synchronization between the 80186(80188) and 8087 in 80186 mode. Upon RESET, the 82188 automatically inserts 3 wait-states per 80186(80188) bus cycle, overlapped with any externally produced wait-states created by ARDY and SRDY.

Since the 8087 has no provision for internal wait-state generation, only externally created wait states will be effective. The 82188, upon RESET, will inject 3 wait states for each of the first 256 80186(80188) bus cycles onto the SRO line. This will allow the 8087 to match the 80186(80188)'s timing.

The internally-generated wait states are overlapped with those produced by the SRDY and ARDY lines. Overlapping the injected wait states insures a minimum of three wait states for the first 256 80186(80188) bus cycles after RESET. Systems with a greater number of wait states will not be effected. Internal wait state generation by the 82188 will stop on the 256th 80186(80188) bus cycle after RESET. To maintain sychronization between the 80186(80188) and 8087, the following conditions are necessary:

- The 80186(80188)'s control block must be mapped in I/O space before it is written to or read from.
- All memory chip-select lines must be set to 0 WAIT STATES, EXTERNAL READY ALSO USED within the first 256 80186(80188) bus cycles after RESET.

An equivalent READY logic diagram is shown in Figure 6.

SYSTEM CONSIDERATIONS

In any 82188 configuration, clock compatibility must be considered. Depending on the device, a 50% or a 33% duty-cycle clock is needed. For example, the 80186 and 80188 (as well as the 82188, 82586, and 82730) requires a 50% duty-cycle clock. The 8086, 8088 and their 'kit' devices' (8087, 8089, 8288, and 8289) clock requirements, on the other hand, require a 33% duty-cycle clock signal. The system designer must make sure clock requirements of all the devices in the system are met.

Figure 7 demonstrates the usage of the 82188 in 80186 Mode where it is used to interface an 8087 into an 80186 system.

Status bit six (S6) from the main processor (8086, 8088, 80186, or 80188) is used by the 8087 to track the instruction flow. S6 is multiplexed with address bit 19 (A19). If the third processor generates only 16 bits of address, S6 is not generated. A19/S6 must be driven high by external circuitry during the status portion of bus cycles controlled by the third processor.

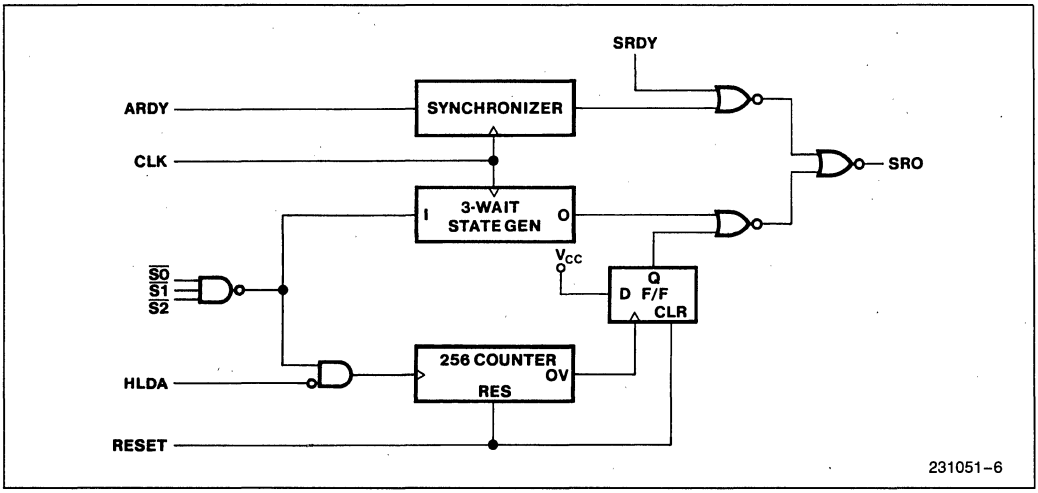
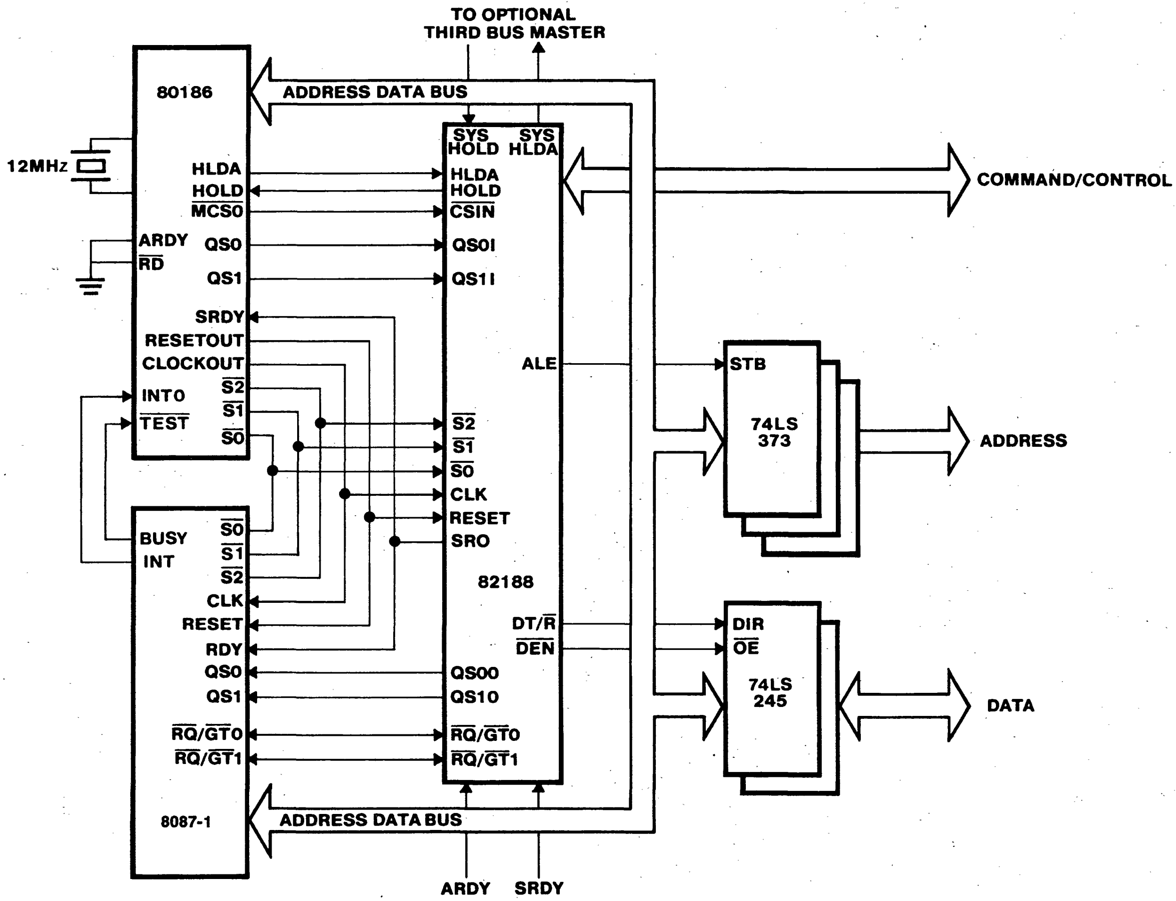


Figure 6. Equivalent 82188 READY Circuit



231051-7

Figure 7.
80186-6/8087-2 System Using the 82188 in 80186 Mode

ABSOLUTE MAXIMUM RATINGS *

Temperature Under Bias	0°C to 70°C
Storage Temperature	−65°C to 150°C
Case Temperature	0°C to +85°C
Voltage on any Pin with Respect to GND	−1.0V to 7.0V
Power Dissipation	0.7 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

DC CHARACTERISTICS

(V_{CC} = 5V ± 10%, T_A = 0°C to 70°C, T_{CASE} = 0°C to +85°C)

Symbol	Parameter	Min	Max	Units	Test Cond.
V_{IL}	Input Low Voltage	−0.5	+0.8	volts	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	volts	
V_{OL}	Output Low Voltage		0.45	volts	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		volts	I_{OH} = −400 μA
I_{CC}	Power Supply Current		100	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	$OV<V_{IN}<V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45<V_{OUT}<V_{CC}$
V_{CLI}	CLK Input Low Voltage	−0.5	+0.6	volts	
V_{CHI}	CLK Input High Voltage	3.9	V_{CC} +1.0	volts	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

AC CHARACTERISTICS

(V_{CC} = 5V ± 10%, T_A = 0°C to 70°C, T_{CASE} = 0°C to +85°C)

TIMING REQUIREMENTS

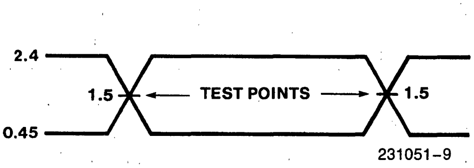
Symbol	Parameter	Min	Max	Units	Notes
TCLCL	Clock Period	125	500	ns	
TCLCH	Clock LOW Time	½TCLCL-7.5		ns	
TCHCL	Clock HIGH Time	½TCLCL-7.5		ns	
TARYHCL	ARDY Active Setup Time	20		ns	
TCHARYL	ARDY Hold Time	15		ns	8
TARYLCH	ARDY Inactive Setup Time	35		ns	
TSRYHCL	SRDY Input Setup Time	65,50		ns	1
TSVCH	STATUS Active Setup Time	55		ns	
TSXCL	STATUS Inactive Setup Time	50		ns	
TQIVCL	QS0I, QS1I Setup Time	15		ns	
THAVGV	HLDA Setup Time	50		ns	
TSHVCL	SYSHOLD Asynchronous Setup Time	25		ns	
TGVCH	$\overline{RQ}/\overline{GT}$ Input Setup Time	0		ns	6

TIMING RESPONSES

Symbol	Parameter	Min	Max	Units	Notes
TSVLH	STATUS Valid to ALE Delay		30	ns	4
TCHLL	ALE Inactive Delay		30	ns	
TCLML	$\overline{RD}$, $\overline{WR}$ Active Delay	10	70	ns	
TCLMH	$\overline{RD}$, $\overline{WR}$ Inactive Delay	10	55	ns	
TSVDTV	STATUS to DT/$\overline{R}$ Delay		30	ns	3
TCLDTV	DT/$\overline{R}$ Active Delay		55	ns	3
TCHDNV	$\overline{DEN}$ Active Delay	10	55	ns	
TCHDNX	$\overline{DEN}$ Inactive Delay	10	55	ns	
TCLQOV	QS0O, QS1O Delay	5	50	ns	
TCHHV	HOLD Delay		50	ns	2,6
TCLSAV	SYSHLDA Delay		50	ns	6
TCLGV	$\overline{RQ}/\overline{GT}$ Output Delay		40	ns	6
TGVHV	$\overline{RQ}/\overline{GT0}$ To HOLD Delay		50	ns	2,6
TCLLH	ALE Active Delay		30	ns	4
TAELCV	Command Enable Delay		40	ns	
TAEHCX	Command Disable Delay		40	ns	
TCHRO	SRO Output Delay	5	30	ns	5,6
TSRYHRO	SRDY To SRO Delay		30	ns	5
TCSICSO	$\overline{CSIN}$ To $\overline{CSOUT}$ Delay		30	ns	
TCLCSOV	CLK Low to $\overline{CSOUT}$ Delay	10		ns	
TCLCSOH	CLK Low to $\overline{CSOUT}$ Inactive Delay	10		ns	

NOTES (applicable to both spec listing and timing diagrams):
1. TSRYHOL = (80186's) TSRYCL + 30 ns=65 ns for 6 MHz operation and 50 ns for 8 MHz operation.
2. Timing not tested.
3. DT/$\overline{R}$ will be asserted to the latest of TSVDTV & TCLDTV.
4. ALE will be asserted to the latest of TSVLH & TCLLH.
5. SRO will be asserted to the latest of TCHRO & TSRYHRO.
6. CL = 20–100 pF
7. Address/Data bus shown for reference only.
8. The falling edge of ARDY must be synchronized to CLK.

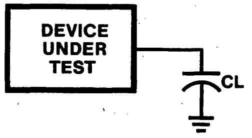
A.C. TESTING INPUT, OUTPUT WAVEFORM



231051-9

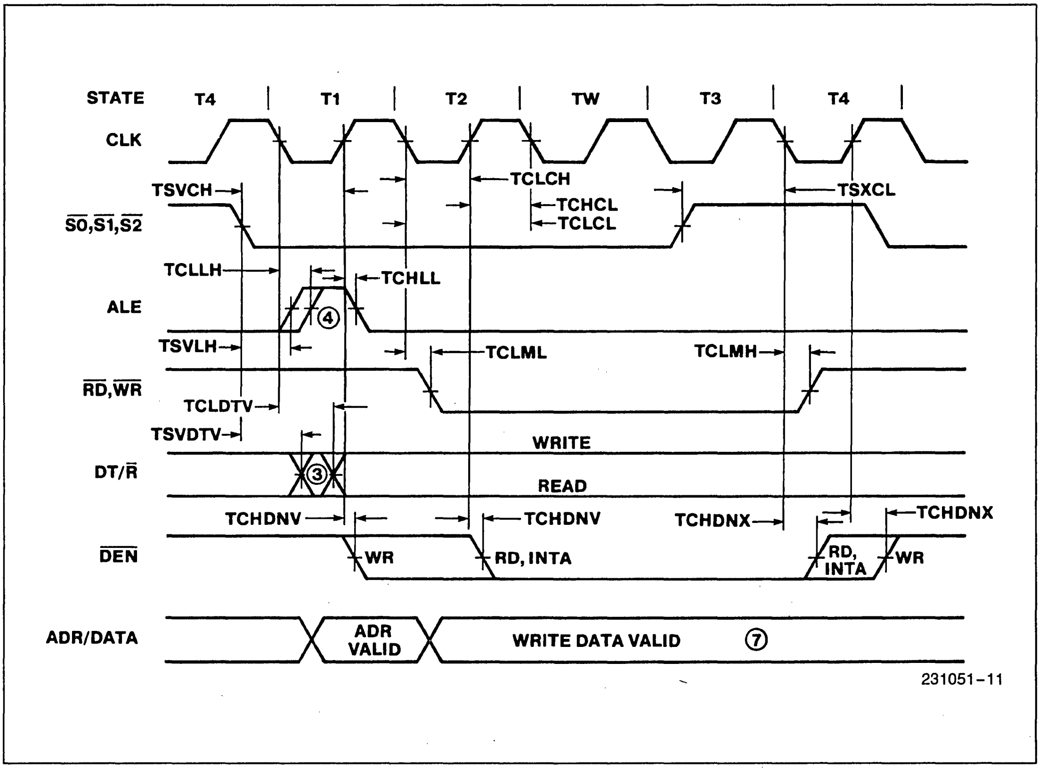
A.C. Testing: Inputs are driven at 2.4V for a Logic '1' and 0.45V for a Logic '0'.

A.C. TESTING LOAD CIRCUIT

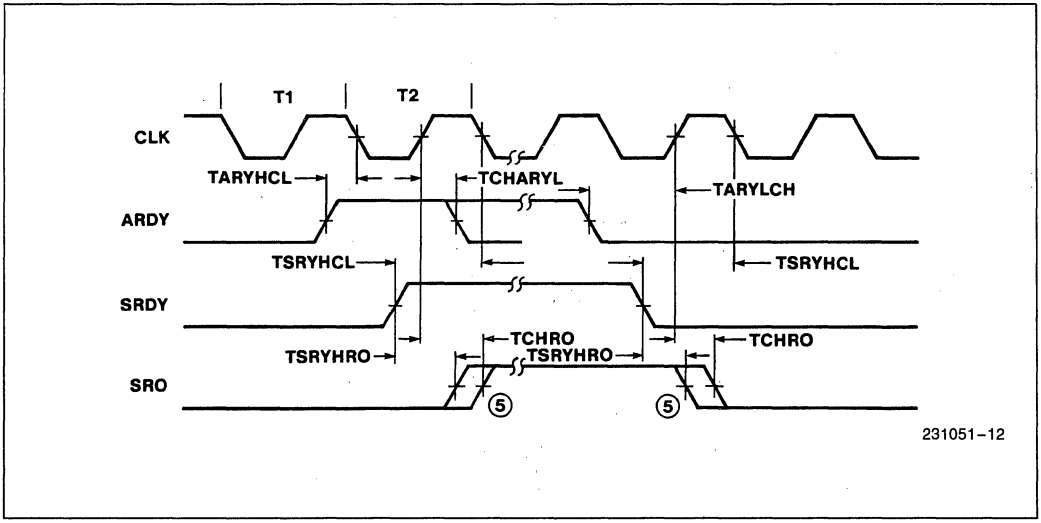


231051-10

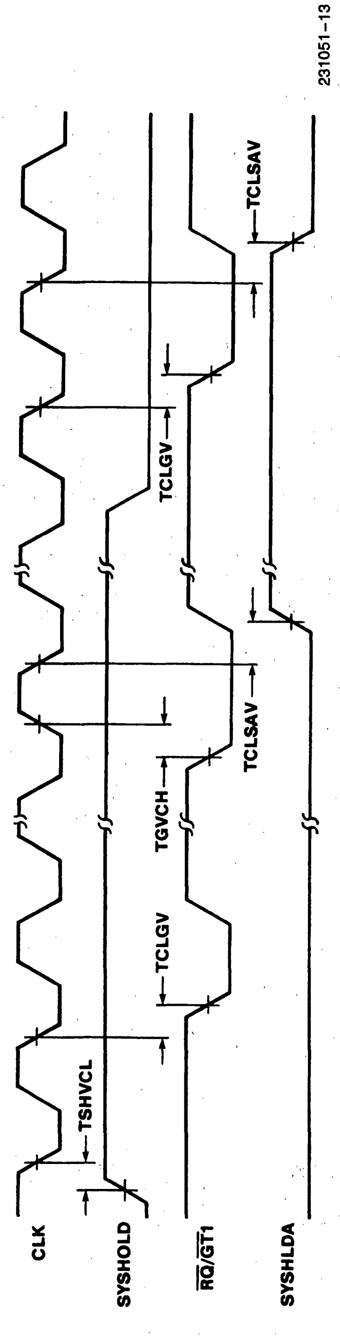
C_L includes Jig Capacitance
C_L = 20–200 pF unless otherwise noted



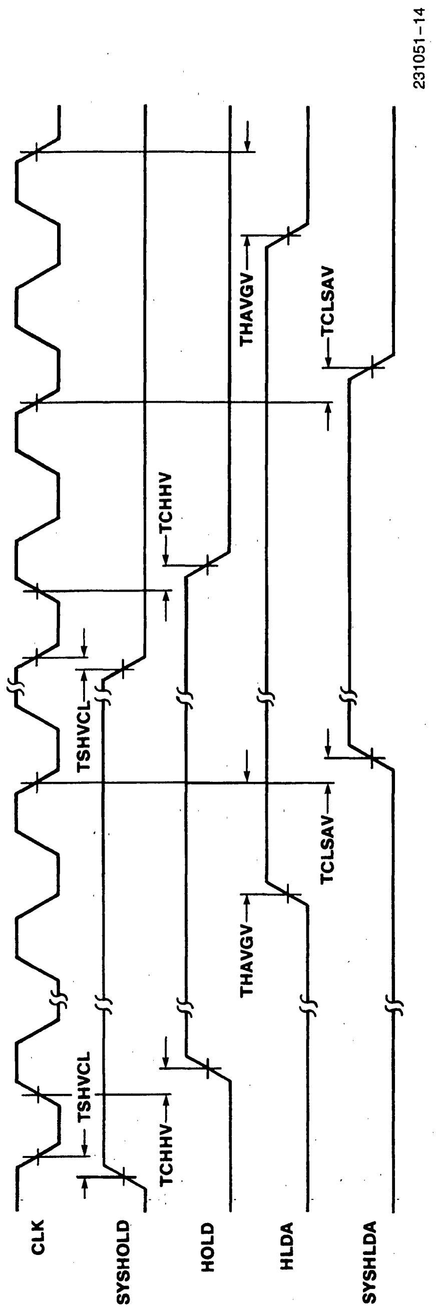
Command and Control Waveforms-80186 Mode



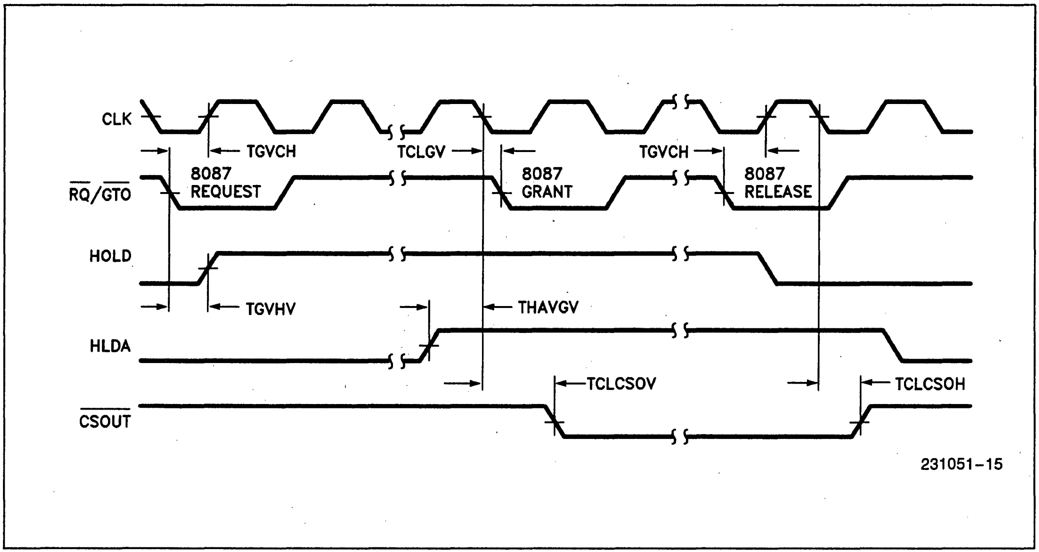
READY Timing-80186 Mode



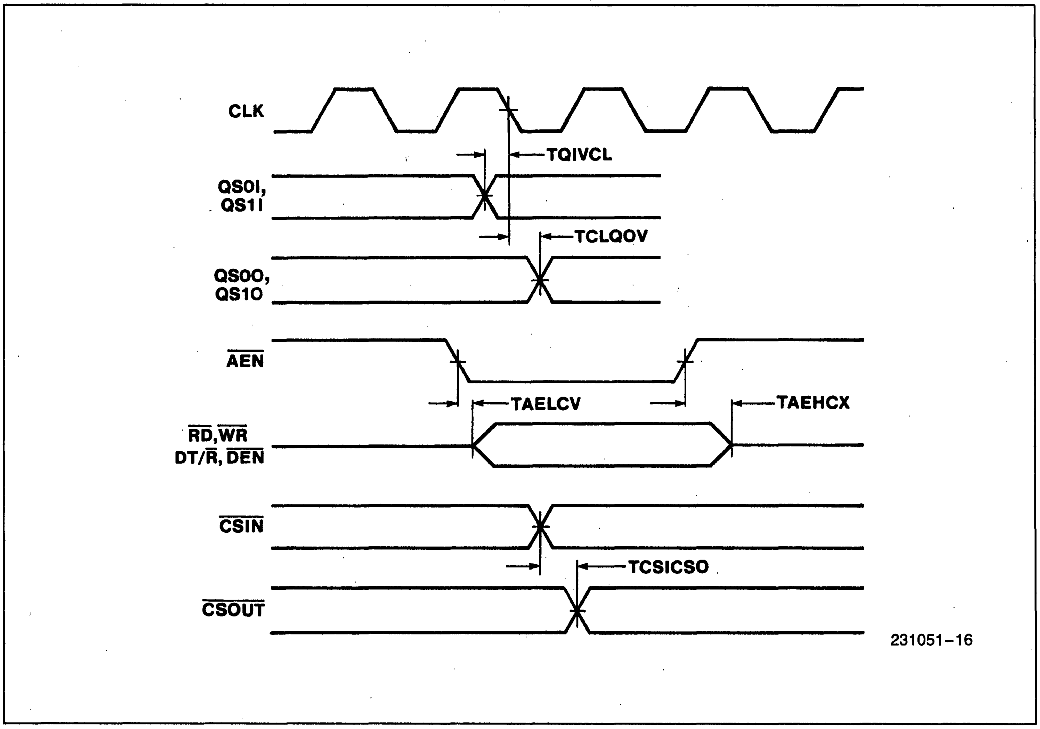
SYSHOLD-SYSHLDA to $\overline{RQ}/\overline{GT}1$ Timing–80186 Mode and 8086 Mode



SYSHOLD-SYSHLDA To HOLD-HLDA Timing–80186 Mode



RQ/GT0 to HOLD-HLDA Timing-80186 Mode



Queue Status, $\overline{\text{ALE}}$, Chip Select Delay Timing-80186 Mode

intel®

APPLICATION NOTE

AP-186

November 1987

Introduction to the 80186 Microprocessor

Order Number: 210973-005

1.0 INTRODUCTION

As state of the art technology has increased the number of transistors possible on a single integrated circuit, these devices have attained new, higher levels of both performance and functionality. Riding this crest are the Intel 80186 and 80286 microprocessors. While the 80286 has added memory protection and management to the basic 8086 architecture, the 80186 has integrated six separate functional blocks into a single device.

The purpose of this note is to explain, through example, the use of the 80186 with various peripheral and memory devices. Because the 80186 integrates a DMA unit, timer unit, interrupt controller unit, bus controller unit and chip select and ready generation unit with the CPU on a single chip (see Figure 1), system construction is simplified since many of the peripheral interfaces are integrated onto the device.

The 80186 family actually consists of two processors: the 80186 and 80188. The only difference between the two processors is that the 80186 maintains a 16-bit external data bus while the 80188 has an 8-bit external data bus. Internally, they both implement the same processor with the same integrated peripheral components. Thus, except where noted, all 80186 information in this note also applies to the 80188. The implications of having an 8-bit external data bus on the 80188 are explicitly noted in Appendix I. Any parametric values indicated in this note are taken from 80186 data sheet and refer to 8 MHz devices. Different values apply to 10 MHz devices.

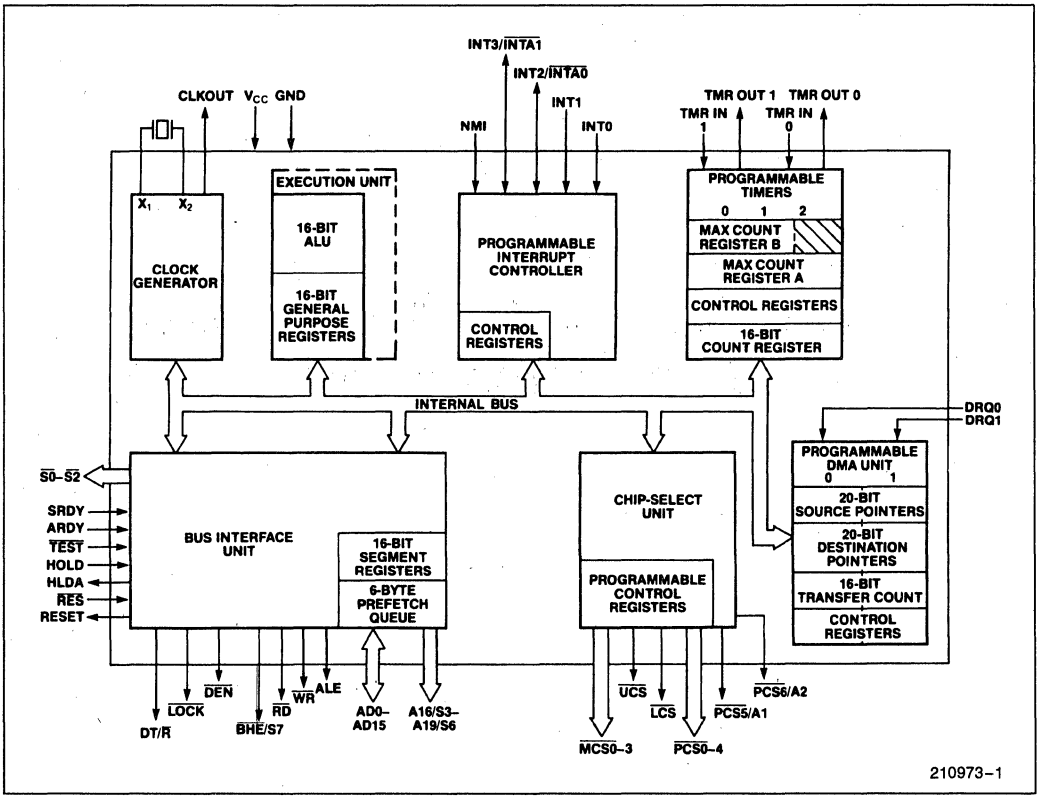


Figure 1. 80186 Block Diagram

2.0 OVERVIEW OF THE 80186

2.1 The CPU

The 80186 CPU shares a common base architecture with the 8086, 8088 and 80286. It is completely object code compatible with the 8086/88. This architecture features four 16-bit general purpose registers (AX, BX, CX, DX) which may be used as operands in most arithmetic operations in either 8 or 16 bit units. It also features four 16-bit "pointer" registers (SI, DI, BP, SP) which may be used both in arithmetic operations and in accessing memory based variables. Four 16-bit segment registers (CS, DS, SS, ES) are provided allowing simple memory partitioning to aid construction of modular programs. Finally, it has a 16-bit instruction pointer and a 16-bit status register.

Physical memory addresses are generated by the 80186 identically to the 8086. The 16-bit segment value is left shifted 4 bits and then is added to an offset value which is derived from combinations of the pointer registers, the instruction pointer, and immediate values (see Figure 2). Any carry out of this addition is ignored. The result of this addition is a 20-bit physical address which is presented to the system memory.

The 80186 has a 16-bit ALU which performs 8 or 16-bit arithmetic and logical operations. It provides for data movement among registers, memory and I/O space. In addition, the CPU allows for high speed data transfer from one area of memory to another using string move instructions, and to or from an I/O port and memory using block I/O instructions. Finally, the CPU provides a wealth of conditional branch and other control instructions.

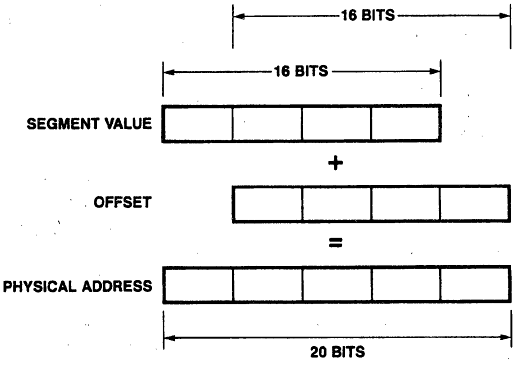
In the 80186, as in the 8086, instruction fetching and instruction execution are performed by separate units: the bus interface unit and the execution unit, respectively. The 80186 also has a 6-byte prefetch queue as does the 8086. The 80188 has a 4-byte prefetch queue as does the 8088. As a program is excecuting, opcodes are fetched from memory by the bus interface unit and placed in this queue. Whenever the execution unit requires another instruction, it takes it out of the queue. Effective processor throughput is increased by adding this queue, since the bus interface unit may continue to fetch instructions while the execution unit executes a long instruction. Then, when the CPU completes this instruction, it does not have to wait for another instruction to be fetched from memory.

2.2 80186 CPU Enhancements

Although the 80186 is completely object code compatible with the 8086, most of the 8086 instructions require fewer clock cycles to execute on the 80186 than on the 8086 because of hardware enhancements in the bus interface unit and the execution unit. In addition, the 80186 provides many new instructions which simplify assembly language programming, enhance the performance of high level language implementations, and reduce object code sizes for the 80186. A complete description of the architecture and instruction execution of the 80186 can be found in volume I of the 8086/80186 Users Manual. The algorithm for the new instructions are also given in appendix H of this note.

2.3 DMA Unit

The 80186 includes a DMA unit which provides two high speed DMA channels. This DMA unit will per-



210973-2

Figure 2. Physical Address Generation in the 80186

form transfers to or from any combination of I/O space and memory space in either byte or word units. Every DMA cycle requires two to four bus cycles, one or two to fetch the data to an internal register, and one or two to deposit the data. This allows word data to be located on odd boundaries, or byte data to be moved from odd locations to even locations. This is normally difficult, since odd data bytes are transferred on the upper 8 data bits of the 16-bit data bus, while even data bytes are transferred on the lower 8 data bits of the data bus.

Each DMA channel maintains independent 20-bit source and destination pointers which are used to access the source and destination of the data transferred. Each of these pointers may independently address either I/O or memory space. After each DMA cycle, the pointers may be independently incremented, decremented, or maintained constant. Each DMA channel also maintains a transfer count which may be used to terminate a series of DMA transfers after a pre-programmed number of transfers.

2.4 Timers

The 80186 includes a timer unit which contains 3 independent 16-bit timer/counters. Two of these timers can be used to count external events, to provide waveforms derived from either the CPU clock or an external clock of any duty cycle, or to interrupt the CPU after a specified number of timer "events". The third timer counts only CPU clocks and can be used to interrupt the CPU after a programmable number of CPU clocks, to give a count pulse to either or both of the other two timers after a programmable number of CPU clocks, or to give a DMA request pulse to the integrated DMA unit after a programmable number of CPU clocks.

2.5 Interrupt Controller

The 80186 includes an interrupt controller. This controller arbitrates interrupt requests between all internal and external sources. It can be directly cascaded as the master to two external 8259A interrupt controllers. In addition, it can be configured as a slave controller.

2.6 Clock Generator

The 80186 includes a clock generator and crystal oscillator. The crystal oscillator can be used with a parallel resonant, fundamental mode crystal at 2X the desired CPU clock speed (i.e., 16 MHz for an 8 MHz 80186), or with an external oscillator also at 2X the CPU clock. The output of the oscillator is internally divided by two to provide the 50% duty cycle CPU clock from which all 80186 system timing derives. The CPU clock is externally available, and all timing parameters are referenced to this externally available signal. The clock generator also provides ready synchronization for the processor.

2.7 Chip Select and Ready Generation Unit

The 80186 includes integrated chip select logic which can be used to enable memory or peripheral devices. Six output lines are used for memory addressing and seven output lines are used for peripheral addressing.

The memory chip select lines are split into 3 groups for separately addressing the major memory areas in a typical 80186 system: upper memory for reset ROM, lower memory for interrupt vectors, and mid-range memory for program memory. The size of each of these regions is user programmable. The starting location and ending location of lower memory and upper memory are fixed at 00000H and FFFFFH respectively; the starting location of the mid-range memory is user programmable.

Each of the seven peripheral select lines address one of seven contiguous 128 byte blocks above a programmable base address. This base address can be located in either memory or I/O space in order that peripheral devices may be I/O or memory mapped.

Each of the programmed chip select areas has associated with it a set of programmable ready bits. These ready bits control an integrated wait state generator. This allows a programmable number of wait states (0 to 3) to be automatically inserted whenever an access is made to the area of memory associated with the chip select area. In addition, each set of ready bits includes a bit which determines whether the external ready signals (ARDY and SRDY) will be used, or whether they will be ignored (i.e., the bus cycle will terminate even though a ready has not been returned on the external pins). There are 5 total sets of ready bits which allow independent ready generation for each of upper memory, lower memory, mid-range memory, peripheral devices 0–3 and peripheral devices 4–6.

2.8 Integrated Peripheral Accessing

The integrated peripheral and chip select circuitry is controlled by sets of 16-bit registers accessed using standard input, output, or memory access instructions. These peripheral control registers are all located within a 256 byte block which can be placed in either memory or I/O space. Because they are accessed exactly as if they were external devices, no new instruction types are required to access and control the integrated peripherals. For more information concerning the interfacing and accessing of the integrated 80186 peripherals not included in this note, please consult the 80186 data sheet, or the 8086/80186 User's Manual Hardware Reference.

3.0 USING THE 80186

3.1 Bus Interfacing to the 80186

3.1.1 OVERVIEW

The 80186 bus structure is very similar to the 8086 bus structure. It includes a multiplexed address/data bus, along with various control and status lines (see Table 1). Each bus cycle requires a minimum of 4 CPU clock cycles along with any number of wait states required to accommodate the speed access limitations of external memory or peripheral devices. The bus cycles initiated by the 80186 CPU are identical to the bus cycles intitiated by the 80186 integrated DMA unit.

Each clock cycle of the 80186 bus cycle is called a "T" state, and are numbered sequentially T_1, T_2, T_3, T_W and T_4. Additional idle T states (T_i) can occur between T_4 and T_1 when the processor requires no bus activity (instruction fetches, memory writes, I/O reads, etc.). The ready signals control the number of wait states (t_W) inserted in each bus cycle. The maximum number of wait states is unbounded.

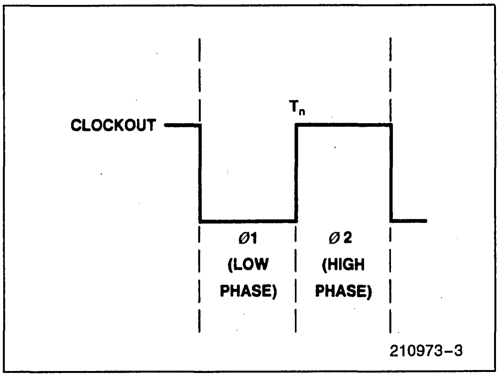


Figure 3. T-state in the 80186

The beginning of a T state is signaled by a high to low transition of the CPU clock. Each T state is divided into two phases, phase 1 (or the low phase) and phase 2 (or the high phase) which occur during the low and high levels of the CPU clock respectively (see Figure 3).

Different types of bus activity occur for all of the T-states (see Figure 4). Address generation information occurs during T_1, data generation during T_2, T_3, T_W-

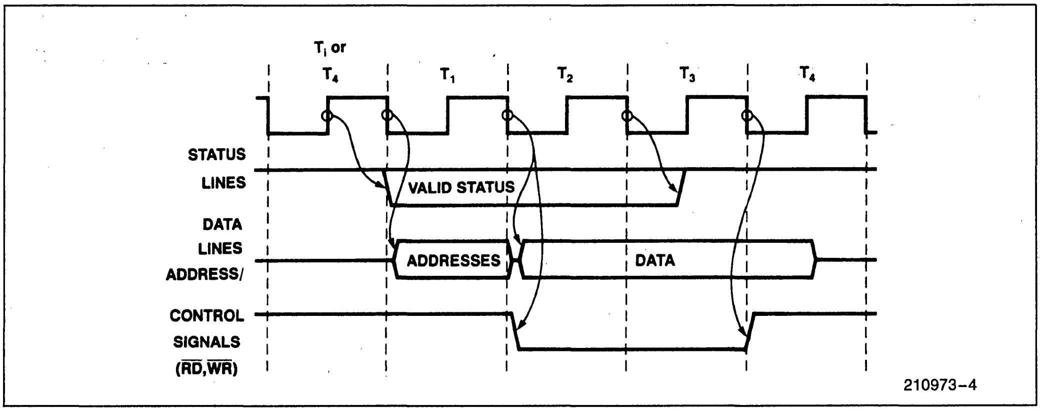


Figure 4. Example Bus Cycle of the 80186

Table 1. 80186 Bus Signals

Function	Signal Name
address/data	AD0–AD15
address/status	A16/S3-A19-S6, $\overline{\text{BHE}}$/S7
co-processor control	$\overline{\text{TEST}}$
local bus arbitration	HOLD, HLDA
local bus control	ALE, $\overline{\text{RD}}$, $\overline{\text{WR}}$, DT/$\overline{\text{R}}$, $\overline{\text{DEN}}$
multi-master bus	$\overline{\text{LOCK}}$
ready (wait) interface	SRDY, ARDY
status information	$\overline{\text{S0}}$–$\overline{\text{S2}}$

and T_4. The beginning of a bus cycle is signaled by the status lines of the processor going from a passive state (all high) to an active state in the middle of the T-state immediately before T_1 (either a T_4 or a T_i). Because information concerning an impending bus cycle occurs during the T-state immediately before the first T-state of the cycle itself, two different types of T_4 and T_i can be generated: one where the T state is immediately followed by a bus cycle, and one where the T state is immediately followed by an idle T state.

During the first type of T_4 or T_i, status information concerning the impending bus cycle is generated for the bus cycle immediately to follow. This information will be available no later than t_{CHSV} (55 ns) after the low-to-high transition of the 80186 clock in the middle of the T state. During the second type of T_4 or T_i the status outputs remain inactive (high), since no bus cycle is to be started. This means that the decision per the nature of a T_4 or T_i state (i.e., whether it is immediately followed by a T_i or a T_1) is decided at the beginning of the T-state immediately preceding the T_4 or T_i (see Figure 5). This has consequences for the bus latency time (see section 3.3.2 on bus latency).

3.1.2. PHYSICAL ADDRESS GENERATION

Physical addresses are generated by the 80186 during T_1 of a bus cycle. Since the address and data lines are multiplexed on the same set of pins, addresses must be latched during T_1 if they are required to remain stable for the duration of the bus cycle. To facilitate latching of the physical address, the 80186 generates an active high ALE (Address Latch Enable) signal which can be directly connected to a transparent latch's strobe input.

Figure 6 illustrates the physical address generation parameters of the 80186. Addresses are guaranteed valid no greater than t_{CLAV} (55 ns) after the beginning of T_1, and remain valid at least t_{CLAX} (10 ns) after the end of T_1. The ALE signal is driven high in the middle of the T state (either T_4 or T_i) immediately preceding T_1 and is driven low in the middle of T_1, no sooner than t_{AVLL} (30 ns) after addresses become valid. This parameter (t_{AVLL}) is required to satisfy the address latch set-up times of address valid until strobe inactive. Addresses remain stable on the address/data bus at least t_{LLAX} (30 ns) after ALE goes inactive to satisfy address latch hold times of strobe inactive to address invalid.

Because ALE goes high long before addresses become valid, the delay through the address latches will be chiefly the propagation delay through the latch rather than the delay from the latch strobe, which is typically longer than the propagation delay. For the Intel 8282 latch, this parameter is t_{IVOV}, the input valid to output valid delay when strobe is held active (high). Note that the 80186 drives ALE high one full clock phase earlier than the 8086 or the 8288 bus controller, and keeps it high throughout the 8086 or 8288 ALE high time (i.e., the 80186 ALE pulse is wider).

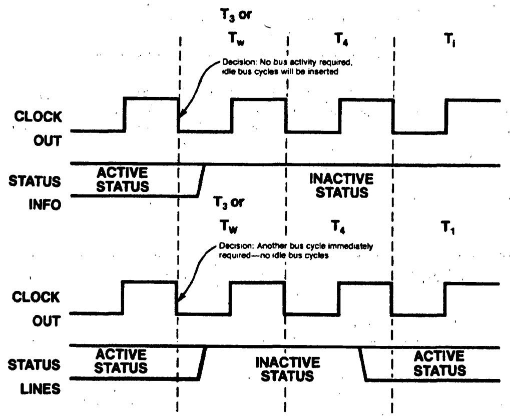
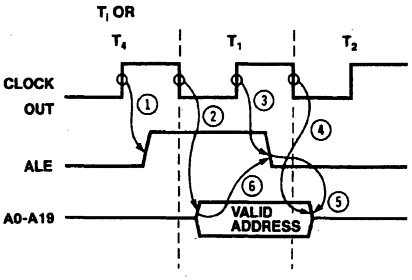


Figure 5. Active-Inactive Status Transitions in the 80186



210973-6

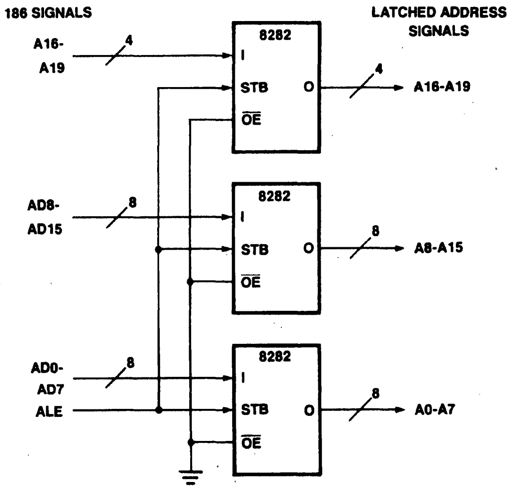
NOTES:

1. t_{CHLH}: Clock high to ALE high = 35 ns max
2. t_{CLAV}: Clock low to address valid = 55 ns max
3. t_{CHLL}: Clock high to ALE low = 35 ns max
4. t_{CLAX}: Clock low to address invalid (address hold from clock low) = 10 ns min
5. t_{LLAX}: ALE low to address invalid (address hold from ALE) = 30 ns min
6. t_{AVLL}: Address valid to ALE low (address setup to ALE) = 30 ns min

Figure 6. Address Generation Timing of the 80186

A typical circuit for latching physical addresses is shown in Figure 7. This circuit uses 3 8282 transparent octal non-inverting latches to demultiplex all 20 address bits provided by the 81086. Typically, the upper 4 address bits are used only to select among various memory components or subsystems, so when the integrated chip selects (see section 8) are used, these upper bits need not be latched. The worst case address generation time from the beginning of T_1 (including address latch propagation time (t_{IVOV}) of the Intel 8282) for the circuit is:

$$t_{CLAV}\ (44\ \text{ns}) + t_{IVOV}\ (30\ \text{ns}) = 74\ \text{ns}$$



210973-7

Figure 7. Demultiplexing the Address Bus of the 80186

Many memory or peripheral devices may not require addresses to remain stable throughout a data transfer. If a system is constructed wholly with these types of devices, addresses need not be latched. In addition, two of the peripheral chip select outputs of the 80186 may be configured to provide latched A1 and A2 outputs for peripheral register selects in a system which does not demultiplex the address/data bus.

One more signal is generated by the 80186 to address memory: $\overline{\text{BHE}}$ (Bus High Enable). This signal, along with A0, is used to enable byte devices connected to either or both halves (bytes) of the 16-bit data bus (see section 3.1.3 on data bus operation section). Because A0 is used only to enable devices onto the lower half of the data bus, memory chip address inputs are usually driven by address bits A1–A19, NOT A0–A19. This provides 512K unique *word* addresses, or 1M unique BYTE addresses.

Of course, $\overline{\text{BHE}}$ is not present on the 8 bit 80188. All data transfers occur on the 8 bits of the data bus.

3.1.3 80186 DATA BUS OPERATION

Throughout T_2, T_3, T_W and T_4 of a bus cycle the multiplexed address/data bus becomes a 16-bit data bus. Data transfers on this bus may be either in bytes or in words. All memory is byte addressable, that is, the upper and lower byte of a 16-bit word each have a unique byte address by which they may be individually accessed, even though they share a common word address (see Figure 8).

All bytes with even addresses (A0 = 0) reside on the lower 8 bits of the data bus, while all bytes with odd addresses (A0 = 1) reside on the upper 8 bits of the data bus. Whenever an access is made to only the even byte, A0 is driven low, $\overline{\text{BHE}}$ is driven high, and the data transfer occurs on D0–D7 of the data bus. Whenever an access is made to only the odd byte, $\overline{\text{BHE}}$ is driven low, A0 is driven high, and the data transfer occurs on D8–D15 of the data bus. Finally, if a word access is performed to an even address, both A0 and $\overline{\text{BHE}}$ are driven low and the data transfer occurs on D0–D15.

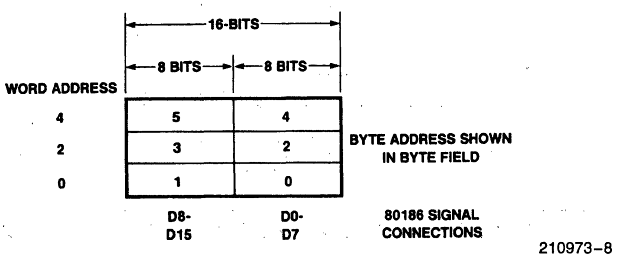
Word accesses are made to the addressed byte and to the next higher numbered byte. If a word access is performed to an odd address, two byte accesses must be performed, the first to access the odd byte at the first word address on D8–D15, the second to access the even byte at the next sequential word address on D0–D7. For example, in Figure 8, byte 0 and byte 1 can be individually accessed (read or written) in two separate bus cycles (byte accesses) to byte addresses 0 and 1 at word address 0. They may also be accessed together in a single bus cycle (word access) to word address 0. However, if a word access is made to address 1, two bus cycles will be required, the first to access byte 1 at word address 0 (note byte 0 will not be accessed), and the second to access byte 2 at word address 2 (note byte 3 will not be accessed). This is why all word data should be located at even addresses to maximize processor performance.

When byte reads are made, the data returned on the half of the data bus not being accessed is ignored. When byte writes are made, the data driven on the half of the data bus not being written is indeterminate.

3.1.4 80188 DATA BUS OPERATION

Because the 80188 externally has only an 8-bit data bus, the above discussion about upper and lower bytes of the data bus does not apply to the 80188. No performance improvement will occur if word data is placed on even boundaries in memory space. All word accesses require two bus cycles, the first to access to lower byte of the word; the second to access the upper byte of the word.

Any 80188 access to the integrated peripherals must be done 16 bits at a time: thus in this special case, a word access will occur in a single bus cycle in the 80188. The



WORD ADDRESS		
4	5	4
2	3	2
0	1	0
	D8-D15	D0-D7

Figure 8. Physical Memory Byte/Word Addressing in the 80186

external data bus will record only a single byte being transferred, however.

3.1.5 GENERAL DATA BUS OPERATION

Because of the bus drive capabilities of the 80186 (200 pF, sinking 2 mA, sourcing 400 μA, roughly twice that of the 8086), this bus may not require additional buffering in many small systems. If data buffers are not used in the system, care should be taken not to allow bus contention between the 80186 and the devices directly connected to the 80186 data bus. Since the 80186 floats the address/data bus before activating any command lines, the only requirement on a directly connected device is that it floats its output drivers after a read *BEFORE* the 80186 beings to drive address information for the next bus cycle. The parameter of interest here is the minimum time from $\overline{\text{RD}}$ inactive until addresses active for the next bus cycle (t_{RHAV}) which has a minimum value of 85 ns. If the memory or peripheral device cannot disable its output drivers in this time, data buffers will be required to prevent both the 80186 and the peripheral or memory device from driving these lines concurrently. Note, this parameter is unaffected by the addition of wait states. Data buffers solve this problem because their output float times are typically much faster than the 80186 required minimum.

If the buffers are required, the 80186 provides $\overline{\text{DEN}}$ (Data ENable) and DT/$\overline{\text{R}}$ (Data Transmit/Receive) signals to simplify buffer interfacing. The $\overline{\text{DEN}}$ and DT/$\overline{\text{R}}$ signals are activated during all bus cycles, whether or not the cycle addresses buffered devices. The $\overline{\text{DEN}}$ signal is driven low whenever the processor is either ready to receive data (during a read) or when the processor is ready to send data (during a write) (that is, any time during an active bus cycle when address information is not being generated on the address/data pins). In most systems, the $\overline{\text{DEN}}$ signal should NOT be directly connected to the $\overline{\text{OE}}$ input of buffers, since unbuffered devices (or other buffers) may be directly connected to the processor's address/data pins. If $\overline{\text{DEN}}$ were directly connected to several buffers, contention would occur during read cycles, as many devices attempt to drive the processor bus. Rather, it should be a factor (along with the chip selects for buffered devices) in generating the output enable input of a bi-directional buffer.

The DT/$\overline{\text{R}}$ signal determines the direction of data propagation through the bi-directional bus buffers. It is high whenever data is being driven out from the processor, and is low whenever data is being read into the processor. Unlike the $\overline{\text{DEN}}$ signal, it may be directly connected to bus buffers, since this signal does not usually directly enable the output drivers of the buffer. An example data bus subsystem supporting both buffered and unbuffered devices is shown in Figure 9. Note that the A side of the 8286 buffer is connected to the 80186, the B side to the external device. The B side of the buffer has greater drive capacity than the A side (since it is meant to drive much greater loads). The DT/$\overline{\text{R}}$ signal can directly drive the T (transmit) signal of the buffer, since it has the correct polarity for this configuration.

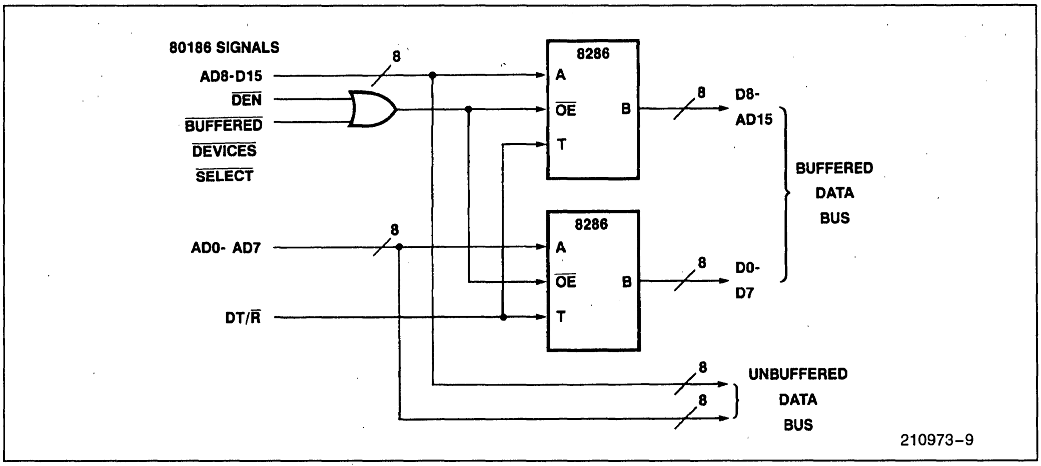
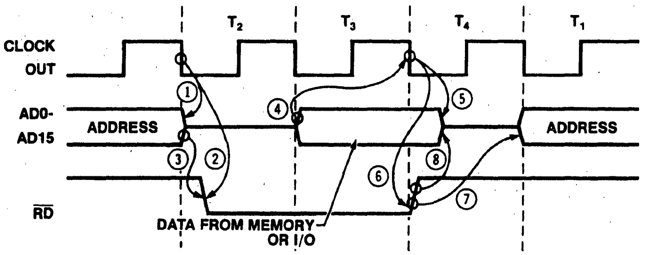


Figure 9. Example 80186 Buffered/Unbuffered Data Bus



210973-10

NOTES:

1. t_{CLAZ}: Clock low until address float = 35 ns max
2. t_{CLRL}: Clock low until $\overline{RD}$ active = 70 ns max
3. t_{AZRL}: Address float until $\overline{RD}$ active = 0 ns min
4. t_{DVCL}: Data valid until clock low (data input set-up time) = 20 ns min*
5. t_{CLDX}: Clock low unitl data invalid (data input hold time from clock) = 10 ns min*
6. t_{CLRH}: Clock low until $\overline{RD}$ high = 10 ns min
7. t_{RHAV}: $\overline{RD}$ high until addresses valid = 85 ns min
8. t_{RHDX}: Read high until data invalid (data input hold from $\overline{RD}$) = 0 ns min*

*Input requirements of 80186, all others are output characteristics

Figure 10. Read Cycle Timing of the 80186

3.1.6 CONTROL SIGNALS

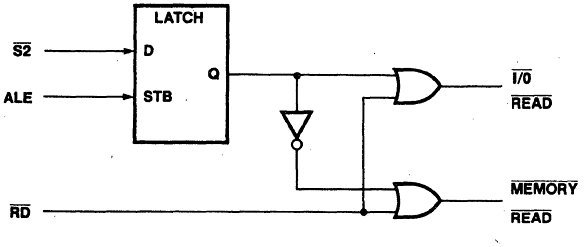
The 80186 directly provides the control signals $\overline{RD}$, $\overline{WR}$, $\overline{LOCK}$ and $\overline{TEST}$. In addition, the 80186 provides the status signals S0–S2 and S6 from which all other required bus control signals can be generated.

3.1.6.1 $\overline{RD}$ and $\overline{WR}$

The $\overline{RD}$ and $\overline{WR}$ signals strobe data to or from memory or I/O space. The $\overline{RD}$ signal is driven low off the beginning of T_2, and is driven high off the beginning of T_4 during all memory and I/O reads (see Figure 10). $\overline{RD}$ will not become active until the 80186 has ceased driving address information on the address/data bus. Data is sampled into the processor at the beginning of T_4. $\overline{RD}$ will not go inactive until the processor's data hold time (10 ns) has been satisfied.

Note that the 80186 does not provide separate I/O and memory $\overline{RD}$ signals. If separate I/O read and memory read signals are required, they can be synthesized using the $\overline{S2}$ signal (which is low for all I/O operations and high for all memory operations) and the $\overline{RD}$ signal (see Figure 11). It should be noted that if this approach is used, the $\overline{S2}$ signal will require latching, since the $\overline{S2}$ signal (like $\overline{S0}$ and $\overline{S1}$) goes to a passive state well before the beginning of T_4 (where $\overline{RD}$ goes inactive). If $\overline{S2}$ was directly used for this purpose, the type of read command (I/O or memory) could change just before T_4 as $\overline{S2}$ goes to the passive state (high). The status signals may be latched using ALE in an identical fashion as is used to latch the address signals (often using the spare bits in the address latches).

Often the lack of a separate I/O and memory $\overline{RD}$ signal is not important in an 80186 system. Each of the



210973-11

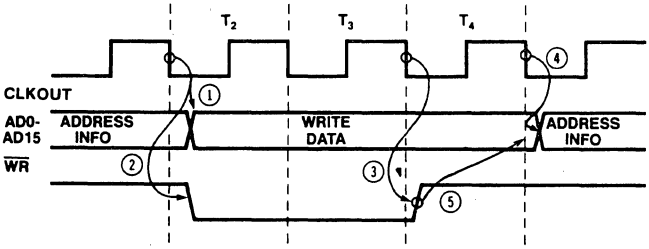
Figure 11. Generating I/O and Memory Read Signals from the 80186

80186 chip select signals will respond on only one of memory or I/O accesses (the memory chip selects respond only to accesses memory space; the peripheral chip selects can respond to accesses in either I/O or memory space, at programmer option). Thus, the chip select signal enables the external device only during accesses to the proper address in the proper space.

The WR signal is also driven low off the beginning of T_2 and driven high off the beginning of T_4 (see Figure 12). Like the RD signal, the WR signal is active for all memory and I/O writes, and also like the RD signal, separate I/O and memory writes may be generated using the latched S2 signal along with the WR signal. More importantly, however, is the active going edge of write. At the time WR makes its active (high to low) transition, valid write data is NOT present on the data bus. This has consequences when using this signal as a write enable signal for DRAMs and iRAMs since both of these devices require that the write data be stable on the data bus at the time of the inactive to active transition of the WE signal. In DRAM applications, this problem is solved by a DRAM controller (such as the Intel 8207 or 8203), while with iRAMs this problem may be solved by placing cross-coupled NAND gates between the CPU and the iRAMS on the WR line (see Figure 13). This will delay the active going edge of the WR signal to the iRAMs by a clock phase, allowing valid data to be driven onto the data bus.

3.1.6.2 Queue Status Signals

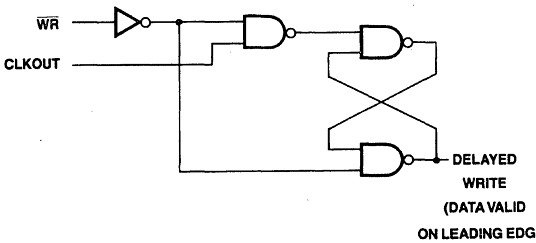
If the RD line is externally grounded during reset and remains grounded during processor operation, the 80186 will enter "queue status" mode. When in this mode, the WR and ALE signals become queue status outputs, reflecting the status of the internal prefetch queue during each clock cycle. These signals are provided to allow a processor extension (such as the Intel 8087 floating point processor) to track execution of instructions within the 80186. The interpretation of QS0 (ALE) and QS1 (WR) are given in Table 2. These signals change on the high-to-low clock transition, one clock phase earlier than on the 8086. Note that since execution unit operation is independent of bus interface unit operation, queue status lines may change in any T state.



210973-12

NOTES:
1. t_{CLDV}: Clock low until data valid = 44 ns max
2. t_{CVCTV}: Clock low until WR active = 50 ns max
3. t_{CVCTX}: Clock low until WR inactive = 55 ns max
4. t_{CLDOX}: Clock high until data invalid = 10 ns min
5. t_{WHDX}: WR inactive until data invalid = $t_{CLCL} - 40$ = 85 ns min

Figure 12. Write Cycle Timing of the 80186



210973-13

Figure 13. Synthesizing Delayed Write from the 80186

Table 2. 80186 Queue Status

QS1	QS0	Interpretation
0	0	no operation
0	1	first byte of instruction taken from queue
1	0	queue was reinitialized
1	1	subsequent byte of instruction taken from queue

Since the ALE, $\overline{RD}$, and $\overline{WR}$ signals are not directly available from the 80186 when it is configured in queue status mode, these signals must be derived from the status lines $\overline{S0}$–$\overline{S2}$ using an external 8288 bus controller (see below). To prevent the 80186 from accidentally entering queue status mode during reset, the $\overline{RD}$ line is internally provided with a weak pullup device. $\overline{RD}$ is the ONLY three-state or input pin on the 80186 which is supplied with a pullup or pulldown device.

3.1.6.3 Status Lines

The 80186 provides 3 status outputs which are used to indicate the type of bus cycle currently being executed. These signals go from an inactive state (all high) to one of seven possible active states during the T state immediately preceding T_1 of a bus cycle (see Figure 5). The possible status line encodings and their interpretations are given in Table 3. The status lines are driven to their inactive state in the T state (T_3 or T_W) immediately preceding T_4 of the current bus cycle.

The status lines may be directly connected to an 8288 bus controller, which can be used to provide local bus control signals or multi-bus control signals (see Figure 14). Use of the 8288 bus controller does not preclude the use of the 80186 generated $\overline{RD}$, $\overline{WR}$ and ALE signals, however. The 80186 directly generated signals, may be used to provide local bus control signals, while an 8288 is used to provide multi-bus control signals, for example.

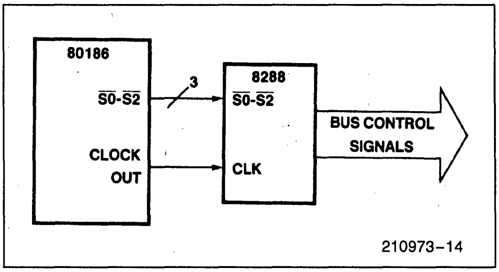

80186
$\overline{S0}$-$\overline{S2}$
CLOCK OUT
3
8288
$\overline{S0}$-$\overline{S2}$
CLK
BUS CONTROL SIGNALS
210973-14

Figure 14. 80186/8288 Bus Controller Interconnection

Table 3. 80186 Status Line Interpretation

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Operation
0	0	0	interrupt acknowledge
0	0	1	read I/O
0	1	0	write I/O
0	1	1	halt
1	0	0	instruction fetch
1	0	1	read memory
1	1	0	write memory
1	1	1	passive

The 80186 provides two additional status signals: S6 and S7. S7 is equivalent to $\overline{BHE}$ (see section 3.1.2) and appears on the same pin as $\overline{BHE}$. $\overline{BHE}$/S7 changes state at the beginning of the T_1 state in the bus cycle. $\overline{BHE}$/S7 does not need to be latched, i.e., it may be used directly as the $\overline{BHE}$ signal. S6 provides information concerning the unit generating the bus cycle. It is time multiplexed with A19, and is available during T_2, T_3, T_4 and T_W. In the 8086 family, all central processors (e.g., the 8086, 8088 and 8087) drive this line low, while all I/O processors (e.g., 8089) drive this line high during their respective bus cycles. Following this scheme, the 80186 drives this line low whenever the bus cycle is generated by the 80186 CPU, but drives it high when the bus cycle is generated by the integrated 80186 DMA unit. This allows external devices to distinguish between bus cycles fetching data for the CPU from those transfering data for the DMA unit.

Three other status signals are available on the 8086 but not on the 80186. They are S3, S4, and S5. Taken together, S3 and S4 indicate the segment register from which the current physical address drives. S5 indicates the state of the interrupt flip-flop. On the 80186, these signals will ALWAYS be low.

3.1.6.4 $\overline{TEST}$ and $\overline{LOCK}$

Finally, the 80186 provides a $\overline{TEST}$ input and a $\overline{LOCK}$ output. The $\overline{TEST}$ input is used in conjunction with the processor WAIT instruction. It is typically driven by a processor extension (like the 8087) to indicate whether it is busy. Then, by executing the WAIT (or FWAIT) instruction, the central processor may be forced to temporarily suspend program execution until the processor extension indicates that it is idle by driving the $\overline{TEST}$ line low.

The $\overline{LOCK}$ output is driven low whenever the data cycles of a LOCKED instruction are executed. A LOCKED instruction is generated whenever the LOCK prefix occurs immediately before an instruction.

The LOCK prefix is active for the single instruction immediately following the LOCK prefix. This signal is used to indicate to a bus arbiter (e.g., the 8289) that a series of locked data transfers is occurring. The bus arbiter should under no circumstances release the bus while locked transfers are occurring. The 80186 will not recognize a bus HOLD, nor will it allow DMA cycles to be run by the integrated DMA controller during locked data transfers. LOCKED transfers are used in multiprocessor systems to access memory based semaphore variables which control access to shared system resources.

On the 80186, the $\overline{\text{LOCK}}$ signal will go active during T_1 of the first DATA cycle of the locked transfer. It is driven inactive 3 T-states after the beginning of the last DATA cycle of the locked transfers. On the 8086, the $\overline{\text{LOCK}}$ signal is activated immediately after the LOCK prefix is executed. The LOCK prefix may be executed well before the processor is prepared to perform the locked data transfer. This has the unfortunate consequence of activating the $\overline{\text{LOCK}}$ signal before the first LOCKED data cycle is performed. Since $\overline{\text{LOCK}}$ is active before the processor requires the bus for the data transfer, opcode pre-fetching can be LOCKED. However, since the 80186 does not activate the $\overline{\text{LOCK}}$ signal until the processor is ready to actually perform the locked transfer, locked pre-fetching will not occur with the 80186.

The LOCK output is also driven low by hardware during interrupt acknowledge cycles when the integrated interrupt controller operates in cascaded or iRMX 86 modes (see sections 6.5.2 and 6.5.3). In these modes, the operation of the LOCK pin may be altered when an interrupt occurs during execution of a software-LOCKED instruction. See section 6.5.4 for a description of additional hardware necessary to block DMA and HOLD requests under such circumstances.

3.1.7 HALT TIMING

A HALT bus cycle is used to signal the world that the 80186 CPU has executed a HLT instruction. It differs from a normal bus cycle in two important ways.

The first way in which a HALT bus cycle differs from a normal bus cycle is that since the processor is entering a halted state, none of the control lines ($\overline{\text{RD}}$ or $\overline{\text{WR}}$) will be driven active. Address and data information will not be driven by the processor, and no data will be returned. The second way a HALT bus cycle differs from a normal bus cycle is that the $\overline{\text{S0}}$–$\overline{\text{S2}}$ status lines go to their passive state (all high) during T_2 of the bus cycle, well before they go to their passive state during a normal bus cycle.

Like a normal bus cycle, however, ALE is driven active. Since no valid address information is present, the information strobed into the address latches should be ignored. This ALE pulse can be used, however, to latch the HALT status from the $\overline{\text{S0}}$–$\overline{\text{S2}}$ status lines.

The processor being halted does not interfere with the operation of any of the 80186 integrated peripheral units. This means that if a DMA transfer is pending while the processor is halted, the bus cycles associated with the DMA transfer will run. In fact, DMA latency time will improve while the processor is halted because the DMA unit will not be contending with the processor for access to the 80186 (see section 4.4.1).

3.1.8 8288 AND 8289 INTERFACING

The 8288 and 8289 are the bus controller and multimaster bus arbitration devices used with the 8086 and 8088. Because the 80186 bus is similar to the 8086 bus, they can be directly used with the 80186. Figure 15 shows an 80186 interconnection to these two devices.

The 8288 bus controller generates control signals ($\overline{\text{RD}}$, $\overline{\text{WR}}$, ALE, DT/$\overline{\text{R}}$, $\overline{\text{DEN}}$, etc.) for an 8086 maximum mode system. It derives its information by decoding status lines $\overline{\text{S0}}$–$\overline{\text{S2}}$ of the processor. Because the 80186 and the 8086 drive the same status information on these

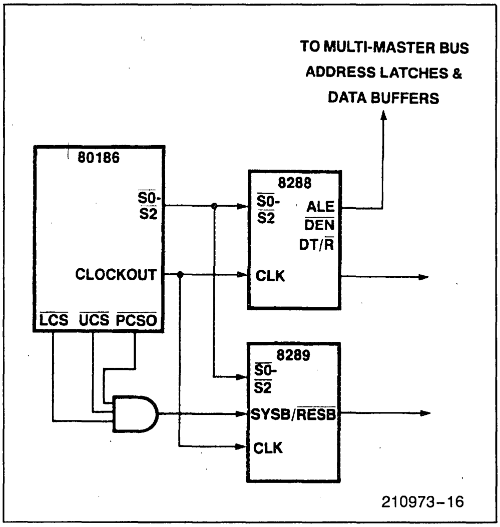
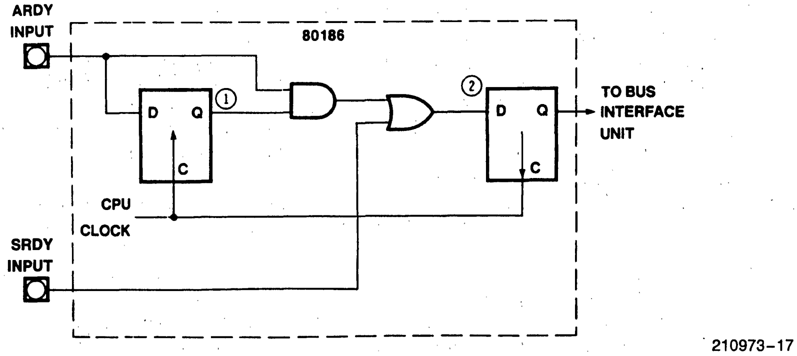


Figure 15. 80186/8288/8289 Interconnection



NOTES:
1. Asynchronous Resolution Flip Flop
2. Ready Latch Flip Flop

The illustrated logic devices are shown for conceptual purposes only. The MOS latches and switches in the actual circuit are not necessarily organized in this manner.

Figure 16. Ready Circuitry of the 80186

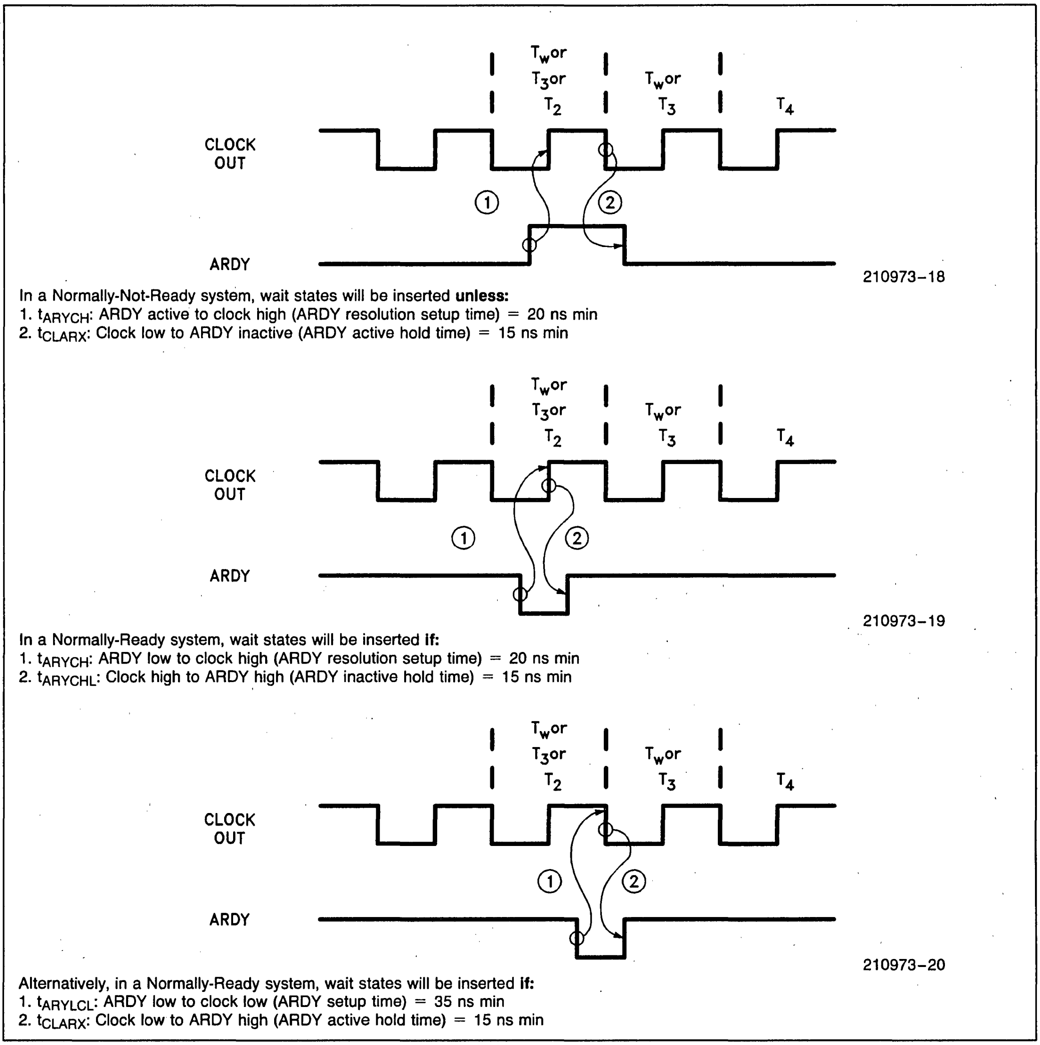
lines, the 80186 can be directly connected to the 8288 just as in an 8086 system. Using the 8288 with the 80186 does not prevent using the 80186 control signals directly. Many systems require both local bus control signals and system bus control signals. In this type of system, the 80186 lines could be used as the local signals, with the 8288 lines used as the system signals. Note that in an 80186 system, the 8288 generated ALE pulse occurs later than that of the 80186 itself. In many multimaster bus systems, the 8288 ALE pulse should be used to strobe the addresses into the system bus address latches to insure that the address hold times are met.

The 8289 bus arbiter arbitrates the use of a multi-master system bus among various devices each of which can become the bus master. This component also decodes status lines $\overline{S0}$–$\overline{S2}$ of the processor directly to determine when the system bus is required. When the system bus is required, the 8289 forces the processor to wait until it has acquired control of the bus, then it allows the processor to drive address, data and control information onto the system bus. The system determines when it requires system bus resources by an address decode. Whenever the address being driven coincides with the address of an on-board resource, the system bus is not required and thus will not be requested. The circuit shown factors the 80186 chip select lines to determine when the system bus should be requested, or when the 80186 request can be satisfied using a local resource.

3.1.9 READY INTERFACING

The 80186 provides two ready lines, a synchronous ready (SRDY) line and an asynchronous ready (ARDY) line. These lines signal the processor to insert wait states (T_W) into a CPU bus cycle. This allows slower devices to respond to CPU service requests (reads or writes). Wait states will only be inserted when both ARDY and SRDY are low, i.e., only one of the lines need be active to terminate a bus cycle. Figure 16 depicts the logical ORing of the ARDY and SRDY functions. Any number of wait states may be inserted into a bus cycle. The 80186 will ignore the RDY inputs during any accesses to the integrated peripheral registers and to any area where the chip select ready bits indicate that the external ready should be ignored.

The timing required by the two RDY lines is different. The ARDY line is meant to be used with asynchronous ready inputs. Thus, inputs to this line will be internally synchronized to the CPU clock before being presented to the processor. The synchronization circuitry used with the ARDY line is shown in Figure 16. The first flip-flop is used to "resolve" the asynchronous transition of the ARDY line. It will achieve a definite level (either high or low) before its output is latched into the second flip-flop for presentation to the CPU. When latched high, it allows the level present on the ARDY line to pass directly to the CPU; when latched low, it forces not ready to be presented to the CPU (see Appendix B for synchronizer information).



In a Normally-Not-Ready system, wait states will be inserted **unless:**
1. t_{ARYCH}: ARDY active to clock high (ARDY resolution setup time) = 20 ns min
2. t_{CLARX}: Clock low to ARDY inactive (ARDY active hold time) = 15 ns min

In a Normally-Ready system, wait states will be inserted **if:**
1. t_{ARYCH}: ARDY low to clock high (ARDY resolution setup time) = 20 ns min
2. t_{ARYCHL}: Clock high to ARDY high (ARDY inactive hold time) = 15 ns min

Alternatively, in a Normally-Ready system, wait states will be inserted **if:**
1. t_{ARYLCL}: ARDY low to clock low (ARDY setup time) = 35 ns min
2. t_{CLARX}: Clock low to ARDY high (ARDY active hold time) = 15 ns min

Figure 17. ARDY Transitions

Asynchronous ready logic may be implemented as either Normally-Ready or Normally-Not-Ready. Figure 17 depicts activity for both implementations. Remember that for ARDY to force wait states, SRDY must be low as well.

In a Normally-Not-Ready implementation the setup and hold times of **both** the resolution flip-flop **and** the ready latch must be satisfied. The ARDY pin must go active at least 20 ns before the rising edge of T_2, T_3, or T_W and stay active until 15 ns after the falling edge of T_3 or T_W to stop generation of wait states and terminate the bus cycle. If ARDY goes active before the rising edge of T_2 and stays active after the falling edge of T_3 there will be no wait state inserted.

In a Normally-Ready implementation the setup and hold times of **either** the resolution flip-flop or the ready latch must be met. Wait states will be generated if ARDY goes inactive 20 ns before the rising edge of T_2 and stays inactive a minimum of 15 ns after the falling edge, **or** if ARDY goes inactive at least 35 ns before the falling edge of T_3 and stays inactive a minimum of 15 ns after the edge. The 80186 ready circuitry performs in this manner in order to allow a slow device the maximum amount of time to respond with a not ready after it has been selected.

The synchronous ready (SRDY) line requires that ALL transitions on this line during T_2, T_3 or T_W satisfy a certain setup and hold time (t_{SRYCL} = 35 ns and t_{CLSRY} = 15 ns respectively). If these requirements are not met, the CPU will not function properly. Valid transitions on this line, and subsequent wait state insertion is shown in Figure 18. The processor looks at this line at the beignning of each T_3 and T_W. If the line is sampled active at the beginning of either of these two cycles, that cycle will be immediately followed by T_4.

On the other hand, if the line is sampled inactive at the beginning of either of these two cycles, that cycle will be followed by a T_W. Any asynchronous transition on the SRDY line not occurring at the beginning of T_3 or T_W, that is, when the processor is not "looking at" the ready lines will not cause CPU malfunction.

3.1.10 BUS PERFORMANCE ISSUES

Bus cycles occur sequentially, but do not necessarily come immediately one after another, that is the bus may remain idle for several T states (T_i) between each bus access initiated by the 80186. This occurs whenever the 80186 internal queue is full and no read/write cycles are being requested by the execution unit or integrated DMA unit. The reader should recall that a separate unit, the bus interface unit, fetches opcodes (including immediate data) from memory, while the execution unit actually executes the pre-fetched instructions. The number of clock cycles required to execute an 80186 instruction vary from 2 clock cycles for a register to register move to 67 clock cycles for an integer divide.

If a program contains many long instructions, program execution will be CPU limited, that is, the instruction queue will be constantly filled. Thus, the execution unit does not need to wait for an instruction to be fetched. If a program contains mainly short instructions or data move instructions, the execution will be bus limited. Here, the execution unit will be required to wait often for an instruction to be fetched before it continues its operation. Programs illustrating this effect and performance degradation of each with the addition of wait states are given in appendix G.

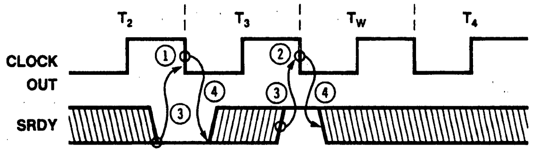
All instruction fetches are word (16-bit) fetches from even addresses unless the fetch occurs as a result of a jump to an odd location. This maximizes the utilization of each bus cycle used for instruction fetching, since each fetch will access two bytes of information. It is also good programming practice to locate all word data at even locations, so that both bytes of the word may be accessed in a single bus cycle (see discussion on data bus operation for further information, section 3.1.3 of this note).

Although the amount of bus utilization, i.e., the percentage of bus time used by the 80186 for instruction fetching and execution required for top performance will vary considerably from one program to another, a typical instruction mix on the 80186 will require greater bus utilization than the 8086. This is caused by the higher performance execution unit requiring instructions from the prefetch queue at a greater rate. This also means that the effect of wait states is more pronounced in an 80186 system than in an 8086 system. In all but a few cases, however, the performance degradation incurred by adding a wait state is less than might be expected because instruction fetching and execution are performed by separate units.

3.2 Example Memory Systems

3.2.1 2764 INTERFACE

With the above knowledge of the 80186 bus, various memory interfaces may be generated. One of the simplest of these is the example EPROM interface shown in Figure 19.



NOTES:
1. Decision: Ready, T-State will be followed by a wait state
2. Decision: Ready, T-State will **not** be followed by a wait state
3. t_{SRYCL}: Synchronous ready stable until clock low (SRDY set-up time) = 35 ns min
4. t_{CLSRY}: Clock low until synchronous ready transition (SRDY hold time) = 15 ns min

Figure 18. Valid Transitions on the 80186

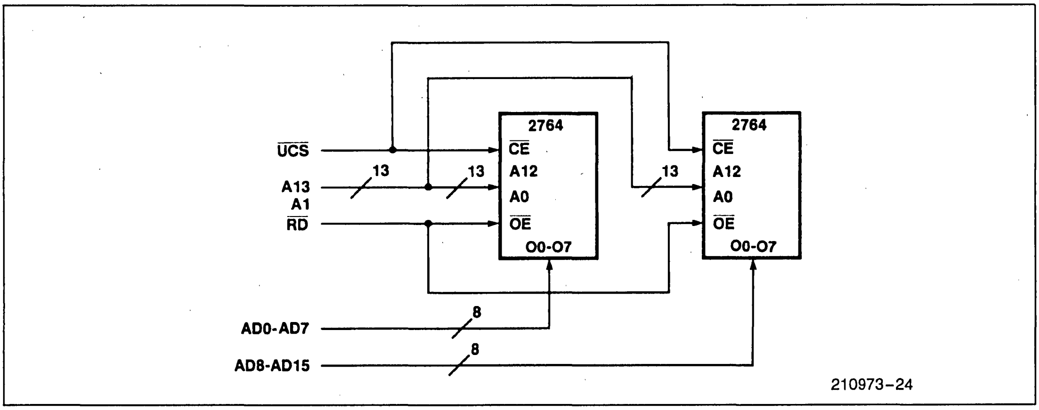


Figure 19. Example 2764/80186 Interface

The addresses are latched using the address generation circuit shown earlier. Note that the A0 line of each EPROM is connected to the A1 address line from the 80186, NOT the A0 line. Remember, A0 only signals a data transfer on the lower 8 bits of the 16-bit data bus! The EPROM outputs are connected directly to the address/data inputs of the 80186, and the 80186 $\overline{RD}$ signal is used as the $\overline{OE}$ for the EPROMs.

The chip enable of the EPROM is driven directly by the chip select output of the 80186 (see section 8). In this configuration, the access time calculation for the EPROMs are:

time from

address: $(3 + N) * t_{CLCL} - t_{CLAV} - t_{IVOV}\ (8282) - t_{DVCL}$

$= 375 + (N * 125) - 44 - 30 - 20$

$= 281 + (N * 125)$ ns

time from

chip select: $(3 + N) * t_{CLCL} - t_{CLCSV} - t_{DVCL}$

$= 375 + (N * 125) - 66 - 20$

$= 289 + (N * 125)$ ns

time from

$\overline{RD}$ ($\overline{OE}$): $(2 + N)\ t_{CLCL} - t_{CLRL} - t_{DVCL}$

$= 250 + (N * 125) - 70 - 20$

$= 160 + (N * 125)$ ns

where:

t_{CLAV} = time from clock low in T_1 until addresses are valid

t_{CLCL} = clock period of processor

t_{IVOV} = time from input valid of 8282 until output valid of 8282

t_{DVCL} = 186 data valid input setup time until clock low time of T_4

t_{CLCSV} = time from clock low in T_1 until chip selects are valid

t_{CLRL} = time from clock low in T_2 until $\overline{RD}$ goes low

N = number of wait states inserted

Thus, for 0 wait state operation, 250 ns EPROMs must be used. The only significant parameter not included above is t_{RHAV}, the time from $\overline{RD}$ inactive (high) until the 80186 begins driving address information. This parameter is 85 ns, which meets the 2764-25 (250 ns speed selection) output float time of 85 ns. If slower EPROMs are used, a discrete data buffer *MUST* be inserted between the EPROM data lines and the address/data bus, since these devices may continue to drive data information on the multiplexed address/data bus when the 80186 begins to drive address information for the next bus cycle.

3.2.2 8203 DRAM INTERFACE

An example 8203/DRAM interface is shown in Figure 20. The 8203 provides all required DRAM control signals, address multiplexing, and refresh generation. In this circuit, the 8203 is configured to interface with 64K DRAMs.

All 8203 cycles are generated off control signals (RD and WR) provided by the 80186. These signals will not go active until T_2 of the bus cycle. In addition, since the 8203 clock (generated by the internal crystal oscillator of the 8203) is asynchronous to the 80186 clock, all memory requests by the 80186 must be synchronized to the 8203 before the cycle will be run. To minimize this synchronization time, the 8203 should be used with the highest speed crystal that will maintain DRAM compatability. Even if a 25 MHz crystal is used (the maximum allowed by the 8203) two wait states will be required by the example circuit when using 150 ns DRAMs with an 8 MHz 80186, three wait states if 200 ns DRAMs are used (see timing analysis, Figure 21).

The entire RAM array controlled by the 8203 can be selected by one or a group of the 80186 provided chip selects. These chip selects can also be used to insert the wait states required by the interface.

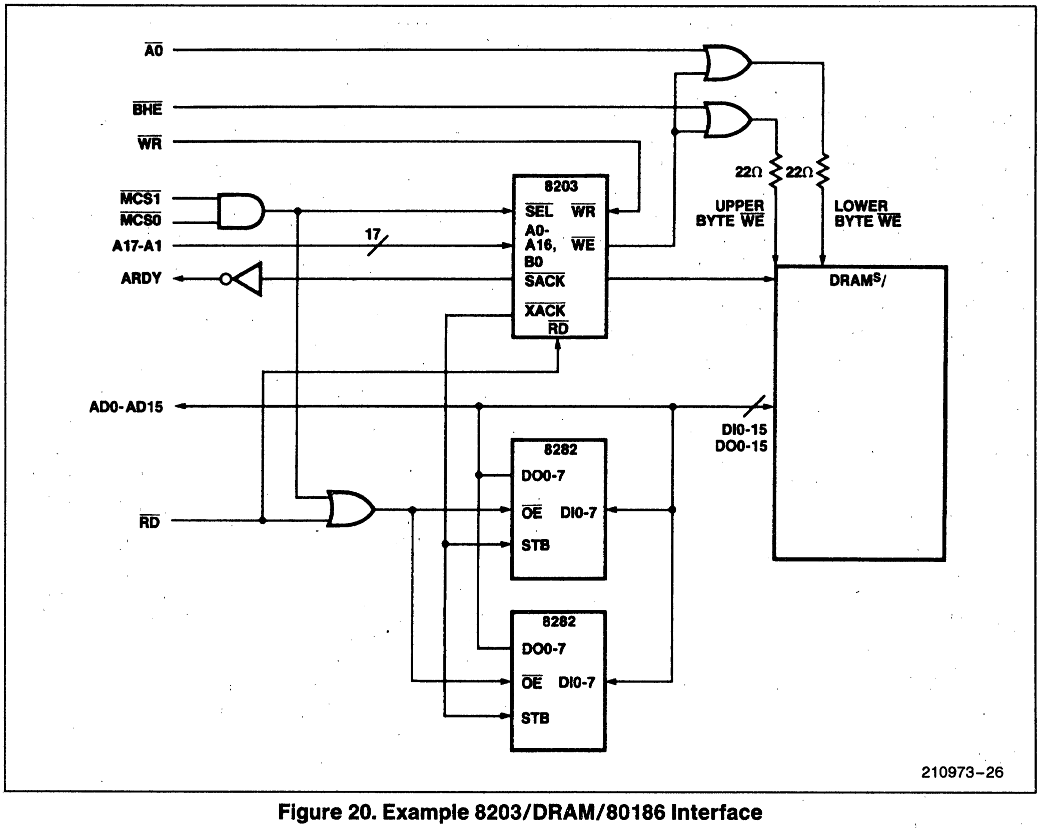
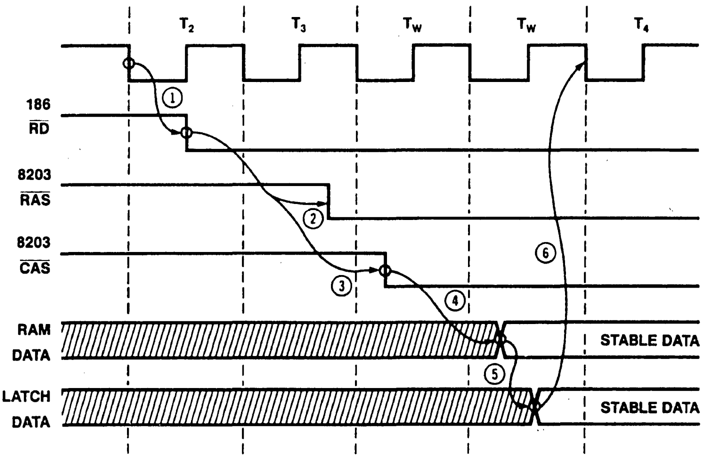


Figure 20. Example 8203/DRAM/80186 Interface



NOTES:
1. t_{CLEL}: Clock low until read low = 70 ns max
2. t_{CR}: Command active until RAS = 150 ns max*
3. t_{CC}: Command active until CAS = 245 ns max*
4. t_{CAC}: Access time from CAS = 85 ns max
5. t_{ISOU}: Input to output delay = 30 ns max
6. t_{DVCL}: Data valid to clock low (data in set up) = 20 ns min
Total Access Time = 70 + 245 + 85 + 30 +20 = 450 ns (3.6 T-states)

① & ② are 186 specs
② & ③ are 8203 specs
④ is a 2164A-15 spec
⑤ is on 8282 spec

*Assumes 25 MHz
8203 operation

Figure 21. Example 8203/2164A-15 Access Time Calculation

Since the 8203 is operating asynchronously to the 80186, the RDY output of the 8203 (used to suspend processor operation when a processor DRAM request coincides with a DRAM refresh cycle) must be synchronized to the 80186. The 80186 ARDY line is used to provide the necessary ready synchronization. The 8203 ready outputs operate in a normally not ready mode, that is, they are only driven active when an 8203 cycle is being executed, and a refresh cycle is not being run. The 8203 $\overline{\text{SACK}}$ is presented to the 80186 only when the DRAM is being accessed. Notice that the $\overline{\text{SACK}}$ output of the 8203 is used, rather than the $\overline{\text{XACK}}$ output. Since the 80186 will insert at least one full CPU clock cycle between the time RDY is sampled active, and the time data must be present on the data bus, using the $\overline{\text{XACK}}$ signal would insert unnecessary additional wait states, since it does not indicate ready until valid data is available from the memory.

3.2.3 8207 DRAM INTERFACE

The 8207 advanced dual-port DRAM controller provides a high performance DRAM memory interface specifically for 80186 and 80286 microcomputer systems. This controller provides all address multiplexing and DRAM refresh circuitry. In addition, it synchronizes and arbitrates memory requests from two different ports (e.g., an 80186 and a Multibus), allowing the two ports to share memory. Finally, the 8207 provides a simple interface to the 8206 error detection and correction chip.

The simplest 8207 (and also the highest performance) interface is shown in Figure 22. This shows the 80186 connected to an 8207 using the 8207 slow cycle, synchronous status interface. In this mode, the 8207 decodes the type of cycle to be run directly from the status lines of the 80186. In addition, since the 8207 CLOCKIN is driven by the CLOCKOUT of the 80186, any performance degradation caused by required memory request synchronization between the 80186 and the 8207 is not present. Finally, the entire memory array driven by the 8207 may be selected using one or a group of the 80186 memory chip selects, as in the 8203 interface above.

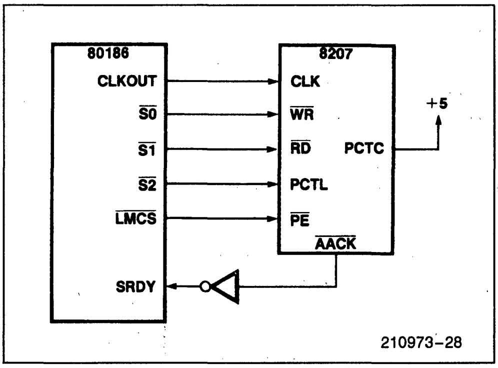


Figure 22. 80186/8207/DRAM Interface

The 8207 $\overline{AACK}$ signal may be used to generate a synchronous ready signal to the 80186 in the above interface. Since dynamic memory periodically requires refreshing, 80186 access cycles may occur simultaneously with an 8207 generated refresh cycle. When this occurs, the 8207 will hold the $\overline{AACK}$ line high until the processor initiated access is run (note, the sense of this line is reversed with respect to the 80186 SRDY input). This signal should be factored with the DRAM (8207) select input and used to drive the SRDY line of the 80186. Remember that only one of SRDY and ARDY needs to be active for a bus cycle to be terminated. If asynchronous devices (e.g., a Multibus interface) are connected to the ARDY line with the 8207 connected to the SRDY line, care must be taken in design of the ready circuit such that only one of the RDY lines is driven active at a time to prevent premature termination of the bus cycle.

A single-port version of the 8207 is available as the 8208. For more information about DRAM interfacing and timing, consult the 8207 and 8208 data sheets.

3.3 HOLD/HLDA Interface

The 80186 employs a HOLD/HLDA bus exchange protocol. This protocol allows other asynchronous bus master devices (i.e., ones which drive address, data, and control information on the bus) to gain control of the bus to perform bus cycles (memory or I/O reads or writes).

3.3.1 HOLD RESPONSE

In the HOLD/HLDA protocol, a device requiring bus control (e.g., an external DMA device) raises the HOLD line. In response to this HOLD request, the 80186 will raise its HLDA line after it has finished its current bus activity. When the external device is finished with the bus, it drops its bus HOLD request. The 80186 responds by dropping its HLDA line and resuming bus operation.

When the 80186 recognizes a bus hold by driving HLDA high, it will float many of its signals (see Figure 23). AD0–AD15 (address/data 0–15) and $\overline{DEN}$ (data enable) are floated within t_{CLAZ} (35 ns) after the same clock edge that HLDA is driven active. A16–A19 (address 16–19) $\overline{RD}$, $\overline{WR}$, $\overline{BHE}$ (Bus High Enable), DT/$\overline{R}$ (Data Transmit/Receive) and $\overline{S0}$–$\overline{S2}$ (status 0–2) are floated within t_{CHCZ} (45 ns) after the clock edge immediately before the clock edge on which HLDA comes active.

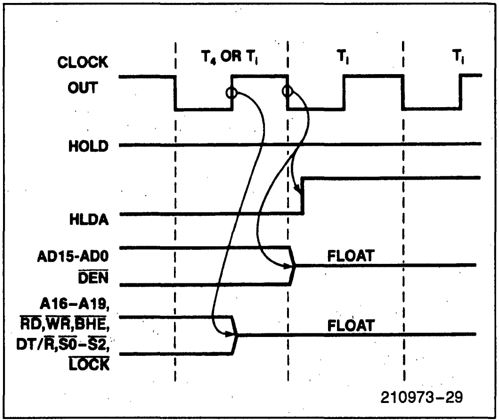


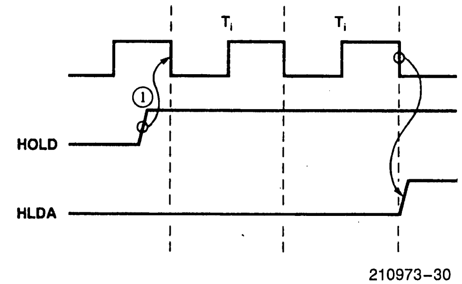
Figure 23. Signal Float/HLDA Timing of the 80186

Only the above mentioned signals are floated during bus HOLD. Of the signals not floated by the 80186, some have to do with peripheral functionality (e.g., TmrOut). Many others either directly or indirectly control bus devices. These signals are ALE (Address Latch Enable, see section 3.1.2) and all the chip select lines ($\overline{UCS}$, $\overline{LCS}$, $\overline{MCS0-3}$, and $\overline{PCS0-6}$). The designer must be aware that the chip select circuitry does not look at externally generated addresses (see section 8 for a discussion of the chip select logic). Thus, for memory or peripheral devices which are addressed by external bus master devices, discrete chip select and ready generation logic must be used.

3.3.2 HOLD/HLDA TIMING AND BUS LATENCY

The time required between HOLD going active and the 80186 driving HLDA active is known as bus latency. Many factors affect this latency, including synchronization delays, bus cycle times, locked transfer times and interrupt acknowledge cycles.

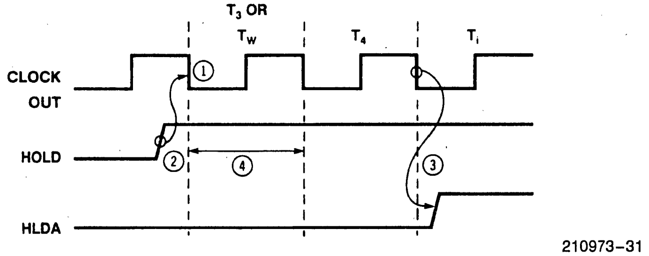
The HOLD request line is internally synchronized by the 80186, and may therefore be an asynchronous signal. To guarantee recognition on a certain clock edge, it must satisfy a certain setup and hold time to the falling edge of the CPU clock. A full CPU clock cycle is required for this synchronization, that is, the internal HOLD signal is not presented to the internal bus arbitration circuitry until one full clock cycle after it is latched from the HOLD input (see Appendix B for a discussion of 80186 synchronizers). If the bus is idle, HLDA will follow HOLD by two CPU clock cycles plus a small amount of setup and propagation delay time. The first clock cycle synchronizes the input; the second signals the internal circuitry to initiate a bus hold. (See Figure 24).



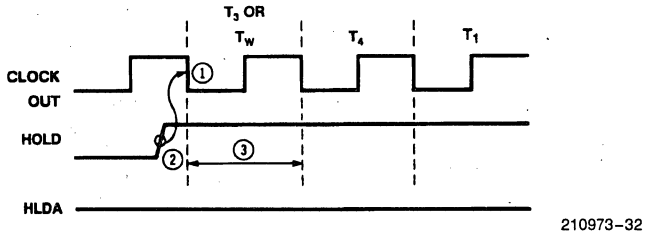
NOTES:
1. t_{HVCL}: Hold valid until clock low = 25 ns min
2. t_{CLHAV}: Clock low until HLDA active = 50 ns max

Figure 24. 80186 Idle Bus Hold/HLDA Timing

Many factors influence the number of clock cycles between a HOLD request and a HLDA. These may make bus latency longer than the best case shown above. Perhaps the most important factor is that the 80186 will not relinquish the local bus until the bus is idle. An idle bus occurs whenever the 80186 is not performing any bus transfers. As stated in section 3.1.1, when the bus is idle, the 80186 generates idle T-states. The bus can become idle only at the end of a bus cycle. Thus, the 80186 can recognize HOLD only after the end of its current bus cycle. The 80186 will normally insert no T_i states between T_4 and T_1 of the next bus cycle if it requires any bus activity (e.g., instruction fetches or I/O reads). However, the 80186 may not have an immediate need for the bus after a bus cycle, and will insert T_i states independent of the HOLD input (see Section 3.1.1).

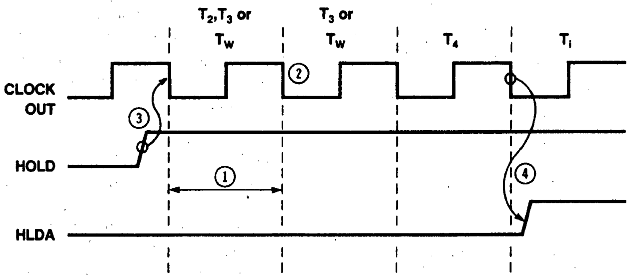


NOTES:
1. Decision: No additional internal bus cycles required, idle T-states will be inserted after T_4
2. Greater than 25 ns (t_{HVCL})
3. Less than 50 ns (t_{CLHAV})
4. HOLD request internally synchronized



NOTES:
1. Decision: Additional internal bus cycles required, no idle T-states will be inserted, HOLD not active soon enough to force idle T-states
2. Greater than 25 ns (t_{HVCL}): not required since it will not get recognized anyway
3. HOLD request internally synchronized

Figure 25. HLD/HLDA Timing in the 80186



NOTES:
1. HOLD request internally synchronized
2. Decision: HOLD request active, idle t-states will be inserted at end of current bus cycle
3. Greater than 25 ns
4. Less than 50 ns

Figure 26. HOLD/HLDA Timing in the 80186

When the HOLD request is active, the 80186 will be forced to proceed from T_4 to T_i in order that the bus may be relinquished. HOLD must go active 3 T-states before the end of a bus cycle to force the 80186 to insert idle T-states after T_4 (one to synchronize the request, and one to signal the 80186 that T_4 of the bus cycle will be followed by idle T-states, see section 3.1.1). After the bus cycle has ended, the bus hold will be immediately acknowledged. If, however, the 80186 has already determined that an idle T-state will follow T_4 of the current bus cycle, HOLD need go active only 2 T-states before the end of a bus cycle to force the 80186 to relinquish the bus at the end of the current bus cycle. This is because the external HOLD request is not required to force the generation of idle T-states. Figure 26 graphically portrays the scenarios depicted above.

An external HOLD has higher priority than both the 80186 CPU or integrated DMA unit. However, an external HOLD will not separate the two cycles needed to perform a word access when the word accessed is located at an odd location (see Section 3.1.3). In addition, an external HOLD will not separate the two-to-four bus cycles required to perform a DMA transfer using the integrated controller. Each of these factors will add additional bus cycle times to the bus latency of the 80186.

Another factor influencing bus latency time is locked transfers. Whenever a locked transfer is occurring, the 80186 will not recognize external HOLDs (nor will it recognize internal DMA bus requests). Locked transfers are programmed by preceding an instruction with the LOCK prefix. Any transfers generated by such a prefixed instruction will be locked, and will not be separated by any external bus requesting device. String instructions may be locked. Since string transfers may require thousands of bus cycles, bus latency time will suffer if they are locked.

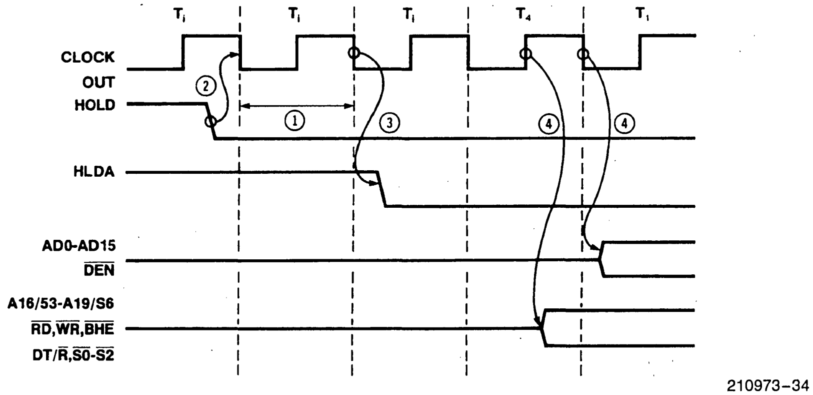
The final factor affecting bus latency time is interrupt acknowledge cycles. When an external interrupt controller is used, or if the integrated interrupt controller is used in Slave mode (see Section 4.4.1) the 80186 will run two interrupt acknowledge cycles back to back. These cycles are automatically "locked" and will never be separated by any bus HOLD, either internal or external. See Section 6.5 on interrupt acknowledge timing for more information concerning interrupt acknowledge timing.

3.3.3 COMING OUT OF HOLD

After the 80186 recognizes that the HOLD input has gone inactive, it will drop its HLDA line in a single clock. Figure 27 shows this timing. The 80186 will insert only two T_i after HLDA has gone inactive, assuming that the 80186 has internal bus cycles to run. During the last T_i, status information will go active concerning the bus cycle about to be run (see Section 3.1.1). If the 80186 has no pending bus activity, it will maintain all lines floating (high impedance) until the last T_i before it begins its first bus cycle after the HOLD.

3.4 Differences between the 8086 Bus and the 80186 Bus

The 80186 bus was defined to be upward compatible with the 8086 bus. As a result, the 8086 bus interface components (the 8288 bus controller and the 8289 bus arbiter) may be used directly with the 80186. There are a few significant differences between the two processors which should be considered.



NOTES:
1. HOLD internally synchronized
2. Greater than 25 ns
3. Less than 50 ns
4. Lines come out of float only if a bus cycle is pending

Figure 27. 80186 Coming out of Hold

CPU Duty Cycle and Clock Generator

The 80186 employs an integrated clock generator which provides a 50% duty cycle CPU clock (1/2 of the time it is high, the other 1/2 of the time it is low). This is different than the 8086, which employs an external clock generator (the 8284A) with a 33% duty cycle CPU clock (1/3 of the time it is high, the other 2/3 of the time, it is low). These differences manifest themselves as follows:

1) No oscillator output is available from the 80186, as it is available from the 8284A clock generator.

2) The 80186 does not provide a PCLK (50% duty cycle, 1/2 CPU clock frequency) output as does the 8284A.

3) The clock low phase of the 80186 is narrower, and the clock high phase is wider than on the same speed 8086.

4) The 80186 does not internally factor AEN with RDY. This means that if both RDY inputs (ARDY and SRDY) are used, external logic must be used to prevent the RDY not connected to a certain device from being driven active during an access to this device (remember, only one RDY input needs to be active to terminate a bus cycle, see Section 3.1.6).

5) The 80186 concurrently provides both a single asynchronous ready input and a single synchronous ready input, while the 8284A provides either two synchronous ready inputs or two asynchronous ready inputs as a user strapable option.

6) The CLOCKOUT (CPU clock output signal) drive capacity of the 80186 is less than the CPU clock drive capacity of the 8284A. This means that not as many high speed devices (e.g., Schottky TTL flip-flops) may be connected to this signal as can be used with the 8284A clock output.

7) The crystal or external oscillator used by the 80186 is twice the CPU clock frequency, while the crystal or external oscillator used with the 8284A is three times the CPU clock frequency.

Local Bus Controller and Control Signals

The 80186 simultaneously provides both local bus controller outputs ($\overline{RD}$, $\overline{WR}$, ALE, $\overline{DEN}$ and DT/$\overline{R}$) and status outputs ($\overline{S0}$, $\overline{S1}$ $\overline{S2}$) for use with the 8288 bus controller. This is different from the 8086 where the local bus controller outputs (generated only in min mode) are sacrificed if status outputs (generated only in max mode) are desired. These differences will manifest themselves in 8086 systems and 80186 systems as follows:

1) Because the 80186 can simultaneously provide local bus control signals and status outputs, many systems supporting both a system bus (e.g.,

a MULTIBUS®) and a local bus will not require two separate external bus controllers, that is, the 80186 bus control signals may be used to control the local bus while the 80186 status signals are concurrently connected to the 8288 bus controller to drive the control signals of the system bus.

2) The ALE signal of the 80186 goes active a clock phase earlier on the 80186 then on the 8086 or 8288. This minimizes address propagation time through the address latches, since typically the delay time through these latches from inputs valid is less than the propagation delay from the strobe input active.

3) The 80186 $\overline{RD}$ input must be tied low to provide queue status outputs from the 80186 (see Figure 28). When so strapped into "queue status mode," the ALE and $\overline{WR}$ outputs provide queue status information. Notice that this queue status information is available one clock phase earlier from the 80186 than from the 8086 (see Figure 29).

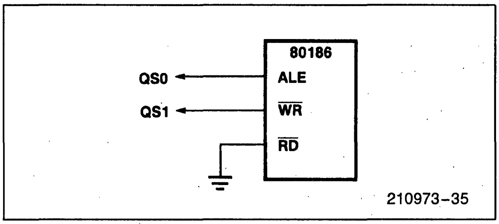


Figure 28. Generating Queue Status Information from the 80186

HOLD/HLDA vs. RQ/GT

As discussed earlier, the 80186 uses a HOLD/HLDA type of protocol for exchanging bus mastership (like the 8086 in min mode) rather than the RQ/GT protocol used by the 8086 in max mode. This allows compatibility with Intel's the new generation of high performance/high integration bus master peripheral devices (for example the 82586 Ethernet controller or 82730 high performance CRT controller/text coprocessor).

Status Information

The 80186 does not provide S3–S5 status information. On the 8086, S3 and S4 provide information regarding the segment register used to generate the physical address of the currently executing bus cycle. S5 provides information concerning the state of the interrupt enable flip-flop. These status bits are always low on the 80186.

Status signal S6 is used to indicate whether the current bus cycle is initiated by either the CPU or a DMA device. Subsequently, it is always low on the 8086. On the 80186, it is low whenever the current bus cycle is initiated by the 80186 CPU, and is high when the current bus cycle is initiated by the 80186 integrated DMA unit.

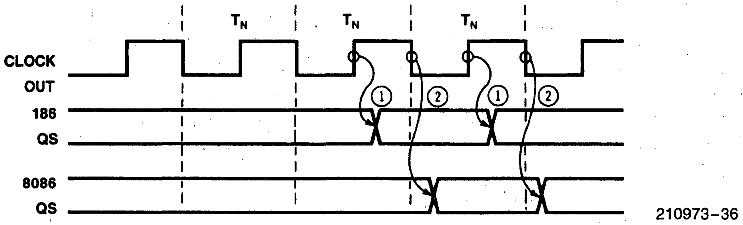
Bus Drive

The 80186 output drivers will drive 200 pF loads. This is double that of the 8086 (100 pF). This allows larger systems to be constructed without the need for bus buffers. It also means that it is very important to provide good grounds to the 80186, since its large drivers can discharge its outputs very quickly causing large current transients on the 80186 ground pins.

Miscellaneous

The 80186 does not provide early and late write signals, as does the 8288 bus controller. The $\overline{WR}$ signal generated by the 80186 corresponds to the early write signal of the 8288. This means that data is not stable on the address/data bus when this signal is driven active.

The 80186 also does not provide differentiated I/O and memory read and write command signals. If these signals are desired, an external 8288 bus controller may be used, or the $\overline{S2}$ signal may be used to synthesize differentiated commands (see Section 3.1.4).



NOTES:
1. 80186 changes queue status off falling edge of CLK
2. 8086 changes queue status off rising edge of CLK

Figure 29. 80186 and 8086 Queue Status Generation

4.0 DMA UNIT INTERFACING

The 80186 includes a DMA unit which provides two independent high speed DMA channels. These channels operate independently of the CPU, and drive all integrated bus interface components (bus controller, chip selects, etc.) exactly as the CPU (see Figure 30). This means that bus cycles initiated by the DMA unit are exactly the same as bus cycles initiated by the CPU (except that S6 = 1 during al! DMA initiated cycles, see Section 3.1). Thus interfacing with the DMA unit itself is very simple, since except for the addition of the DMA request connection, it is exactly the same as interfacing to the CPU.

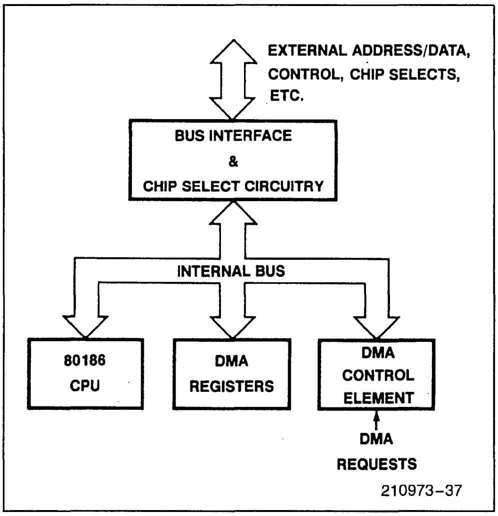

EXTERNAL ADDRESS/DATA, CONTROL, CHIP SELECTS, ETC.
BUS INTERFACE & CHIP SELECT CIRCUITRY
INTERNAL BUS
80186 CPU
DMA REGISTERS
DMA CONTROL ELEMENT
DMA REQUESTS
210973-37

Figure 30. 80186 CPU/DMA Channel Internal Model

4.1 DMA Features

Each of the two DMA channels provides the following features:

- Independent 20-bit source and destination pointers which are used to access the I/O or memory location from which data will be fetched or to which data will be deposited
- Programmable auto-increment, auto-decrement or neither of the source and destination pointers after each DMA transfer
- Programmable termination of DMA activity after a certain number of DMA transfers
- Programmable CPU interruption at DMA termination
- Byte or word DMA transfers to or from even or odd memory or I/O addresses
- Programmable generation of DMA requests by:
  1) the source of the data
  2) the destination of the data
  3) timer 2 (see Section 5)
  4) the DMA unit itself (continuous DMA requests)

4.2 DMA Unit Programming

Each of the two DMA channels contains a number of registers which are used to control channel operation. These registers are included in the 80186 integrated peripheral control block (see Appendix A). These registers include the source and destination pointer registers, the transfer count register and the control register. The layout and interpretation of the bits in these registers is given in Figure 31.

The 20-bit source and destination pointers allow access to the complete 1 Mbyte address space of the 80186, and that all 20 bits are affected by the auto-increment or auto-decrement unit of the DMA (i.e., the DMA channels address the full 1 Mbyte address space of the 80186 as a flat, linear array without segments). When addressing I/O space, the upper 4 bits of the DMA pointer registers should be programmed to be 0. If they are not programmed 0, then the programmed value (greater than 64K in *I/O space*) will be driven onto the address bus (an area of I/O space not accessible to the CPU). The data transfer will occur correctly, however.

After every DMA transfer the 16-bit DMA transfer count register it is decremented by 1, whether a byte transfer or a word transfer has occurred. If the TC bit in the DMA control register is set, the DMA ST/$\overline{\text{STOP}}$ bit (see below) will be cleared when this register goes to 0, causing all DMA activity to cease. A transfer count of zero allows 65536 (2^{16}) transfers.

The DMA control register (see Figure 32) contains bits which control various channel characteristics, including for each of the data source and destination whether the pointer points to memory or I/O space, or whether the pointer will be incremented, decremented or left alone after each DMA transfer. It also contains a bit which selects between byte or word transfers. Two synchronization bits are used to determine the source of the DMA requests (see Section 4.7). The TC bit determines whether DMA activity will cease after a programmed number of DMA transfers, and the INT bit is used to enable interrupts to the processor when this has occurred (note that an interrupt will not be generated to the CPU when the transfer count register reaches zero unless both the INT bit and the TC bit are set).

The control register also contains a start/stop (ST/$\overline{\text{STOP}}$) bit. This bit is used to enable DMA transfers. Whenever this bit is set, the channel is "armed,"

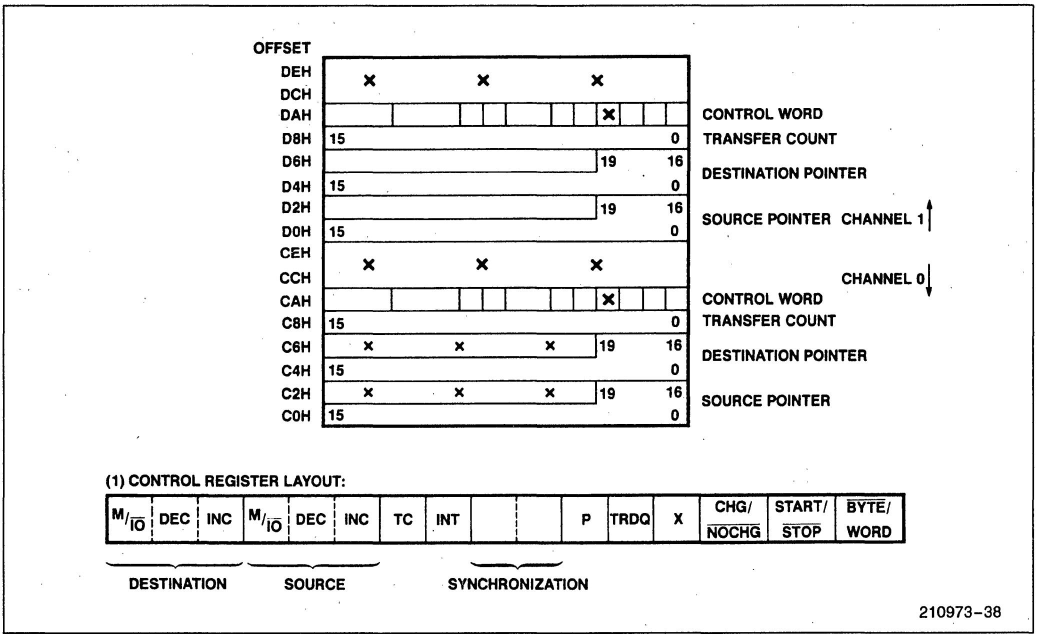


Figure 31. 80186 DMA Register Layout

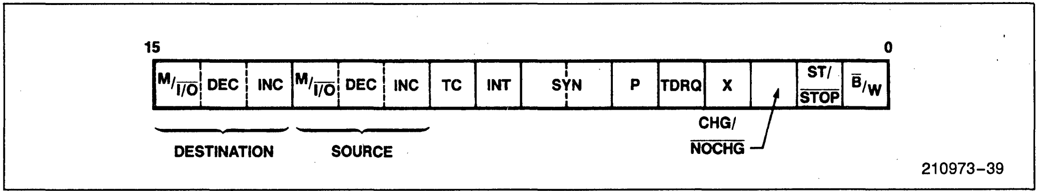
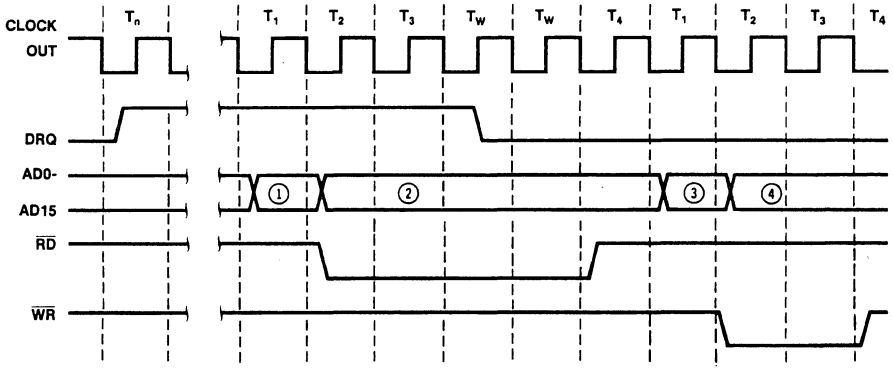


Figure 32. DMA Control Register

that is, a DMA transfer will occur whenever a DMA request is made to the channel. If this bit is cleared, no DMA transfers will be performed by the channel. A companion bit, the CHG/NOCHG bit, allows the contents of the DMA control register to be changed without modifying the state of the start/stop bit. The ST/STOP bit will only be modified if the CHG/NOCHG bit is also set during the write to the DMA control register. The CHG/NOCHG bit is write only. It will always be read back as a 0. Because DMA transfers could occur immediately after the ST/STOP bit is set, it should only be set after all other DMA controller registers have been programmed. This bit is automatically cleared when the transfer count register reaches zero and the TC bit in the DMA control register is set, or when the transfer count register reaches zero and unsynchronized DMA transfers are programmed.

All DMA unit programming registers are directly accessible by the CPU. This means the CPU can, for example, modify the DMA source pointer register after 137 DMA transfers have occurred, and have the new pointer value used for the 138th DMA transfer. If more than one register in the DMA channel is being modified at any time that a DMA request may be generated and the DMA channel is enabled (the ST/STOP bit in the control register is set), the register programming values should be placed in memory locations and moved into the DMA registers using a locked string move instruction. This will prevent a DMA transfer from occurring after only half of the register values have changed. The above also holds true if a read/modify/write type of operation is being performed (e.g., ANDing off bits in a pointer register in a single AND instruction to a pointer register mapped into memory space).



NOTES:
1. Source address
2. Source data
3. Destination address
4. Destination data

Wait states are inserted by the bus condition during the bus cycle, **not** by the DMA controller

Figure 33. Example DMA Transfer Cycle on the 80186

4.3 DMA Transfers

Every DMA transfer in the 80186 consists of two independent bus cycles, the fetch cycle and the deposit cycle (see Figure 33). During the fetch cycle, the byte or word data is accessed from memory or I/O space using the address in the source pointer register. The data accessed is placed in an internal temporary register, which is not accessible to the CPU. During the deposit cycle, the byte or word data in this internal register is placed in memory or I/O space using the address in the destination pointer register. These two bus cycles will not be separated by bus HOLD or by the other DMA channel, and one will never be run without the other except when the CPU is RESET. Notice that the bus cycles run by the DMA unit are exactly the same as memory or I/O bus cycles run by the CPU. The only difference between the two is the state of the S6 status line (which is multiplexed on the A19 line): on all CPU initiated bus cycles, this status line will be driven low; on all DMA initiated bus cycles, this status line will be driven high.

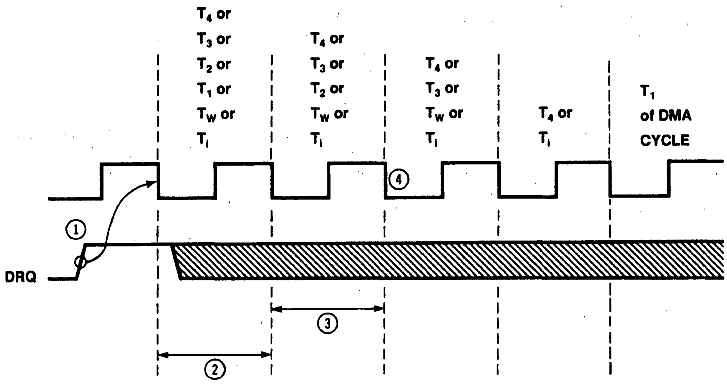
4.4 DMA Requests

Each DMA channel has a single DMA request line by which an external device may request a DMA transfer. The synchronization bits in the DMA control register determine whether this line is interpreted to be connected to the source of the DMA data or the destination of the DMA data. All transfer requests on this line are synchronized to the CPU clock before being presented to internal DMA logic. This means that any asynchronous transitions of the DMA request line will not cause the DMA channel to malfunction. In addition to external requests, DMA requests may be generated whenever the internal Timer 2 times out, or continuously by programming the synchronization bits in the DMA control register to call for unsynchronized DMA transfers.

4.4.1 DMA REQUEST TIMING AND LATENCY

Before any DMA request can be generated, the 80186 internal bus must be granted to the DMA unit. A certain amount of time is required for the CPU to grant this internal bus to the DMA unit. The time between a DMA request being issued and the DMA transfer being run is known as DMA latency. Many of the issues concerning DMA latency are the same as those concerning bus latency (see Section 3.3.2). The only important difference is that external HOLD always has bus priority over an internal DMA transfer. Thus, the latency time of an internal DMA cycle will suffer during an external bus HOLD.

Each DMA channel has a programmed priority relative to the other DMA channel. Both channels may be programmed to be the same priority, or one may be programmed to be of higher priority than the other channel. If both channels are active, DMA latency will suffer on the lower priority channel. If both channels are active and both channels are of the same programmed priority, DMA transfer cycles will alternate between the two channels (i.e., the first channel will perform a



210973-41

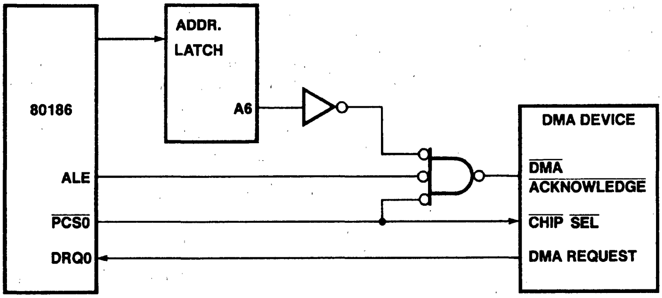
NOTES:
1. t_{DRQCL} = DMA request to clock low = 25 ns to guarantee recognition
2. Synchronizer resolution time
3. DMA unit priority arbitration, etc. time
4. Bus Interface Unit latches DMA request and decides to run DMA cycle

Figure 34. DMA Request Timing on the 80186 (showing minimum response time to request)

fetch and deposit, followed by a fetch and deposit by the second channel, etc.).

The minimum timing required to generate a DMA cycle is shown in Figure 34. Note that the minimum time from DRQ becoming active until the beginning of the first DMA cycle is 4 CPU clock cycles, that is, a DMA request is sampled 4 clock cycles before the beginning of a bus cycle to determine if any DMA activity will be required. This time is independent of the number of wait states inserted in the bus cycle. The maximum DMA latency is a function of other processor activity (see above).

Also notice that if DRQ is sampled active at 1 in Figure 34, the DMA cycle will be executed, even if the DMA request goes inactive before the beginning of the first DMA cycle. This does not mean that the DMA request is latched into the processor such that any transition on the DMA request line will cause a DMA cycle eventually. Quite the contrary, DMA request must be active at a certain time before the end of a bus cycle for the DMA request to be recognized by the processor. If the DMA request line goes inactive before that window, then no DMA cycles will be run.


80186
ALE
$\overline{PCS0}$
DRQ0
ADDR.
LATCH
A6
DMA DEVICE
$\overline{DMA ACKNOWLEDGE}$
$\overline{CHIP}$ $\overline{SEL}$
DMA REQUEST

210973-42

Figure 35. DMA Acknowledge Synthesis from the 80186

4.5 DMA Acknowledge

The 80186 generates no explicit DMA acknowledge signal. Instead, the 80186 performs a read or write directly to the DMA requesting device. If required, a DMA acknowledge signal can be generated by a decode of an address, or by merely using one of the $\overline{PCS}$ lines (see Figure 35). Note ALE must be used to factor the DACK because addresses are not guaranteed stable when chip selects go active. This is required because if the address is not stable when the $\overline{PCS}$ goes active, glitches can occur at the output of the DACK generation circuitry as the address lines change state. Once ALE has gone low, the addresses are guaranteed to have been stable for at least t_{AVAL} (30 ns).

4.6 Internally Generated DMA Requests

There are two types in internally synchronized DMA transfers, that is, transfer initiated by a unit integrated in the 80186. These two types are transfers in which the DMA request is generated by Timer 2, or where DMA request is generated by the DMA channel itself.

The DMA channel can be programmed such that whenever Timer 2 reaches its maximum count, a DMA request will be generated. This feature is selected by setting the TDRQ bit in the DMA channel control register. A DMA request generated in this manner will be latched in the DMA controller, so that once the timer request has been generated, it cannot be cleared except by running the DMA cycle or by clearing the TDRQ bits in both DMA control registers. Before any DMA requests are generated in this mode, Timer 2 must be initiated and enabled.

A timer requested DMA cycle being run by either DMA channel will reset the timer request. Thus, if both channels are using it to request a DMA cycle, only one DMA channel will execute a transfer for every timeout of Timer 2. Another implication of having a single bit timer DMA request latch in the DMA controller is that if another Timer 2 timeout occurs before a DMA channel has a chance to run a DMA transfer, the first request will be lost, i.e., only a single DMA transfer will occur, even though the timer has timed out twice.

The DMA channel can also be programmed to provide its own DMA requests. In this mode, DMA transfer cycles will be run continuously at the maximum bus bandwidth, one after the other until the preprogrammed number of DMA transfers (in the DMA transfer count register) have occurred. This mode is selected by programming the synchronization bits in the DMA control register for unsynchronized transfers. Note that in this mode, the DMA controller will monopolize the CPU bus, i.e., the CPU will not be able to perform opcode fetching, memory operations, etc., while the DMA transfers are occurring. Also notice that the DMA will only perform the number of transfers indicated in the maximum count register regardless of the state of the TC bit in the DMA control register.

4.7 Externally Synchronized DMA Transfers

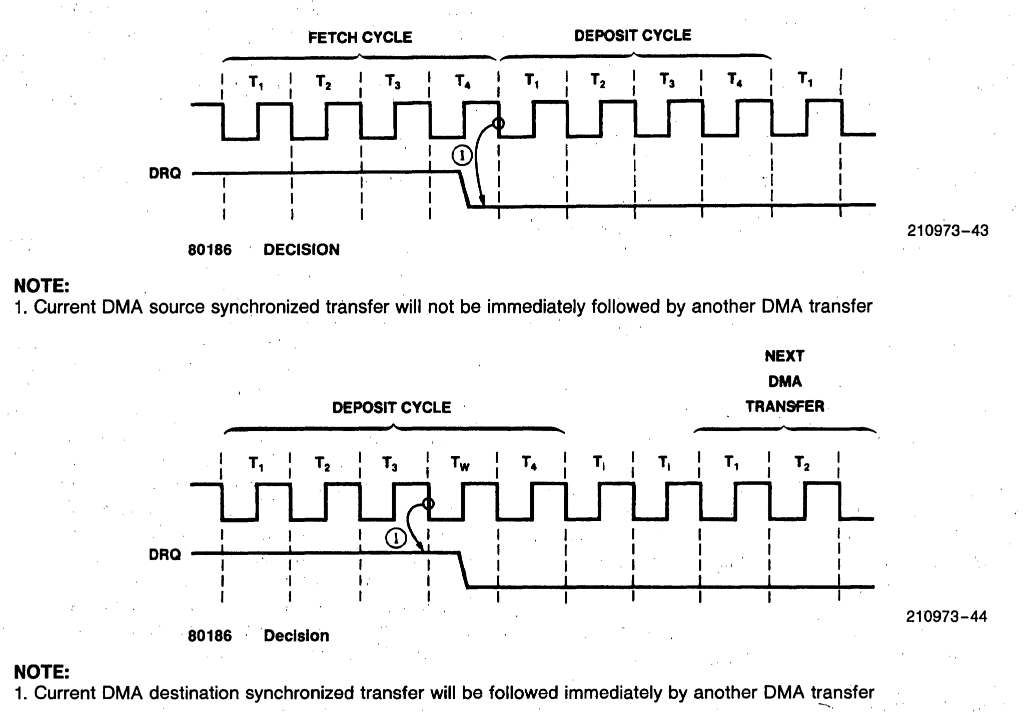
There are two types of externally synchronized DMA transfers, that is, DMA transfers which are requested by an external device rather than by integrated Timer 2 or by the DMA channel itself (in unsynchronized transfers). These are source synchronized and destination synchronized transfers. These modes are selected by programming the synchronization bits in the DMA channel control register. The only difference between the two is the time at which the DMA request pin is sampled to determine if another DMA transfer is immediately required after the currently executing DMA transfer. On source synchronized transfers, this is done such that two source synchronized DMA transfers may occur one immediately after the other, while on destination synchronized transfers a certain amount of idle time is automatically inserted between two DMA transfers to allow time for the DMA requesting device to drive its DMA request inactive.

4.7.1 SOURCE SYNCHRONIZED DMA TRANSFERS

In a source synchronized DMA transfer, the source of the DMA data requests the DMA cycle. An example of this would be a floppy disk read from the disk to main memory. In this type of transfer, the device requesting the transfer is read during the fetch cycle of the DMA transfer. Since it takes 4 CPU clock cycles from the time DMA request is sampled to the time the DMA transfer is actually begun, and a bus cycle takes a minimum of 4 clock cycles, the earliest time the DMA request pin will be sampled for another DMA transfer will be at the beginning of the deposit cycle of a DMA transfer. This allows over 3 CPU clock cycles between the time the DMA requesting device receives an acknowledge to its DMA request (around the beginning of T_2 of the DMA fetch cycle), and the time it must drive this request inactive (assuming no wait states) to insure that another DMA transfer is not performed if it is not desired (see Figure 36).

4.7.2 DESTINATION SYNCHRONIZED DMA TRANSFERS

In destination synchronized DMA transfers, the destination of the DMA data requests the DMA transfer. An example of this would be a floppy disk write from main memory to the disk. In this type of transfer, the device requesting the transfer is written during the de-



NOTE:
1. Current DMA source synchronized transfer will not be immediately followed by another DMA transfer

NOTE:
1. Current DMA destination synchronized transfer will be followed immediately by another DMA transfer

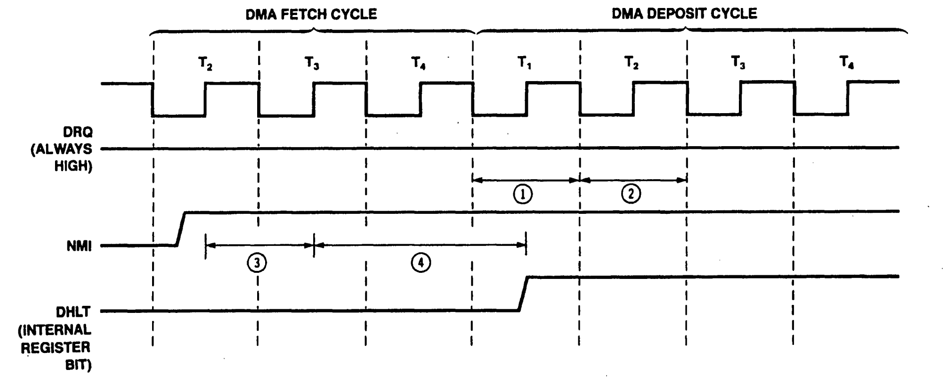
Figure 36. Source & Destination Synchronized DMA Request Timing

posit cycle of the DMA transfer. This causes a problem, since the DMA requesting device will not receive notification of the DMA cycle being run until 3 clock cycles before the end of the DMA transfer (if no wait states are being inserted into the deposit cycle of the DMA transfer) and it takes 4 clock cycles to determine whether another DMA cycle should be run immediately following the current DMA transfer. To get around this problem, the DMA unit will relinquish the CPU bus after each destination synchronized DMA transfer for at least 2 CPU clock cycles to allow the DMA requesting device time to drop its DMA request if it does not immediately desire another immediate DMA transfer. When the bus is relinquished by the DMA unit, the CPU may resume bus operation (e.g., instruction fetching, memory or I/O reads or writes, etc.). Thus, typically, a CPU initiated bus cycle will be inserted between each destination synchronized DMA transfer. If no CPU bus activity is required, however (and none can be guaranteed), the DMA unit will insert only 2 CPU clock cycles between the deposit cycle of one DMA transfer and the fetch cycle of the next DMA transfer. This means that the DMA destination requesting device must drop its DMA request at least two clock cycles before the end of the deposit cycle regardless of the number of wait states inserted into the bus cycle.

Figure 36 shows the DMA request going away too late to prevent the immediate generation of another DMA transfer. Any wait states inserted in the deposit cycle of the DMA transfer will lengthen the amount of time from the beginning of the deposit cycle to the time DMA will be sampled for another DMA transfer. Thus, if the amount of time a device requires to drop its DMA request after receiving a DMA acknowledge from the 80186 is longer than the 0 wait state 80186 maximum (100 ns), wait states can be inserted into the DMA cycle to lengthen the amount of time the device has to drop its DMA request after receiving the DMA acknowledge. Table 4 shows the amount of time between the beginning of T_2 and the time DMA request is sampled as wait states are inserted in the DMA deposit cycle.

Table 4. DMA Request Inactive Timing

Number of Wait States	Max Time (ns) For DRQ Inactive from Start of T_2
0	100
1	225
2	350
3	475



NOTES:

1. DMA request synchronization
2. Decision: Will DMA cycle be run?
   Answer: **No** DMA request is active but DHLT is set (from NMI request)
3. NMI synchronization time
4. Logic delay time from synchronized NMI until DHLT set (note: DHLT is in the interrupt control status register)

Figure 37. NMI and DMA Interaction

4.8 DMA Halt and NMI

Whenever a Non-Maskable Interrupt is received by the 80186, all DMA activity will be suspended after the end of the current DMA transfer. This is performed by the NMI automatically setting the DMA Halt (DHLT) bit in the interrupt controller status register (see Section 6.3.7). The timing of NMI required to prevent a DMA cycle from occurring is shown in Figure 37. After the NMI has been serviced, the DHLT bit should be cleared by the programmer, and DMA activity will resume exactly where it left off, i.e., none of the DMA registers will have been modified. The DHLT bit is not automatically reset after the NMI has been serviced. It is automatically reset by the IRET instruction. This DHLT bit may also be set by the programmer to prevent DMA activity during any critical section of code. However, the DHLT bit is not programmable in the slave mode.

4.9 Example DMA Interfaces

4.9.1 8272 FLOPPY DISK INTERFACE

An example DMA interface to the 8272 Floppy Disk Controller is shown in Figure 38. This shows how a typical DMA device can be interfaced to the 80186. An example floppy disk software driver for this interface is given in Appendix C.

The data lines of the 8272 are connected, through buffers, to the 80186 AD0–AD7 lines. The buffers are required because the 8272 will not float its output drivers quickly enough to prevent contention with the 80186 driven address information after a read from the 8272 (see Section 3.1.3).

DMA acknowledge for the 8272 is driven by an address decode within the region assigned to $\overline{\text{PCS2}}$. If $\overline{\text{PCS2}}$ is assigned to be active between I/O locations 0500H and 057FH, then an access to I/O location 0500H will enable only the chip select, while an access to I/O location 0501H will enable both the chip select and the DMA acknowledge. Remember, ALE must be factored into the DACK generation logic because addresses are not guaranteed stable when the chip selects become active. If ALE were not used, the DACK generation circuitry could glitch as address output changed state while the chip select was active.

Notice that the TC line of the 8272 is driven by a very similar circuit as the one generating DACK (except for the reversed sense of the output!). This line is used to terminate an 8272 command before the command has completed execution. Thus, the TC input to the 8272 is software driven in this case. Another method of driving the TC input would be to connect the DACK signal to one of the 80186 timers, and program the timer to out-

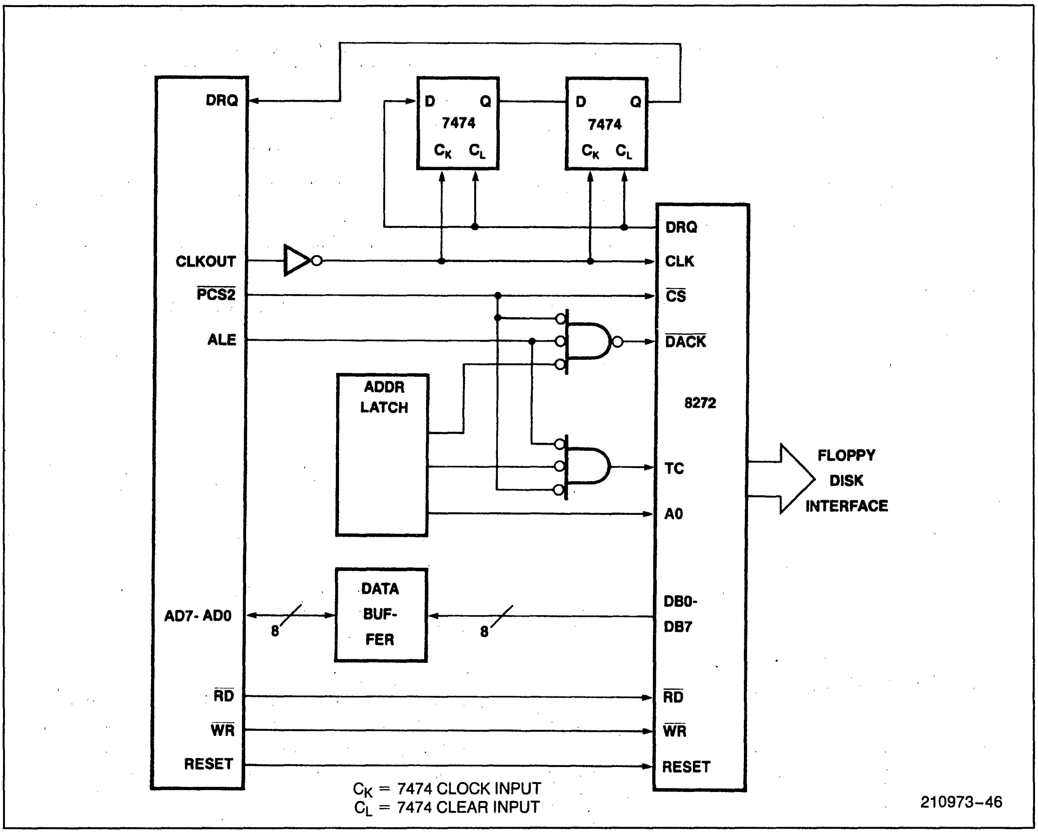


Figure 38. Example 8272/80186 DMA Interface

put a pulse to the 8272 after a certain number of DMA cycles have been run (see next section for 80186 timer information).

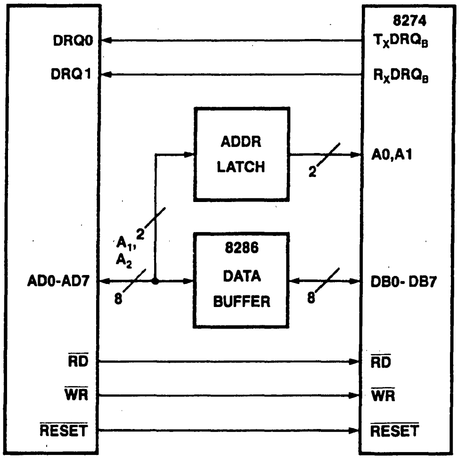
The above discussion assumed that a single 80186 $\overline{\text{PCS}}$ line is free to generate all 8272 select signals. If more than one chip select is free, however, different 80186 generated $\overline{\text{PCS}}$ lines could be used for each function. For example, $\overline{\text{PCS2}}$ could be used to select the 8272, $\overline{\text{PCS3}}$ could be used to drive the DACK line of the 8272, etc.

DMA requests are delayed by two clock periods in going from the 8272 to the 80186. This is required by the 8272 t_{RQR} (time from DMA request to DMA $\overline{\text{RD}}$ going active) spec of 800 ns min. This requires 6.4 80186 CPU clock cycles (at 8 MHz), well beyond the 5 minimum provided by the 80186 (4 clock cycles to the beginning of the DMA bus cycle, 5 to the beginning of T_2 of the DMA bus cycle where $\overline{\text{RD}}$ will go active). The two flip-flops add two complete CPU clock cycles to this response time.

DMA request will go away 200 ns after DACK is presented to the 8272. During a DMA write cycle (i.e., a destination synchronized transfer), this is not soon enough to prevent the immediate generation of another DMA transfer if no wait states are inserted in the deposit cycle to the 8272. Therefore, at least 1 wait state is required by this interface, regardless of the data access parameters of the 8272.

4.9.2 8274 SERIAL COMMUNICATION INTERFACE

An example 8274 synchronous/asynchronous serial chip/80186 DMA interface is shown in Figure 39. The 8274 interface is even simpler than the 8272 interface, since it does not require the generation of a DMA acknowledge signal, and the 8274 does not require the length of time between a DMA request and the DMA read or write cycle that the 8272 does. An example serial driver using the 8274 in DMA mode with the 80186 is given in Appendix C.



210973-47

Figure 39. Example 8274/80186 DMA Interface

The data lines of the 8274 are connected through buffers to the 80186 AD0–AD7 lines. Again, these are required not because of bus drive problems, but because the 8274 will not float its drivers before the 80186 will begin driving address information on its address/data bus. If both the 8274 and the 8272 are included in the same 80186 system, they could share the same data bus buffer (as could any other peripheral devices in the system).

The 8274 does not require a DMA acknowledge signal. The first read from or write to the data register of the 8274 after the 8274 generates the DMA request signal will clear the DMA request. The time between when the control signal ($\overline{RD}$ or $\overline{WR}$) becomes active and when the 8274 will drop its DMA request during a DMA write is 150 ns, which will require at least one wait state be inserted into the DMA write cycle for proper operation of the interface.

5.0 TIMER UNIT INTERFACING

The 80186 includes a timer unit which provides three independent 16-bit timers. These timers operate independently of the CPU. Two of these have input and output pins allowing counting of external events and generation of arbitrary waveforms. The third timer can be used as a timer, as a prescaler for the other two timers, or as a DMA request source.

5.1 Timer Operation

The internal timer unit on the 80186 could be modeled by a single counter element, time multiplexed to three register banks, each of which contains different control and count values. These register banks are, in turn, dual ported between the counter element and the 80186 CPU (see Figure 40). Figure 41 shows the timer element sequencing, and the subsequent constraints on input and output signals. If the CPU modifies one of the timer registers, this change will affect the counter element the next time that register is presented to the counter element. There is no connection between the sequencing of the counter element through the timer register banks and the Bus Interface Unit's sequencing through T-states. Timer operation and bus interface operation are completely asynchronous.

5.2 Timer Registers

Each timer is controlled by a block of registers (see Figure 42). Each of these registers can be read or written whether or not the timer is operating. All processor accesses to these registers are synchronized to all counter element accesses to these registers, meaning that one will never read a count register in which only half of the bits have been modified. Because of this synchronization, one wait state is automatically inserted into any access to the timer registers. Unlike the DMA unit, locking accesses to timer registers will not prevent the timer's counter elements from accessing the timer registers.

Each timer has a 16-bit count register. This register is incremented for each timer event. A timer event can be a low-to-high transition on the external pin (for Timers 0 and 1), a CPU clock transition (divided by 4 because of the counter element multiplexing), or a time out of timer 2 (for Timers 0 and 1). Because the count register is 16 bits wide, up to 65536 (2^{16}) timer events can be counted by a single timer/counter. This register can be both read or written whether the timer is or is not operating.

Each timer includes a maximum count register. Whenever the timer count register is equal to the maximum count register, the count register will be reset to zero, that is, the maximum count value will never be stored in the count register. This maximum count value may

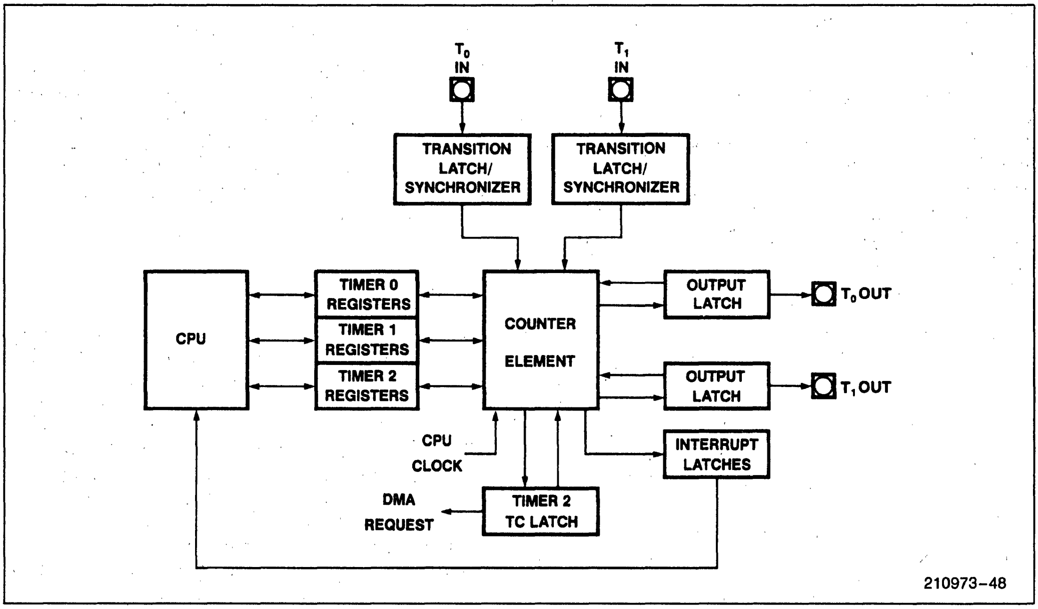
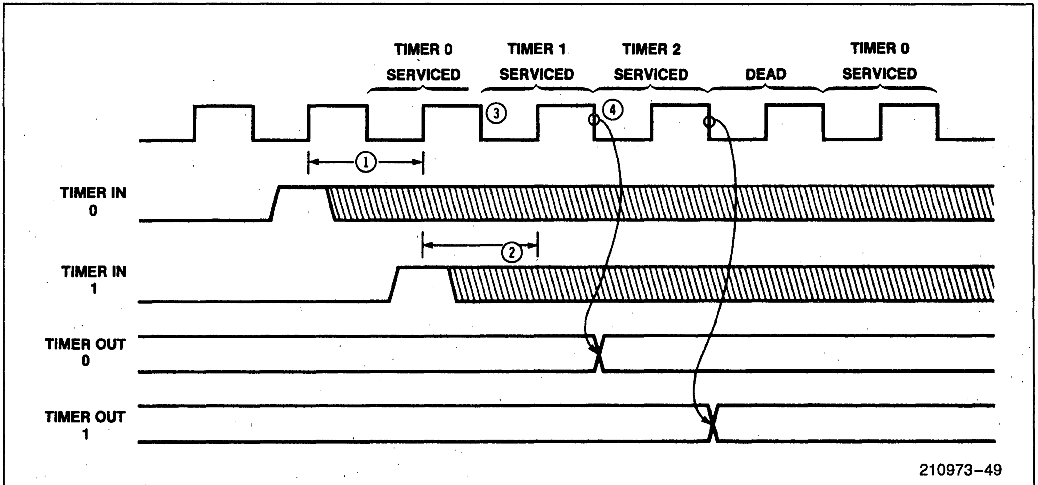
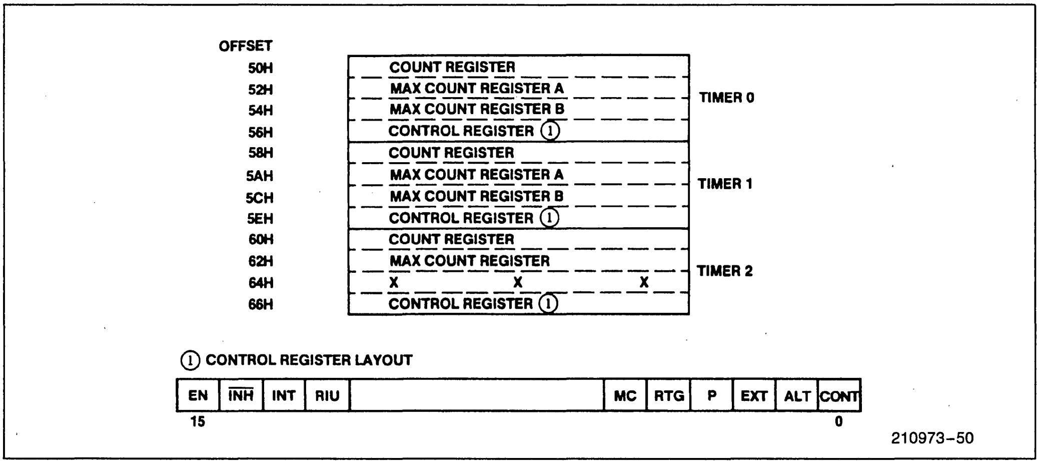


Figure 40. 80186 Timer Model



NOTES:
1. Timer in 0 resolution time
2. Timer in 1 resolution time
3. Modified count value written into 80186 Timer 0 count register
4. Modified count value written into 80186 Timer 1 count register

Figure 41. 80186 Counter Element Multiplexing and Timer Input Synchronization



OFFSET		
50H	COUNT REGISTER	TIMER 0
52H	MAX COUNT REGISTER A	
54H	MAX COUNT REGISTER B	
56H	CONTROL REGISTER (1)	
58H	COUNT REGISTER	TIMER 1
5AH	MAX COUNT REGISTER A	
5CH	MAX COUNT REGISTER B	
5EH	CONTROL REGISTER (1)	
60H	COUNT REGISTER	TIMER 2
62H	MAX COUNT REGISTER	
64H	X X X	
66H	CONTROL REGISTER (1)	

(1) CONTROL REGISTER LAYOUT

EN	$\overline{\text{INH}}$	INT	RIU		MC	RTG	P	EXT	ALT	CONT
15										0

210973-50

Figure 42. 80186 Timer Register Layout

be written while the timer is operating. A maximum count value of 0 implies a maximum count of 65536, a maximum count value of 1 implies a maximum count of 1, etc. The user should be aware that only equivalence between the count value and the maximum count register value is checked, that is, the count value will not be cleared if the value in the count register is greater than the value in the maximum count register. This could only occur by programmer intervention, either by setting the value in the count register greater than the value in the maximum count register, or by setting the value in the maximum count register to be less than the value in the count register. If this is programmed, the timer will count to the maximum possible count (FFFFH), increment to 0, then count up to the value in the maximum count register. The TC bit in the timer control register will not be set when the counter overflows to 0, nor will an interrupt be generated from the timer unit.

Timers 0 and 1 each contain an additional maximum count register. When both maximum count registers are used, the timer will first count up to the value in maximum count register A, reset to zero, count up to the value in maximum count register B, and reset to zero again. The ALTernate bit in the timer control register determines whether one or both maximum count registers are used. If this bit is low, only maximum count register A is used; maximum count register B is ignored. If it is high, both maximum count register A and maximum count register B are used. The RIU (register in use) bit in the timer control register indicates which maximum count register is currently being used. This bit is 0 when maximum count register A is being used, 1 when maximum count register B is being used. This RIU bit is read only. It is unaffected by any write to the timer control register. It will always be read 0 in single maximum count register mode (since only maximum count register A will be used).

Each timer can generate an interrupt whenever the timer count value reaches a maximum count value. That is, an interrupt can be generated whenever the value in maximum count register A is reached, and whenever the value in maximum count register B is reached. In addition, the MC (maximum count) bit in the timer control register is set whenever the timer count reaches a maximum count value. This bit is never automatically cleared, i.e., programmer intervention is required to clear this bit. If a timer generates a second interrupt request before the first interrupt request has been serviced, the first interrupt request to the CPU will be lost.

Each timer has an ENable bit in the timer control register. This bit is used to enable the timer to count. The timer will count timer events only when this bit is set. Any timer events occurring when this bit is reset are ignored. Any write to the timer control register will modify the ENable bit only if the INHibit bit is also set. The timer ENable bit will not be modified by a write to the timer control register if the INHibit bit is not set. The INHibit bit in the timer control register allows selective updating of the timer ENable bit. The value of the INHibit bit is not stored in a write to the timer control register; it will always be read as a 1.

Each timer has a CONTinuous bit in the timer control register. If this bit is cleared, the timer ENable bit will be automatically cleared at the end of each timing cycle. If a single maximum count register is used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register A. If dual maximum count registers are used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register B. If the CONTinuous bit is set, the ENable bit in the timer control register will never be automatically reset. Thus, after each timing cycle, another timing cycle will automatically begin. For example, in single maximum count register mode, the timer will count up to the value in maximum count register A, reset to zero, ad infinitum. In dual maximum count register mode, the timer will count up the value in maximum count register A, reset to zero, count up the value in maximum count register B, reset to zero, count up to the value in maximum count register A, reset to zero, et cetera.

5.3 Timer Events

Each timer counts timer events. All timers can use a transition of the CPU clock as an event. Because of the counter element multiplexing, the timer count value will be incremented every fourth CPU clock. For Timer 2, this is the only timer event which can be used. For Timers 0 and 1, this event is selected by clearing the EXTernal and Prescaler bits in the timer control register.

Timers 0 and 1 can use Timer 2 reaching its maximum count as a timer event. This is selected by clearing the EXTernal bit and setting the Prescaler bit in the timer control register. When this is done, the timer will increment whenever Timer 2 resets to zero having reached its own maximum count. Note that Timer 2 must be initialized and running for the other timer's value to be incremented.

Timers 0 and 1 can also be programmed to count low-to-high transitions on the external input pin. Each transition on the external pin is synchronized to the 80186 clock before it is presented to the timer circuitry, and may, therefore, be asynchronous (see Appendix B for information on 80186 synchronizers). The timer counts transitions on the input pin: the input value must go low, then go high to cause the timer increment. Any transition on this line is latched. If a transition occurs when a timer is not being serviced by the counter element, the transition on the input line will be remembered so that when the timer does get serviced, the input transition will be counted. Because of the counter element multiplexing, the maximum rate at which the timer can count is 1/4 of the CPU clock rate (2 MHz with an 8 MHz CPU clock).

5.4 Timer Input Pin Operation

Timers 0 and 1 each have individual timer input pins. All low-to-high transitions on these input pins are synchronized, latched, and presented to the counter element when the particular timer is being serviced by the counter element.

Signals on this input can affect timer operation in three different ways. The manner in which the pin signals are used is determined by the EXTernal and RTG (retrigger) bits in the timer control register. If the EXTernal bit is set, transitions on the input pin will cause the timer count value to increment if the timer is enabled (the ENable bit in the timer control register is set). Thus, the timer counts external events. If the EXTernal bit is cleared, all timer increments are caused by either the CPU clock or by Timer 2 timing out. In this mode, the RTG bit determines whether the input pin will enable timer operation, or whether it will retrigger timer operation.

If the EXTernal bit is low and the RTG bit is also low, the timer will count internal timer events only when the timer input pin is high and the ENable bit in the timer control register is set. Note that in this mode, the pin is level sensitive, not edge sensitive. A low-to-high transition on the timer input pin is not required to enable timer operation. If the input is tied high, the timer will be continually enabled. The timer enable input signal is completely independent of the ENable bit in the timer control register: both must be high for the timer to count. Example uses for the timer in this mode would be a real time clock or a baud rate generator.

If the EXTernal bit is low and the RTG bit is high, the timer will act as a digital one-shot. In this mode, every low-to-high transition on the timer input pin will cause the timer to reset to zero. If the timer is enabled (i.e., the ENable bit in the timer control register is set) timer operation will begin (the timer will count CPU clock transitions or Timer 2 timeouts). Timer operation will cease at the end of a timer cycle, that is, when the value in the maximum count register A is reached and the timer count value resets to zero (in single maximum count register mode, remember that the maximum count value is never stored in the timer count register) or when the value in maximum count register B is reached and the timer count value resets to zero (in dual maximum count register mode). If another low-to-high transition occurs on the input pin before the end of the timer cycle, the timer will reset to zero and begin the timing cycle again regardless of the state of the CONtinuous bit in the timer control register the RIU bit will not be changed by the input transition. If the CONtinuous bit in the timer control register is cleared, the timer ENable bit will automatically be cleared at the end of the timer cycle. This means that any additional transitions on the input pin will be ignored by the timer. If the CONtinuous bit in the timer control register is set, the timer will reset to zero and begin another timing cycle for every low-to-high transition on the input pin, regardless of whether the timer had reached the end of a timer cycle, because the timer ENable bit would not have been cleared at the end of the timing cycle. The timer will also continue counting at the end of a timer cycle, whether or not another transition has occurred on the input pin. An example use of the timer in this mode is an alarm clock time out signal or interrupt.

5.5 Timer Output Pin Operation

Timers 0 and 1 each contain a single timer output pin. This pin can perform two functions at programmer option. The first is a single pulse indicating the end of a timing cycle. The second is a level indicating the maximum count register currently being used. The timer outputs operate as outlined below whether internal or external clocking of the timer is used. If external clocking is used, however, the user should remember that the time between an external transition on the timer input pin and the time this transition is reflected in the timer out pin will vary depending on when the input transition occurs relative to the timer's being serviced by the counter element.

When the timer is in single maximum count register mode (the ALTernate bit in the timer control register is cleared) the timer output pin will go low for a single CPU clock the clock after the timer is serviced by the counter element where maximum count is reached (see Figure 43). This mode is useful when using the timer as a baud rate generator.

When the timer is programmed in dual maximum count register mode (the ALTernate bit in the timer control register is set), the timer output pin indicates which maximum count register is being used. It is low if maximum count register B is being used for the current count, high if maximum count register A is being used. If the timer is programmed in continuous mode (the CONTinuous bit in the timer control register is set), this pin could generate a waveform of any duty cycle. For example, if maximum count register A contained 10 and maximum count register B contained 20, a 33% duty cycle waveform would be generated.

5.6 Sample 80186 Timer Applications

The 80186 timers can be used for almost any application for which a discrete timer circuit would be used. These include real time clocks, baud rate generators, or event counters.

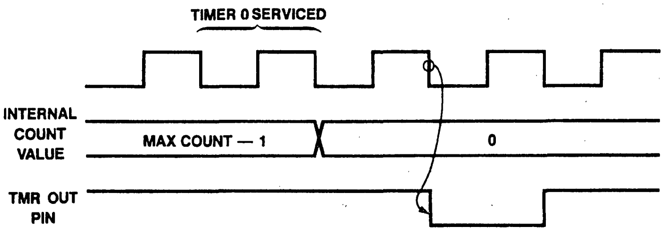


Figure 43. 80186 Timer Out Signal

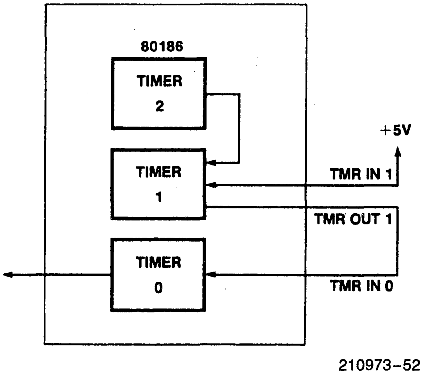


Figure 44. 80186 Real Time Clock

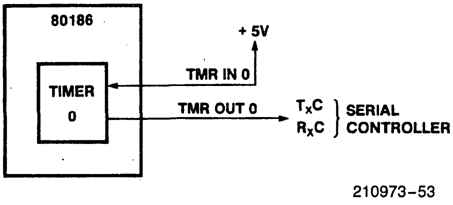


Figure 45. 80186 Baud Rate Generator

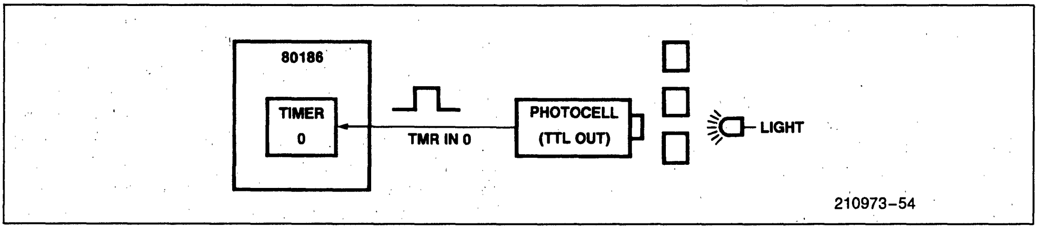


Figure 46

5.6.1 80186 TIMER REAL TIME CLOCK

The sample program in appendix D shows the 80186 timer being used with the 80186 CPU to form a real time clock. In this implementation, Timer 2 is programmed to provide an interrupt to the CPU every millisecond. The CPU then increments memory based clock variables.

5.6.2 80186 TIMER BAUD RATE GENERATOR

The 80186 timers can also be used as baud rate generators for serial communication controllers (e.g., the 8274). Figure 45 shows this simple connection, and the code to program the timer as a baud rate generator is included in Appendix D.

5.6.3 80186 TIMER EVENT COUNTER

The 80186 timer can be used to count events. Figure 46 shows a hypothetical set up in which the 80186 timer will count the interruptions in a light source. The number of interruptions can be read directly from the count register of the timer, since the timer counts up, i.e., each interruption in the light source will cause the timer count value to increase. The code to set up the 80186 timer in this mode is included in Appendix D.

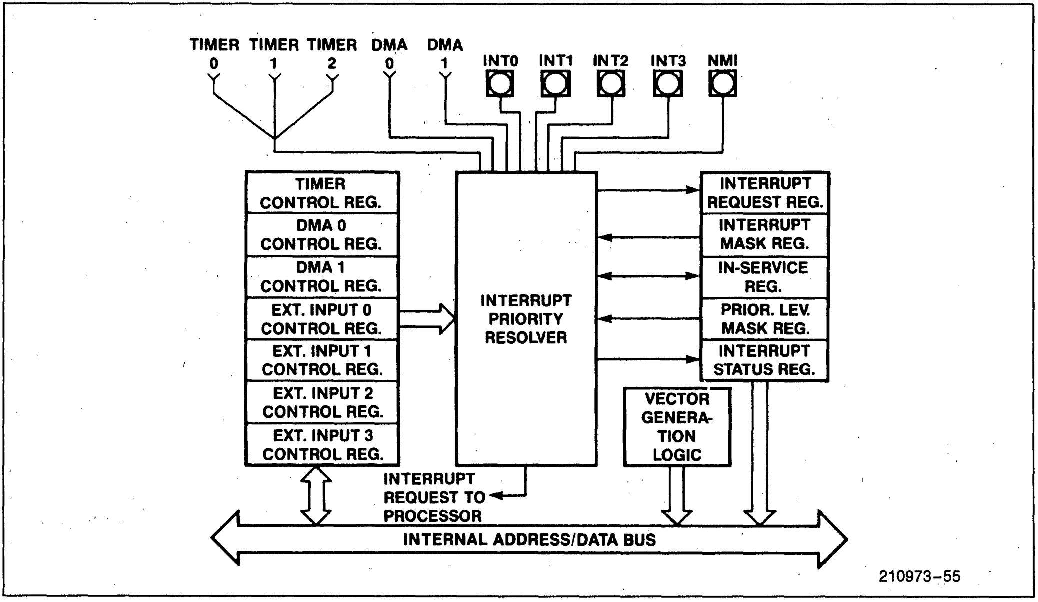


Figure 47. 80186 Interrupt Controller Block Diagram

6.0 80186 INTERRUPT CONTROLLER INTERFACING

The 80186 contains an integrated interrupt controller. This unit performs tasks of the interrupt controller in a typical system. These include synchronization of interrupt requests, prioritization of interrupt requests, and request type vectoring in response to a CPU interrupt acknowledge. It can be a master to two external 8259A interrupt controllers or can be a slave to an external interrupt controller.

6.1 Interrupt Controller Model

The integrated interrupt controller block diagram is shown in Figure 47. It contains registers and a control element. Four inputs are provided for external interfacing to the interrupt controller. Their functions change according to the programmed mode of the interrupt controller. Like the other 80186 integrated peripheral registers, the interrupt controller registers are available for CPU reading or writing at any time.

6.2 Interrupt Controller Operation

The interrupt controller operates in two major modes, **master** and **slave** mode. In master mode the integrated controller acts as the master interrupt controller for the system, while in slave mode the controller operates as a slave to an external interrupt controller which operates as the master interrupt controller for the system. Some of the interrupt controller registers and interrupt controller pins change definition between these two modes, but the basic charter and function of the interrupt controller remains fundamentally the same. The difference is when in master mode, the interrupt controller presents its interrupt input directly to the 80186 CPU, while in slave mode the interrupt controller presents its interrupt input to an external controller (which then presents its interrupt input to the 80186 CPU). Placing the interrupt controller in slave mode is done by setting the SLAVE/$\overline{\text{MASTER}}$ bit in the peripheral control block pointer (see Appendix A).

6.3 Interrupt Controller Registers

The interrupt controller has a number of registers which are used to control its operation (see Figure 48). Some of these change their function between the two major modes of the interrupt controller (master and slave mode). The differences are indicated in the following section. If not indicated, the function and implementation of the registers is the same in the two basic modes of operation of the interrupt controller. The method of interaction among the various interrupt controller registers is shown in the flowcharts in Figures 56 and 57.

6.3.1 CONTROL REGISTERS

Each source of interrupt to the 80186 has a control register in the internal controller. These registers con-

MASTER MODE	OFFSET ADDRESS	SLAVE MODE
INT3 CONTROL REGISTER	3EH	①
INT2 CONTROL REGISTER	3CH	①
INT1 CONTROL REGISTER	3AH	TIMER 2 CONTROL REGISTER
INT0 CONTROL REGISTER	38H	TIMER 1 CONTROL REGISTER
DMA1 CONTROL REGISTER	36H	DMA1 CONTROL REGISTER
DMA0 CONTROL REGISTER	34H	DMA0 CONTROL REGISTER
TIMER CONTROL REGISTER	32H	TIMER 0 CONTROL REGISTER
INTERRUPT CONTROLLER STATUS REGISTER	30H	INTERRUPT CONTROLLER STATUS REGISTER
INTERRUPT REQUEST REGISTER	2EH	INTERRUPT REQUEST REGISTER
IN-SERVICE REGISTER	2CH	IN SERVICE REGISTER
PRIORITY MASK REGISTER	2AH	PRIORITY MASK REGISTER
MASK REGISTER	28H	MASK REGISTER
POLL STATUS REGISTER	26H	①
POLL REGISTER	24H	①
EOI REGISTER	22H	SPECIFIC EOI REGISTER
①	20H	INTERRUPT VECTOR REGISTER

210973-56

NOTE:
1. Unsupported in this mode: values written may or may not be stored

Figure 48. 80186 Interrupt Controller Registers

15								0
0		0	SPECIAL FULLY NESTED BIT ②	CAS-CADE MODE ②	LEVEL TRIG. MODE ①	MASK BIT	PRIORITY BITS	

210973-57

NOTE:
1. This bit present only in INT0–INT3 control registers
2. These bits present only in INT0–INT1 control register

Figure 49. Interrupt Controller Control Register

MASTER MODE

15										0
X	X	X	I3	I2	I1	I0	D1	D0	X	TMR

SLAVE MODE

15									0
X	X	X	TMR 2	TMR 1	D0	D1	X	TMR 0	

210973-58

Figure 50. 80186 Interrupt Controller In-Service, Interrupt Request and Mask Register Format

tain three bits which select one of eight different interrupt priority levels for the interrupt device (0 is highest priority, 7 is lowest priority), and a mask bit to enable the interrupt (see Figure 49). When the mask bit is low, the interrupt is enabled, when it is high, the interrupt is masked.

There are seven control registers in the 80186 integrated interrupt controller. In master mode, four of these serve the external interrupt inputs, one each for the two DMA channels, and one for the collective timer interrupts. In slave mode, the external interrupt inputs are not used, so each timer can have its own individual control register.

6.3.2 REQUEST REGISTER

The interrupt controller includes an interrupt request register (see Figure 50). This register contains seven active bits, one for each interrupt control register. Whenever an interrupt request is made by the interrupt source associated with a specific control register, the bit in interrupt request register is set, regardless if the interrupt is enabled, or if it is of sufficient priority to cause a processor interrupt. The bits in this register which are associated with integrated peripheral devices (the DMA and timer units) can be read or written, while the bits in this register which are associated with the external interrupt pins can only be read (values written to them are not stored). These interrupt request bits are automatically cleared when the interrupt is acknowledged.

6.3.3 MASK REGISTER AND PRIORITY MASK REGISTER

The interrupt controller contains a mask register (see Figure 50). This register contains a mask bit for each interrupt source associated with an interrupt control register. The bit for an interrupt source in the mask register is identically the same bit as is provided in the interrupt control register: modifying a mask bit in the control register will also modify it in the mask register, and vice versa.

The interrupt controller also contains a priority mask register (see Figure 51). This register contains three bits which indicate the lowest priority an interrupt may have that will cause an interrupt acknowledge. Interrupts received which have a lower priority will be effectively masked off. Upon reset this register is set to the lowest priority of 7 to enable all interrupts of any priority. This register may be read or written.

15							0
X	X	X	X	X	P2	P1	P0

210973-59

Figure 51. 80186 Interrupt Controller Priority Mask Register Format

6.3.4 IN-SERVICE REGISTER

The interrupt controller contains an in-service register (see Figure 50). A bit in the in-service register is associated with each interrupt control register so that when an interrupt request by the device associated with the control register is acknowledged by the processor (either by the processor running the interrupt acknowledge or by the processor reading the interrupt poll register) the bit is set. The bit is reset when the CPU issues an End Of Interrupt to the interrupt controller. This register may be both read and written, i.e., the CPU may set in-service bits without an interrupt ever occurring, or may reset them without using the EOI function of the interrupt controller.

6.3.5 POLL AND POLL STATUS REGISTERS

The interrupt controller contains both a poll register and a poll status register (see Figure 52). Both of these registers contain the same information. They have a single bit to indicate an interrupt is pending. This bit is set if an interrupt of sufficient priority has been received. It is automatically cleared when the interrupt is acknowledged. If (and only if) an interrupt is pending, they also contain information as to the interrupt type of the highest priority interrupt pending.

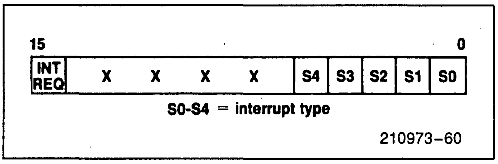


Figure 52. 80186 Poll & Poll Status Register Format

Reading the poll register will acknowledge the pending interrupt to the interrupt controller just as if the processor had acknowledged the interrupt through interrupt acknowledge cycles. The processor will not actually run any interrupt acknowledge cycles, and will not vector through a location in the interrupt vector table. The contents of the interrupt request, in-service, poll, and poll status registers will change appropriately. Reading the poll status register will merely transmit the status of the polling bits without modifying any of the other interrupt controller registers. These registers are read only: data written to them is not stored. These registers are not supported in slave mode. The state of the bits in these registers in slave mode is not defined.

6.3.6 END OF INTERRUPT REGISTER

The interrupt controller contains an End Of Interrupt register (see Figure 53). The programmer issues an End Of Interrupt to the controller by writing to this register. After receiving the End Of Interrupt, the interrupt controller automatically resets the in-service bit for the interrupt. The value of the word written to this register determines whether the End Of Interrupt is specific or non-specific. A non-specific End Of Interrupt is specified by setting the non-specific bit in the word written to the End Of Interrupt register. In a non-specific End Of Interrupt, the in-service bit of the highest priority interrupt set is automatically cleared, while a specific End Of Interrupt allows the in-service bit cleared to be explicitly specified. The in-service bit is reset whether the bit was set by an interrupt acknowledge or if it was set by the CPU writing the bit directly to the in-service register. If the highest priority interrupt is reset, the poll and poll status registers will change to reflect the

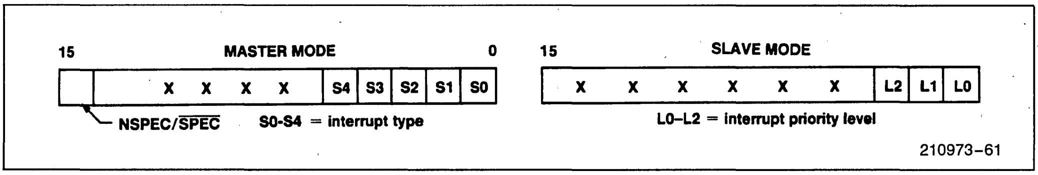


Figure 53. 80186 End of Interrupt Register Format

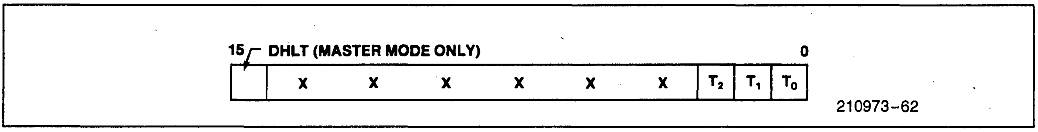


Figure 54. 80186 Interrupt Status Register Format

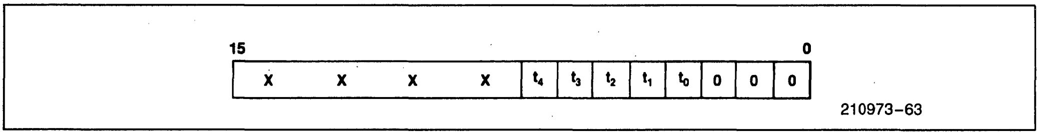


Figure 55. 80186 Interrupt Vector Register Format (slave mode only)

next lowest priority interrupt to be serviced. If a less than highest priority interrupt in-service bit is reset, the priority poll and poll status registers will not be modified (because the highest priority interrupt to be serviced has not changed). Only the specific EOI is supported in slave mode. This register is write only: data written is not stored and cannot be read back.

6.3.7 INTERRUPT STATUS REGISTER

The interrupt controller also contains an interrupt status register (see Figure 54). This register contains four significant bits. There are three bits used to show which timer is causing an interrupt. This is required because in master mode, the timers share a single interrupt control register. A bit in this register is set to indicate which timer has generated an interrupt. The bit associated with a timer is automatically cleared after the interrupt request for the timer is acknowledged. More than one of these bits may be set at a time. The fourth bit in the interrupt status register is the DMA halt bit (not implemented in slave mode). When set, this bit prevents any DMA activity. It is automatically set whenever a NMI is received by the interrupt controller. It can also be set explicitly by the programmer. This bit is automatically cleared whenever the IRET instruction is executed. All significant bits in this register are read/write.

6.3.8 INTERRUPT VECTOR REGISTER

Finally, in slave mode only, the interrupt controller contains an interrupt vector register (see Figure 55). This register is used to specify the 5 most significant bits of the interrupt type vector placed on the CPU bus in response to an interrupt acknowledgement (the lower 3 significant bits of the interrupt type are determined by the priority level of the device causing the interrupt in slave mode).

6.4 Interrupt Sources

The 80186 interrupt controller receives and arbitrates among many different interrupt request sources, both internal and external. Each interrupt source may be programmed to be a different priority level in the interrupt controller. An interrupt request generation flow chart is shown in Figure 56. Such a flowchart would be followed independently by each interrupt source.

6.4.1 INTERNAL INTERRUPT SOURCES

The internal interrupt sources are the three timers and the two DMA channels. An interrupt from each of these interrupt sources is latched in the interrupt controller, so that if the condition causing the interrupt is cleared in the originating integrated peripheral device, the interrupt request will remain pending in the interrupt controller. The state of the pending interrupt can be obtained by reading the interrupt request register of the interrupt controller. For all internal interrupts, the latched interrupt request can be reset by the processor by writing to the interrupt request register. Note that all timers share a common bit in the interrupt request register in master mode. The interrupt controller status register may be read to determine which timer is actually causing the interrupt request in this mode. Each timer has a unique interrupt vector (see Section 6.5.1). Thus polling is not required to determine which timer has caused the interrupt in the interrupt service routine. Also, because the timers share a common interrupt control register, they are placed at a common priority level as referenced to all other interrupt devices. Among themselves they have a fixed priority, with timer 0 as the highest priority timer and timer 2 as the lower priority timer.

6.4.2 EXTERNAL INTERRUPT SOURCES

The 80186 interrupt controller will accept external interrupt requests only when it is programmed in master mode. In this mode, the external pins associated with the interrupt controller may serve either as direct interrupt inputs, or as cascaded interrupt inputs from other interrupt controllers as a programmed option. These options are selected by programming the C and SFNM bits in the INT0 and INT1 control registers (see Figure 49).

When programmed as direct interrupt inputs, the four interrupt inputs are each controlled by an individual interrupt control register. As stated earlier, these registers contain 3 bits which select the priority level for the interrupt and a single bit which enables the interrupt source to the processor. In addition each of these control registers contains a bit which selects either edge or level triggered mode for the interrupt input. When edge triggered mode is selected, a low-to-high transition must occur on the interrupt input before an interrupt is generated, while in level triggered mode, only a high level needs to be maintained to generate an interrupt. In edge triggered mode, the input must remain low at least 1 clock cycle before the input is "re-armed." In both modes, the interrupt level must remain high until the interrupt is acknowledged, i.e., the interrupt request is not latched in the interrupt controller. The status of the interrupt input can be shown by reading the interrupt request register. Each of the external pins has a bit in this register which indicates an interrupt request on the particular pin. Note that since interrupt requests on these inputs are not latched by the interrupt controller, if the external input goes inactive, the interrupt requests (and also the bit in the interrupt request register) will also go inactive (low). Also, if the interrupt input is in edge triggered mode, a low-to-high transition on the input pin must occur before the interrupt request bit will be set in the interrupt request register.

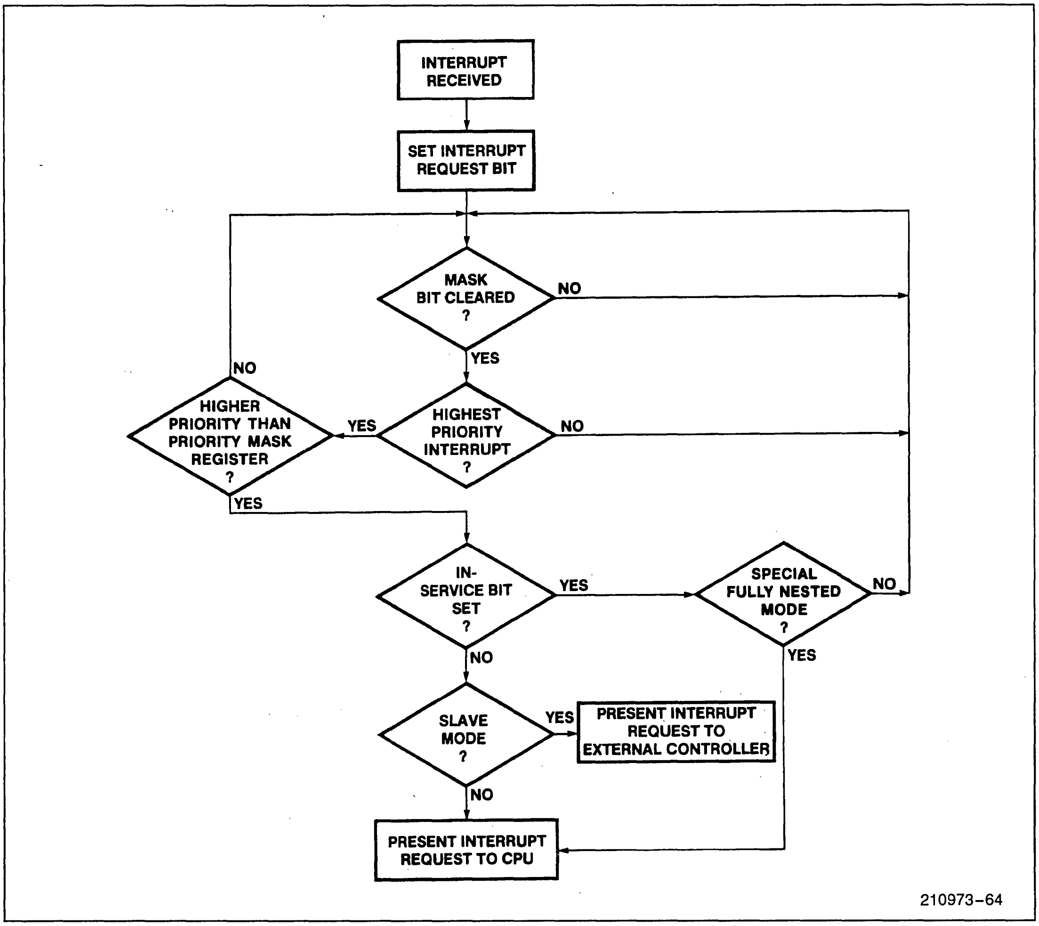

INTERRUPT RECEIVED
SET INTERRUPT REQUEST BIT
MASK BIT CLEARED ?
NO
YES
HIGHEST PRIORITY INTERRUPT ?
NO
YES
HIGHER PRIORITY THAN PRIORITY MASK REGISTER ?
NO
YES
IN-SERVICE BIT SET ?
YES
NO
SPECIAL FULLY NESTED MODE ?
NO
YES
SLAVE MODE ?
YES
NO
PRESENT INTERRUPT REQUEST TO EXTERNAL CONTROLLER
PRESENT INTERRUPT REQUEST TO CPU
210973-64

Figure 56. 80186 Interrupt Request Sequencing

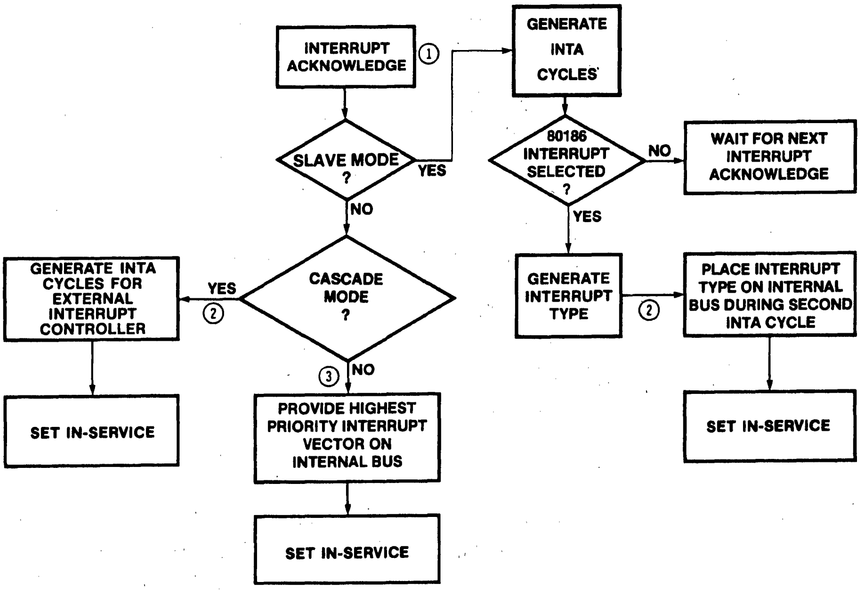
If the C (Cascade) bit of the INT0 or INT1 control registers are set, the interrupt input is cascaded to an external interrupt controller. In this mode, whenever the interrupt presented to the INT0 or INT1 line is acknowledged, the integrated interrupt controller will not provide the interrupt type for the interrupt. Instead, two INTA bus cycles will be run, with the INT2 and INT3 lines providing the interrupt acknowledge pulses for the INT0 and the INT1 interrupt requests respectively. INT0/INT2 and INT1/INT3 may be individually programmed into cascade mode. This allows 128 individually vectored interrupt sources if two banks of 8 external interrupt controllers each are used.

6.4.3 SLAVE MODE INTERRUPT SOURCES

When the interrupt controller is configured in slave mode, the integrated interrupt controller accepts interrupt requests only from the integrated peripherals. Any external interrupt requests must go through an external interrupt controller. This external interrupt controller requests interrupt service directly from the 80186 CPU through the INT0 line on the 80186. In this mode, the function of this line is not affected by the integrated interrupt controller. In addition, in slave mode the integrated interrupt controller must request interrupt service through this external interrupt controller. This interrupt request is made on the INT3 line (see Section 6.6.4 on external interrupt connections).

6.5 Interrupt Response

The 80186 can respond to an interrupt in two different ways. The first will occur if the internal controller is


INTERRUPT ACKNOWLEDGE
①
GENERATE INTA CYCLES
SLAVE MODE ?
YES
NO
80186 INTERRUPT SELECTED ?
NO
WAIT FOR NEXT INTERRUPT ACKNOWLEDGE
YES
GENERATE INTA CYCLES FOR EXTERNAL INTERRUPT CONTROLLER
YES
②
CASCADE MODE ?
GENERATE INTERRUPT TYPE
②
PLACE INTERRUPT TYPE ON INTERNAL BUS DURING SECOND INTA CYCLE
④
③
NO
SET IN-SERVICE
PROVIDE HIGHEST PRIORITY INTERRUPT VECTOR ON INTERNAL BUS
SET IN-SERVICE
SET IN-SERVICE
210973-65

NOTES:
1. Before actual interrupt acknowledge is run by CPU
2. Two interrupt acknowledge cycles will be run, the interrupt type is read by the CPU on the second cycle
3. Interrupt acknowledge cycles will not be run, the interrupt vector address is placed on an internal bus and is not available outside the processor
4. Interrupt type is not driven on external bus in slave mode

Figure 57. 80186 Interrupt Acknowledge Sequencing

providing the interrupt vector information with the controller in master mode. The second will occur if the CPU reads interrupt type information from an external interrupt controller or if the interrupt controller is in slave mode. In both of these instances the interrupt vector information driven by the 80186 integrated interrupt controller is not available outside the 80186 microprocessor.

In each interrupt mode, when the integrated interrupt controller receives an interrupt response, the interrupt controller will automatically set the in-service bit and reset the interrupt request bit in the integrated controller. In addition, unless the interrupt control register for the interrupt is set in Special Fully Nested Mode, the interrupt controller will prevent any interrupts from occurring from the same interrupt line until the in-service bit for that line has been cleared.

6.5.1 INTERNAL VECTORING, MASTER MODE

In master mode, the interrupt types associated with all the interrupt sources are fixed and unalterable. These interrupt types are given in Table 5. In response to an internal CPU interrupt acknowledge the interrupt controller will generate the vector address rather than the interrupt type. On the 80186 (like the 8086) the interrupt vector address is the interrupt type multiplied by 4. This speeds interrupt response.

In master mode, the integrated interrupt controller is the master interrupt controller of the system. As a result, no external interrupt controller need know when the integrated controller is providing an interrupt vector, nor when the interrupt acknowledge is taking place. As a result, no interrupt acknowledge bus cycles will be generated. The first external indication that an interrupt has been acknowledged will be the processor reading the interrupt vector from the interrupt vector table in low memory.

Table 5. 80186 Interrupt Vector Types

Interrupt Name	Vector Type	Default Priority
Timer 0	8	0a
Timer 1	18	0b
Timer 2	19	0c
DMA 0	10	2
DMA 1	11	3
INT 0	12	4
INT 1	13	5
INT 2	14	6
INT 3	15	7

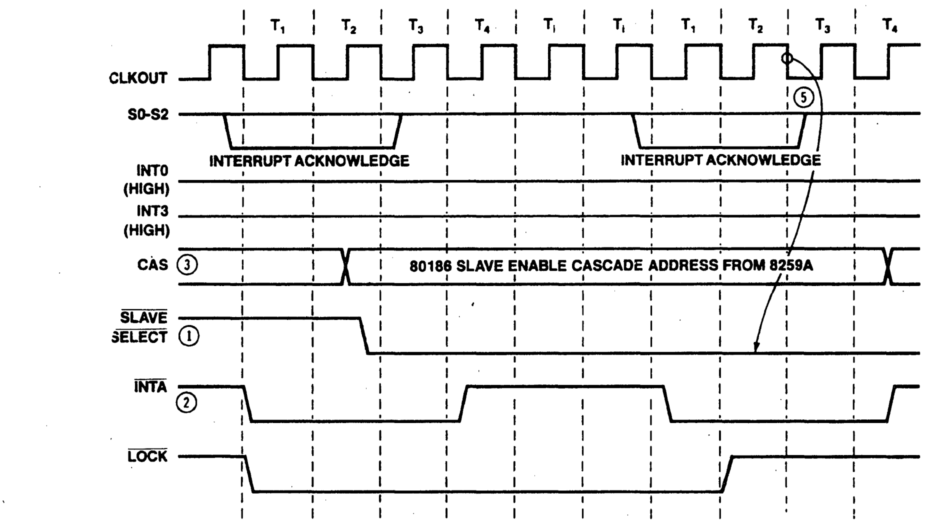
Because the two interrupt acknowledge cycles are not run, and the interrupt vector address does not need to be calculated, interrupt response to an internally vectored interrupt is 42 clock cycles, which is faster than the interrupt response when external vectoring is required, or if the interrupt controller is run in slave mode.

If two interrupts of the same programmed priority occur, the default priority scheme (as shown in Table 5) is used.

6.5.2 INTERNAL VECTORING, SLAVE MODE

In slave mode, the interrupt types associated with the various interrupt sources are alterable. The upper 5 most significant bits are taken from the interrupt vector register, and the lower 3 significant bits are taken from the priority level of the device causing the interrupt. Because the interrupt type, rather than the interrupt vector address, is given by the interrupt controller in this mode the interrupt vector address must be calculated by the CPU before servicing the interrupt.

In slave mode, the integrated interrupt controller will present the interrupt type to the CPU in response to the two interrupt acknowledge bus cycles run by the processor. During the first interrupt acknowledge cycle, the external master interrupt controller determines which slave interrupt controller will be allowed to place its interrupt vector on the microprocessor bus. During the second interrupt acknowledge cycle, the processor reads the interrupt vector from its bus. Thus, these two interrupt acknowledge cycles must be run, since the integrated controller will present the interrupt type information only when the external interrupt controller signals the integrated controller that it has the highest pending interrupt request (see Figure 58). The 80186



NOTES:
1. $\overline{\text{SLAVE}}$ $\overline{\text{SELECT}}$ = INT1
2. $\overline{\text{INTA}}$ = INT 2
3. Driven by external interrupt controller
4. $\overline{\text{SLAVE}}$ $\overline{\text{SELECT}}$ must be driven before Phase 2 of T$_2$ of the second INTA cycle
5. $\overline{\text{SLAVE}}$ $\overline{\text{SELECT}}$ read by 80186

Figure 58. 80186 Slave Mode Interrupt Acknowledge Timing

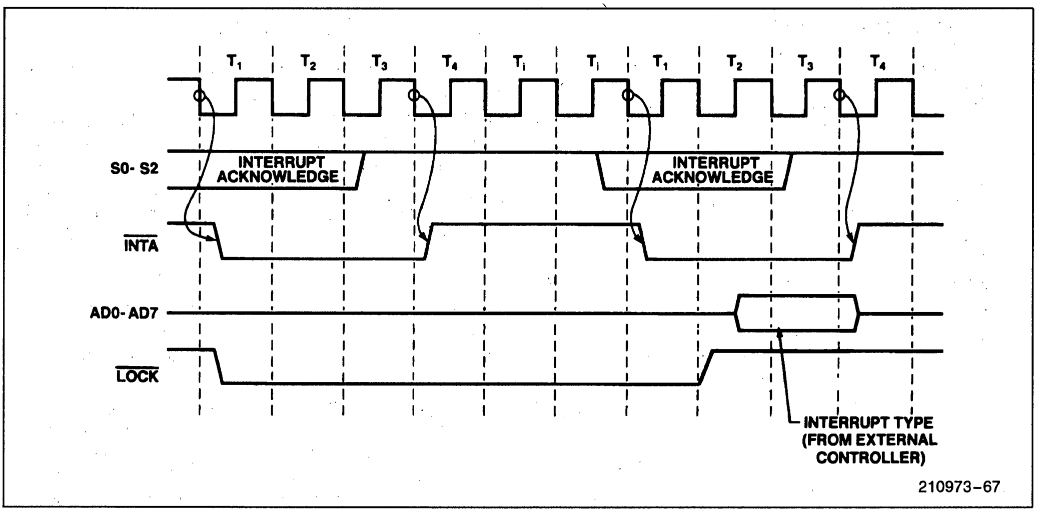


Figure 59. 80186 Cascaded Interrupt Acknowledge Timing

samples the SLAVE SELECT line during the falling edge of the clock at the beginning of T_3 of the second interrupt acknowledge cycle. This input must be stable 20 ns before and 10 ns after this edge.

These two interrupt acknowledge cycles will be run back to back, and will be LOCKED with the LOCK output active (meaning that DMA requests and HOLD requests will not be honored until both cycles have been run). Note that the two interrupt acknowledge cycles will always be separated by two idle T states, and that wait states will be inserted into the interrupt acknowledge cycle if a ready is not returned by the processor bus interface. The two idle T states are inserted to allow compatibility with the timing requirements of an external 8259A interrupt controller.

Because the interrupt acknowledge cycles must be run in slave mode, even for internally generated vectors, and the integrated controller presents an interrupt type rather than a vector address, the interrupt response time here is the same as if an externally vectored interrupt was required, namely 55 CPU clocks.

6.5.3 EXTERNAL VECTORING

External interrupt vectoring occurs whenever the 80186 interrupt controller is placed in cascade mode, special fully nested mode, or slave mode (and the integrated controller is not enabled by the external master interrupt controller). In this mode, the 80186 generates two interrupt acknowledge cycles, reading the in-terrupt type off the lower 8 bits of the address/data bus on the second interrupt acknowledge cycle (see Figure 59). This interrupt response is exactly the same as the 8086, so that the 8259A interrupt controller can be used exactly as it would in an 8086 system. Notice that the two interrupt acknowledge cycles are LOCKED, and that two idle T-states are always inserted between the two interrupt acknowledge bus cycles, and that wait states will be inserted in the interrupt acknowledge cycle if a ready is not returned to the processor. Also notice that the 80186 provides two interrupt acknowledge signals, one for interrupts signaled by the INT0 line, and one for interrupts signaled by the INT1 line (on the INT2/INTA0 and INT3/INTA1 lines, respectively). These two interrupt acknowledge signals are mutually exclusive. Interrupt acknowledge status will be driven on the status lines (S0–S2) when either INT2/INTA0 or INT3/INTA1 signal an interrupt acknowledge.

6.5.4 EFFECT OF LOCK PREFIX ON INTERRUPT ACKNOWLEDGE CYCLES

When the interrupt controller is operating in either the cascade or slave modes and an interrupt occurs during an instruction that has been LOCKED by software, the LOCK signal timing shown in Figures 58 and 59 may be altered. Some peripheral devices used with the 80186 require contiguous INTA cycles to allow correct interrupt controller response. In such cases, the external circuitry in Figure 60 should be used to ensure that DMA or HOLD requests are blocked from "stealing" the bus during INTA cycles.

6.6 Interrupt Controller External Connections

The four interrupt signals can be programmably configured into 3 major options. These are direct interrupt inputs (with the integrated controller providing the interrupt vector), cascaded (with an external interrupt

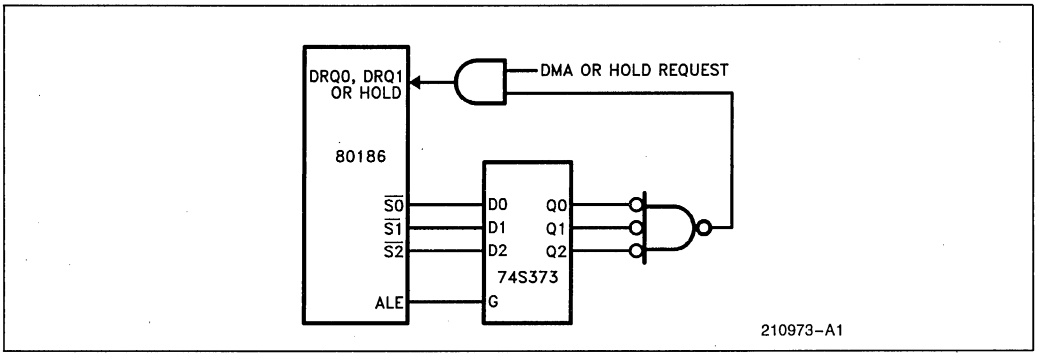


Figure 60. Circuit Blocking DMA or HOLD Request Between $\overline{\text{INTA}}$ Cycles

controller providing the interrupt vector), or slave mode. In all these modes, any interrupt presented to the external lines must remain set until the interrupt is acknowledged.

6.6.1 DIRECT INPUT MODE

When the Cascade mode bits are cleared, the interrupt input lines are configured as direct interrupt input lines (see Figure 61). In this mode an interrupt source (e.g., an 8272 floppy disk controller) may be directly connected to the interrupt input line. Whenever an interrupt is received on the input line, the integrated controller will do nothing unless the interrupt is enabled, and it is the highest priority pending interrupt. At this time, the interrupt controller will present the interrupt to the CPU and wait for an interrupt acknowledge. When the acknowledge occurs, it will present the interrupt vector address to the CPU. In this mode, the CPU will not run any interrupt acknowledge cycles.

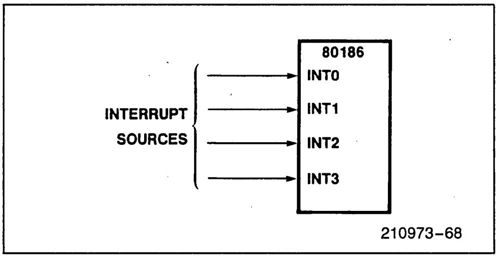


Figure 61. 80186 Non-Cascaded Interrupt Connection

These lines can be individually programmed in either edge or level triggered mode using their respective control registers. In edge triggered mode, a low-to-high transition must occur before the interrupt will be generated to the CPU, while in level triggered mode, only a high level must be present on the input for an interrupt to be generated. In edge trigger mode, the interrupt input must also be low for at least 1 CPU clock cycle to insure recognition. In both modes, the interrupt input must remain active until acknowledged.

6.6.2 CASCADE MODE

When the Cascade mode bit is set and the SFNM bit is cleared, the interrupt input lines are configured in cascade mode. In this mode, the interrupt input line is paired with an interrupt acknowledge line. The INT2/$\overline{\text{INTA0}}$ and INT3/$\overline{\text{INTA1}}$ lines are dual purpose; they can function as direct input lines, or they can function as interrupt acknowledge outputs. INT2/$\overline{\text{INTA0}}$ provides the interrupt acknowledge for an INT0 input, and INT3/$\overline{\text{INTA1}}$ provides the interrupt acknowledge for an INT1 input. Figure 62 shows this connection.

When programmed in this mode, in response to an interrupt request on the INT0 line, the 80186 will provide two interrupt acknowledge pulses. These pulses will be provided on the INT2/$\overline{\text{INTA0}}$ line, and will also be reflected by interrupt acknowledge status being generated on the $\overline{\text{S0}}$–$\overline{\text{S2}}$ status lines. On the second pulse, the interrupt type will be read in. The 80186 externally vectored interrupt response is covered in more detail in Section 6.5.

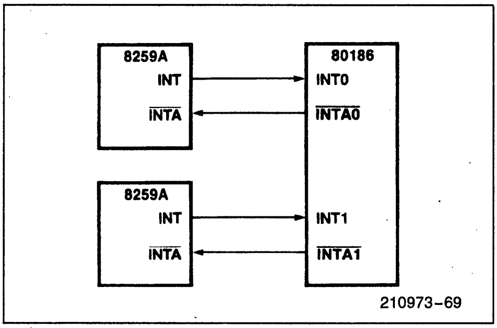


Figure 62. 80186 Cascade and Special Fully Nested Mode Interface

INT0/INT2/$\overline{\text{INTA0}}$ and INT1/INT3/$\overline{\text{INTA1}}$ may be individually programmed into interrupt request/acknowledge pairs, or programmed as direct inputs. This means that INT0/INT2/$\overline{\text{INTA0}}$ may be programmed as an interrupt/acknowledge pair, while INT1 and INT3/$\overline{\text{INTA1}}$ each provide separate internally vectored interrupt inputs.

When an interrupt is received on a cascaded interrupt, the priority mask bits and the in-service bits in the particular interrupt control register will be set into the interrupt controller's mask and priority mask registers. This will prevent the controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Also, since the in-service bit is set, any subsequent interrupt requests on the particular interrupt input line will not cause the integrated interrupt controller to generate an interrupt request to the 80186 CPU. This means that if the external interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the 80186 interrupt request line, it will not subsequently be presented to the 80186 CPU by the integrated interrupt controller until the in-service bit for the interrupt line has been cleared.

6.6.3 SPECIAL FULLY NESTED MODE

When both the Cascade mode bit and the SFNM bit are set, the interrupt input lines are configured in Special Fully Nested Mode. The external interface in this mode is exactly as in Cascade Mode. The only difference is in the conditions allowing an interrupt from the external interrupt controller to the integrated interrupt controller to interrupt the 80186 CPU.

When an interrupt is received from a special fully nested mode interrupt line, it will interrupt the 80186 CPU if it is the highest priority interrupt pending regardless of the state of the in-service bit for the interrupt source in the interrupt controller. When an interrupt is acknowledged from a special fully nested mode interrupt line, the priority mask bits and the in-service bits in the particular interrupt control register will be set into the interrupt controller's in-service and priority mask registers. This will prevent the interrupt controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Unlike cascade mode, however, the interrupt controller will not prevent additional interrupt requests generated by the same external interrupt controller from interrupting the 80186 CPU. This means that if the external (cascaded) interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the integrated controller's interrupt request line, it may cause an interrupt to be generated to the 80186 CPU, regardless of the state of the in-service bit for the interrupt line.

If the SFNM mode bit is set and the Cascade mode bit is not also set, the controller will provide internal interrupt vectoring. It will also ignore the state of the in-service bit in determining whether to present an interrupt request to the CPU. In other words, it will use the SFNM conditions of interrupt generation with an internally vectored interrupt response, i.e., if the interrupt pending is the highest priority type pending, it will cause a CPU interrupt regardless of the state of the in-service bit for the interrupt.

6.6.4 SLAVE MODE

When the SLAVE/$\overline{\text{MASTER}}$ bit in the peripheral relocation register is set, the interrupt controller is set into slave mode. In this mode, all four interrupt controller input lines are used to perform the necessary handshaking with the external master interrupt controller. Figure 63 shows the hardware configuration of the 80186 interrupt lines with an external controller in slave mode.

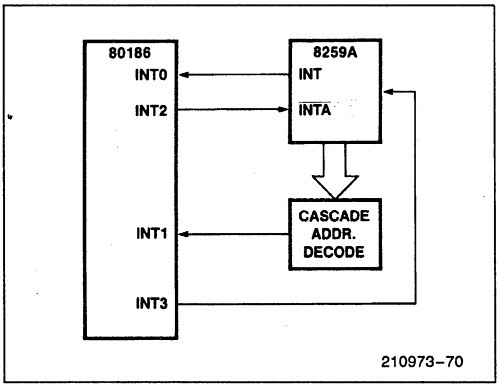


Figure 63. 80186 Slave Mode Interface

Because the integrated interrupt controller is a slave controller, it must be able to generate an interrupt input for an external interrupt controller. It also must be signaled when it has the highest priority pending interrupt to know when to place its interrupt vector on the bus. These two signals are provided by the INT3/Slave Interrupt Output and INT1/Slave Select lines, respectively. The external master interrupt controller must be able to interrupt the 80186 CPU, and needs to know when the interrupt request is acknowledged. The INT0 and INT2/INTA0 lines provide these two functions.

6.7 Example 8259A/Cascade Mode Interface

Figure 64 shows the 80186 and 8259A in cascade interrupt mode. The code to initialize the 80186 interrupt controller is given in Appendix E. Notice that an "interrupt ready" signal must be returned to the 80186 to prevent the generation of wait states in response to the interrupt acknowledge cycles. In this configuration the INT0 and INT2 lines are used as direct interrupt input lines. Thus, this configuration provides 10 external interrupt lines: 2 provided by the 80186 interrupt controller itself, and 8 from the external 8259A. Also, the 8259A is configured as a master interrupt controller. It will only receive interrupt acknowledge pulses in response to an interrupt it has generated. It may be cascaded again to up to 8 additional 8259As (each of which would be configured in slave mode).

6.8 Interrupt Latency

Interrupt latency time is the time from when the 80186 receives the interrupt to the time it begins to respond to the interrupt. This is different from interrupt response

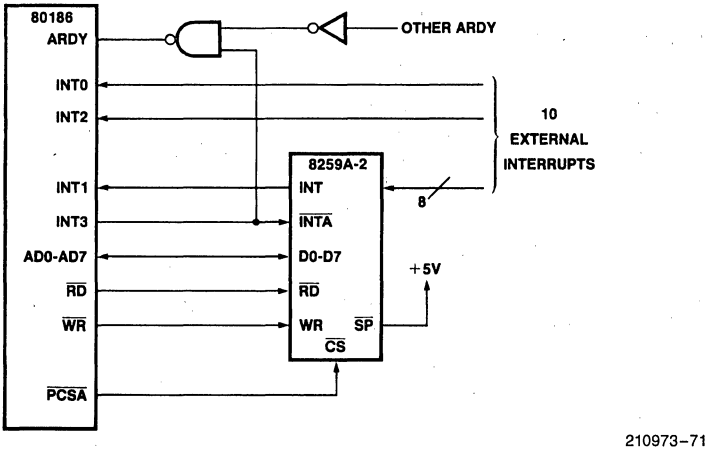


Figure 64. 80186/8259A Interrupt Cascading

time, which is the time from when the processor actually begins processing the interrupt to when it actually executes the first instruction of the interrupt service routine. The factors affecting interrupt latency are the intstruction being executed and the state of the interrupt enable flip-flop.

Interrupts will be acknowledged only if the interrupt enable flip-flop in the CPU is set. Thus, interrupt latency will be very long indeed if interrupts are never enabled by the processor!

When interrupts are enabled in the CPU, the interrupt latency is a function of the instructions being executed. Only repeated instructions will be interrupted before being completed, and those only between their respective iterations. This means that the interrupt latency time could be as long as 69 CPU clocks, which is the time it takes the processor to execute an integer divide instruction (with a segment override prefix, see below), the longest single instruction on the 80186.

Other factors can affect interrupt latency. An interrupt will not be accepted between the execution of a prefix (such as segment override prefixes and lock prefixes) and the instruction. In addition, an interrupt will not be accepted between an instruction which modifies any of the segment registers and the instruction immediately following the instruction. This is required to allow the stack to be changed. If the interrupt were accepted, the return address from the interrupt would be placed on a stack which was not valid (the Stack Segment register would have been modified but the Stack Pointer register would not have been). Finally, an interrupt will not be accepted between the execution of the WAIT instruction and the instruction immediately following it if the $\overline{\text{TEST}}$ input is active. If the WAIT sees the $\overline{\text{TEST}}$ input inactive, however, the interrupt will be accepted, and the WAIT will be re-executed after the interrupt return. This is required, since the WAIT is used to prevent execution by the 80186 of an 8087 instruction while the 8087 is busy.

7.0 CLOCK GENERATOR

The 80186 includes a clock generator which generates the main clock signal for all 80186 integrated components, and all CPU synchronous devices in the 80186 system. This clock generator includes a crystal oscillator, divide by two counter, reset circuitry, and ready generation logic. A block diagram of the clock generator is shown in Figure 65.

7.1 Crystal Oscillator

The 80186 crystal oscillator is a parallel resonant, Pierce oscillator. It was designed to be used as shown in Figure 66. The capacitor values shown are approximate. As the crystal frequency drops, they should be increased, so that at the 4 MHz minimum crystal frequency supported by the 80186 they take on a value of 30 pF. The output of this oscillator is not directly available outside the 80186.

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pF max
C_1 (Load Capacitance):	20 pF ±2 pF
Drive Level:	1 mW max

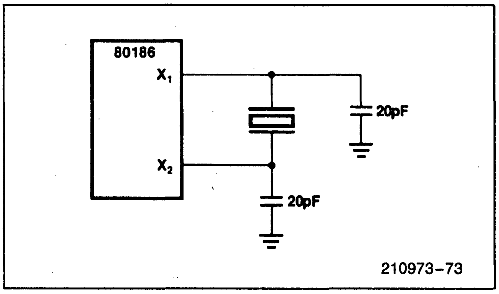


Figure 66. 80186 Crystal Connection

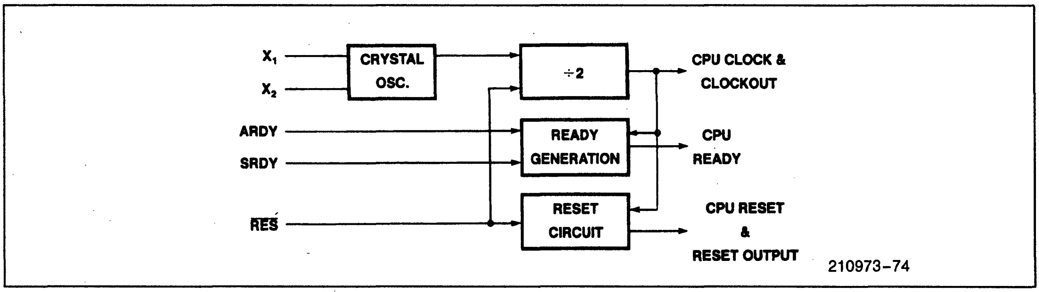


Figure 65. 80186 Clock Generator Block Diagram

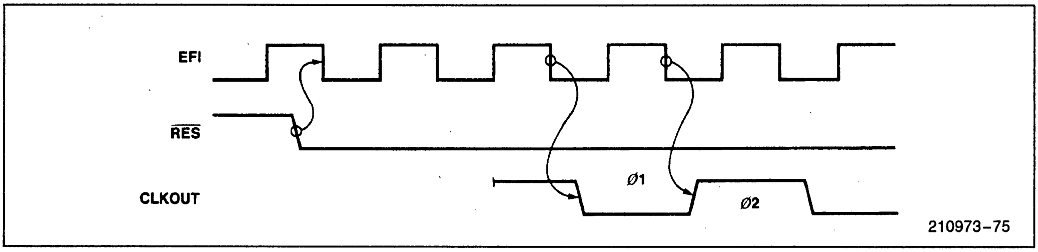


Figure 67. 80186 Clock Generator Reset

7.2 Using an External Oscillator

An external oscillator may be used with the 80186. The external frequency input (EFI) signal is connected directly to the X1 input of the oscillator. X2 should be left open. This oscillator input is used to drive an internal divide-by-two counter to generate the CPU clock signal, so the external frequency input can be of practically any duty cycle, so long as the minimum high and low times for the signal (as stated in the data sheet) are met.

7.3 Clock Generator

The output of the crystal oscillator (or the external frequency input) drives a divide by two circuit which generates a 50% duty cycle clock for the 80186 system. All 80186 timing is referenced to this signal, which is available on the CLKOUT pin of the 80186. This signal will change state on the high-to-low transition of the EFI signal.

7.4 Ready Generation

The clock generator also includes the circuitry required for ready generation. Interfacing to the SRDY and ARDY inputs this provides is covered in Section 3.1.6.

7.5 Reset

The 80186 clock generator also provides a synchronized reset signal for the system. This signal is generated from the reset input ($\overline{RES}$) to the 80186. The clock generator synchronizes this signal to the clockout signal.

The reset input signal also resets the divide-by-two counter. A one clock cycle internal clear pulse is generated when the $\overline{RES}$ input signal first goes active. This clear pulse goes active beginning on the first low-to-high transition of the X1 input after $\overline{RES}$ goes active, and goes inactive on the next low-to-high transition of the X1 input. In order to insure that the clear pulse is generated on the next EFI cycle, the $\overline{RES}$ input signal must satisfy a 25 ns setup time to the high-to-low EFI input signal (see Figure 67). During this clear, clockout will be high. On the next high-to-low transition of X1, clockout will go low, and will change state on every subsequent high-to-low transition of EFI.

The reset signal presented to the rest of the 80186, and also the signal present on the RESET output pin of the 80186 is synchronized by the high-to-low transition of the clockout signal of the 80186. This signal remains active as long as the $\overline{RES}$ input also remains active. After the $\overline{RES}$ input goes inactive, the 80186 will begin to fetch its first instruction (at memory location FFFF0H) after 6 1/2 CPU clock cycles (i.e., T_1 of the first instruction fetch will occur 6 1/2 clock cycles later). To insure that the RESET ouput will go inactive on the next CPU clock cycle, the inactive going edge of the $\overline{RES}$ input must satisfy certain hold and setup times to the low-to-high edge of the clockout signal of the 80186 (see Figure 68).

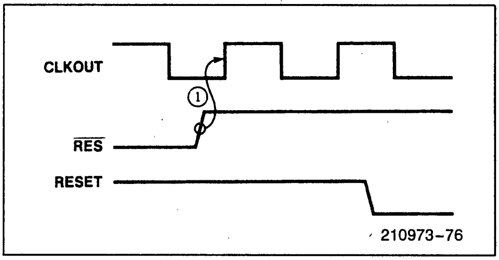


Figure 68. 80186 Coming out of Reset

8.0 CHIP SELECTS

The 80186 includes a chip select unit which generates hardware chip select signals for memory and I/O accesses generated by the 80186 CPU and DMA units. This unit is programmable such that it can be used to fulfill the chip select requirements (in terms of memory device or bank size and speed) of most small and medium sized 80186 systems.

The chip selects are driven only for internally generated bus cycles. Any cycles generated by an external unit (e.g., an external DMA controller) will not cause the chip selects to go active. Thus, any external bus masters must be responsible for their own chip select generation. Also, during a bus HOLD, the 80186 does not

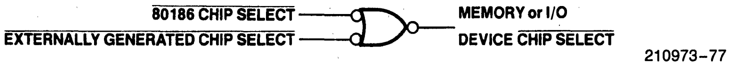

$\overline{\text{80186 CHIP SELECT}}$
$\overline{\text{EXTERNALLY GENERATED CHIP SELECT}}$
MEMORY or I/O
DEVICE $\overline{\text{CHIP SELECT}}$
210973-77

Figure 69. 80186/External Chip Select/Device Chip Select Generation

float the chip select lines. Therefore, logic must be included to enable the devices which the external bus master wishes to access (see Figure 69).

8.1 Memory Chip Selects

The 80186 provides six discrete chip select lines which are meant to be connected to memory components in an 80186 system. These signals are named $\overline{\text{UCS}}$, $\overline{\text{LCS}}$, and $\overline{\text{MCS0-3}}$ for Upper Memory Chip Select, Lower Memory Chip Select and Midrange Memory Chip Select 0–3. They are meant (but not limited) to be connected to the three major areas of the 80186 system memory (see Figure 70).

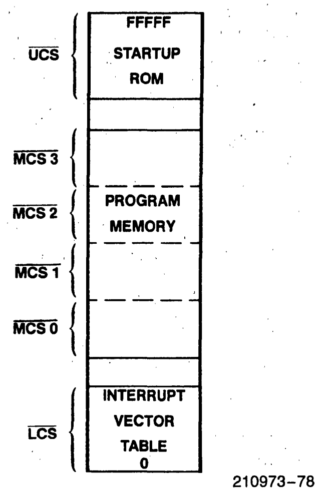


Figure 70. 80186 Memory Areas & Chip Selects

As could be guessed by their names, upper memory, lower memory, and mid-range memory chip selects are designed to address upper, lower, and middle areas of memory in an 80186 system. The upper limit of $\overline{\text{UCS}}$ and the lower limit of $\overline{\text{LCS}}$ are fixed at FFFFFH and 00000H in memory space, respectively. The other limit of these is set by the memory size programmed into the control register for the chip select line. Mid-range memory allows both the base address and the block size of the memory area to be programmed. The only limitation is that the base address must be programmed to be an integer multiple of the total block size. For example, if the block size was 128K bytes (4 32K byte chunks) the base address could be 0 or 20000H, but not 10000H.

The memory chip selects are controlled by 4 registers in the peripheral control block (see Figure 71). These include 1 each for $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$, the values of which determine the size of the memory blocks addressed by these two lines. The other two registers are used to control the size and base address of the mid-range memory block.

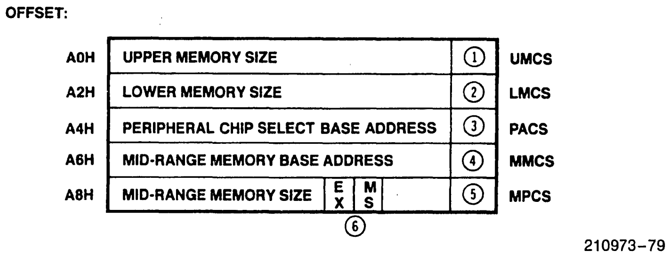
On reset, only $\overline{\text{UCS}}$ is active. It is programmed by reset to be active for the top 1K memory block, to insert 3 wait states to all memory fetches, and to factor external ready for every memory fetch (see Section 8.3 for more information on internal ready generation). All other chip select registers assume indeterminate states after reset, but none of the other chip select lines will be active until all necessary registers for a signal have been accessed (not necessarily written, a read to an uninitialized register will enable the chip select function controlled by that register).

8.2 Peripheral Chip Selects

The 80186 provides seven discrete chip select lines which are meant to be connected to peripheral components in an 80186 system. These signals are named $\overline{\text{PCS0-6}}$. Each of these lines is active for one of seven continuous 128 byte areas in memory or I/O space above a programmed base address.

The peripheral chip selects are controlled by two registers in the internal peripheral control block (see Figure 71). These registers allow the base address of the peripherals to be set, and allow the peripherals to be mapped into memory or I/O space. Both of these registers must be accessed before any of the peripheral chip selects will become active.

A bit in the MPCS register allows $\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$ to become latched A1 and A2 outputs. When this option is selected, $\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$ will reflect the state of A1 and A2 throughout a bus cycle. These are provided to allow external peripheral register selection in a system in which the addresses are not latched. Upon reset, these lines are driven high. They will only reflect A1 and A2 after both PACS and MPCS have been accesssed (and are programmed to provide A1 and A2!).



NOTES:
1. Upper memory ready bits
2. Lower memory ready bits
3. PCS0-PCS3 ready bits
4. Mid-range memory ready bits
5. PCS4-PCS6 ready bits
6. MS: 1 = Peripherals active in memory space
   0 = Peripherals active in I/O space
   EX:1 = 7 PCS lines
   0 = PCS5 = A1, PCS6 = A2

Not all bits of every field are used

Figure 71. 80186 Chip Select Control Registers

8.3 Ready Generation

The 80186 includes a ready generation unit. This unit generates an internal ready signal for all accesses to memory or I/O areas to which the chip select circuitry of the 80186 responds.

For each ready generation area, 0–3 wait states may be inserted by the internal unit. Table 6 shows how the ready control bits should be programmed to provide this. In addition, the ready generation circuit may be programmed to ignore the state of the external ready (i.e., only the internal ready circuit will be used) or to factor the state of the external ready (i.e., a ready will be returned to the processor only after both the internal ready circuit has gone ready and the external ready has gone ready). Some kind of circuit must be included to generate an external ready, however, since upon reset the ready generator is programmed to factor external ready to all accesses to the top 1K byte memory block. If a ready was not returned on one of the external ready lines (ARDY or SRDY) the processor would wait forever to fetch its first instruction.

Table 6. 80186 Wait State Programming

R2	R1	R0	Number of Wait States
0	0	0	0 + external ready
0	0	1	1 + external ready
0	1	0	2 + external ready
0	1	1	3 + external ready
1	0	0	0 (no external ready required)
1	0	1	1 (no external ready required)
1	1	0	2 (no external ready required)
1	1	1	3 (no external ready required)

8.4 Examples of Chip Select Usage

Many examples of the use of the chip select lines are given in the bus interface section of this note (Section 3.2). These examples show how simple it is to use the chip select function provided by the 80186. The key point to remember when using the chip select function is that they are only activated during bus cycles generated by the 80186 CPU or DMA units. When another master has the bus, it must generate its own chip select function. In addition, whenever the bus is given by the 80186 to an external master (through the HOLD/HLDA arrangement) the 80186 does NOT float the chip select lines.

8.5 Overlapping Chip Select Areas

Generally, the chip selects of the 80186 should not be programmed such that any two areas overlap. In addition, none of the programmed chip select areas should overlap any of the locations of the integrated 256-byte control register block. The consequences of doing this are:

Whenever two chip select lines are programmed to respond to the same area, both will be activated during any access to that area. When this is done, the ready bits for both areas *must* be programmed to the same value. If this is not done, the processor response to an access in this area is indeterminate. This rule also applies to overlapping chip selects with the integrated control block.

If any of the chip select areas overlap the integrated 256-byte control block, the timing on the chip select line is altered. An access to the control block will temporarily activate the corresponding chip select pin, but it will go inactive prematurely.

9.0 SOFTWARE IN AN 80186 SYSTEM

Since the 80186 is object code compatible with the 8086 and 8088, the software in an 80186 system is very similar to that in an 8086 system. Because of the hardware chip select functions, however, a certain amount of initialization code must be included when using those functions on the 80186.

9.1 System Initialization in an 80186 System

Most programmable components of a computer system must be initialized before they are used. This is also true for the 80186. The 80186 includes circuitry which directly affects the ability of the system to address memory and I/O devices, namely the chip select circuitry. This circuitry must be initialized before the memory areas and peripheral devices addressed by the chip select signals are used.

Upon reset, the UMCS register is programmed to be active for all memory fetches within the top 1K byte of memory space. It is also programmed to insert three wait states to all memory accesses within this space. If the hardware chip selects are used, they must be programmed before the processor leaves this 1K byte area of memory. If a jump to an area for which the chips are not selected occurs, the microcomputer system will cease to operate (since the processor will fetch garbage from the data bus). Appendix F shows a typical initialization sequence for the 80186 chip select unit.

Once the chip selects have been properly initialized, the rest of the 80186 system may be initialized much like an 8086 system. For example, the interrupt vector table might get set up, the interrupt controller initialized, a serial I/O channel initialized, and the main program begun. Note that the integrated peripherals included in the 80186 do not share the same programming model as the standard Intel peripherals used to implement these functions in a typical 8086 system, i.e. different values must be programmed into different registers to achieve the same function using the integrated peripherals. Appendix F shows a typical initialization sequence for an interrupt driven system using the 80186 interrupt controller.

9.2 Instruction Execution Differences between the 8086 and 80186

There are a few instruction execution differences between the 8086 and the 80186. These differences are:

UNDEFINED OPCODES:

When the opcodes 63H, 64H, 65H, 66H, 67H, F1H, FEH XX111XXXB and FFH XX111XXXB are executed, the 80186 will execute an illegal instruction exception, interrupt type 6. The 8086 will ignore the opcode.

0FH OPCODE:

When the opcode 0FH is encountered, the 8086 will execute a POP CS, while the 80186 will excecute an illegal instruction exception, interrupt type 6.

WORD WRITE AT OFFSET FFFFH:

When a word write is performed at offset FFFFH in a segment, the 8086 will write one byte at offset FFFFH, and the other at offset 0, while the 80186 will write one byte at offset FFFFH, and the other at offset 10000H (one byte beyond the end of the segment). One byte segment underflow will also occur (on the 80186) if a stack PUSH is executed and the Stack Pointer contains the value 1.

SHIFT/ROTATE BY VALUE GREATER THAN 31:

Before the 80186 performs a shift or rotate by a value (either in the CL register, or by an immediate value) it ANDs the value with 1FH, limiting the number of bits rotated to less than 32. The 8086 does not do this.

LOCK PREFIX:

The 8086 activates its LOCK signal immediately after executing the LOCK prefix. The 80186 does not activate the LOCK signal until the processor is ready to begin the data cycles associated with the LOCKed instruction.

NOTE:
When executing more than one LOCKed instruction, always make sure there are 6 bytes of code between the end of the first LOCKed instruction and the start of the second LOCKed instruction.

INTERRUPTED STRING MOVE INSTRUCTIONS:

If an 8086 is interrupted during the execution of a repeated string move instruction, the return value it will push on the stack will point to the last prefix instruction before the string move instruction. If the instruction had more than one prefix (e.g., a segment override prefix in addition to the repeat prefix), it will not be reexecuted upon returning from the interrupt. The 80186 will push the value of the first prefix to the repeated instruction, so long as prefixes are not repeated, allowing the string instruction to properly resume.

CONDITIONS CAUSING DIVIDE ERROR WITH AN INTEGER DIVIDE:

The 8086 will cause a divide error whenever the absolute value of the quotient is greater than 7FFFH (for word operations) or if the absolute value of the quotient is greater than 7FH (for byte operations). The 80186 has expanded the range of negative numbers allowed as a quotient by 1 to include 8000H and 80H. These numbers represent the most negative numbers representable using 2's complement arithmetic (equaling −32768 and −128 in decimal, respectively).

ESC OPCODE:

The 80186 may be programmed to cause an interrupt type 7 whenever an ESCape instruction (used for coprocessors like the 8087) is executed. The 8086 has no such provision. Before the 80186 performs this trap, it must be programmed to do so.

These differences can be used to determine whether the program is being executed on an 8086 or an 80186. Probably the safest execution difference to use for this purpose is the difference in multiple bit shifts. For example, if a multiple bit shift is programmed where the shift count (stored in the CL register!) is 33, the 8086 will shift the value 33 bits, whereas the 80186 will shift it only a single bit.

In addition to the instruction execution differences noted above, the 80186 includes a number of new instruction types, which simplify assembly language programming of the processor, and enhance the performance of higher level languages running on the processor. These new instructions are covered in depth in the 8086/80186 users manual and in Appendix H of this note.

10.0 CONCLUSIONS

The 80186 is a glittering example of state-of-the-art integrated circuit technology applied to make the job of the microprocessor system designer simpler and faster. Because many of the required peripherals and their interfaces have been cast in silicon, and because of the timing and drive latitudes provided by the part, the designer is free to concentrate on other issues of system design. As a result, systems designed around the 80186 allow applications where no other processor has been able to provide the necessary performance at a comparable size or cost.

APPENDIX A PERIPHERAL CONTROL BLOCK

All the integrated peripherals within the 80186 microprocessor are controlled by sets of registers contained within an integrated peripheral control block. The registers are physically located within the peripheral devices they control, but are addressed as a single block of registers. This set of registers encompasses 256 contiguous bytes and can be located on any 256 byte boundary of the 80186 memory or I/O space. A map of these registers is shown in Figure A-1; any unused bytes are reserved.

A.1 SETTING THE BASE LOCATION OF THE PERIPHERAL CONTROL BLOCK

In addition to the control registers for each of the integrated 80186 peripheral devices, the peripheral control block contains the peripheral control block relocation register. This register allows the peripheral control block to be re-located on any 256 byte boundary within the processor's memory or I/O space. Figure A-2 shows the layout of this register.

This register is located at offset FEH within the peripheral control block. Since it is itself contained within the peripheral control block, any time the location of the peripheral control block is moved, the location of the relocation registers will also move.

In addition to the peripheral control block relocation information, the relocation register contains two additional bits. One is used to set the interrupt controller into slave mode. The other is used to force the processor to trap whenever an ESCape (coprocessor) instruction is encountered.

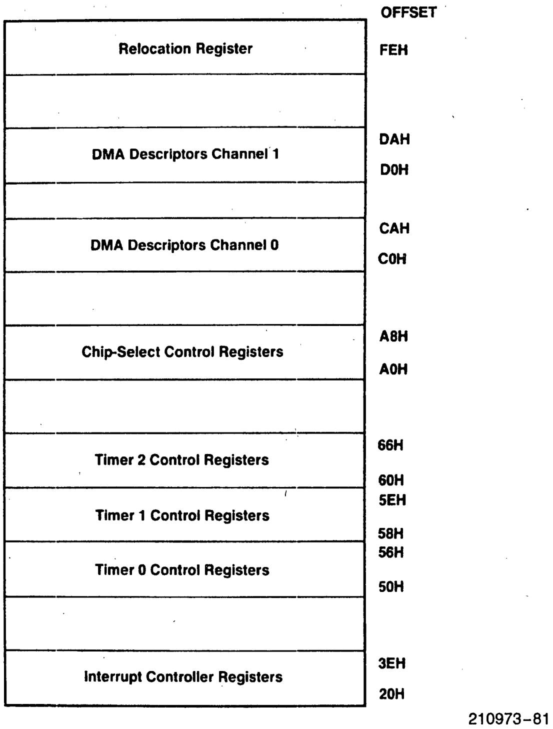
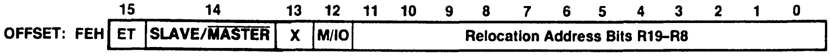

OFFSET
Relocation Register
FEH
DMA Descriptors Channel 1
DAH
D0H
DMA Descriptors Channel 0
CAH
C0H
Chip-Select Control Registers
A8H
A0H
Timer 2 Control Registers
66H
60H
Timer 1 Control Registers
5EH
58H
Timer 0 Control Registers
56H
50H
Interrupt Controller Registers
3EH
20H
210973-81

Figure A-1. 80186 Integrated Peripheral Control Block



OFFSET: FEH	15	14	13	12	11–0
	ET	SLAVE/$\overline{\text{MASTER}}$	X	M/IO	Relocation Address Bits R19–R8

210973-82

NOTES:
ET = ESC Trap / No ESC Trap (1/0)
M/IO = Register Block Located in Memory / I/O Space (1/0)
SLAVE/$\overline{\text{MASTER}}$ = Master Interrupt Controller Mode / Slave Interrupt Controller Mode (0/1)

Figure A-2. 80186 Relocation Register Layout

Because the relocation register is contained within the peripheral control block, upon reset the relocation register is automatically programmed with the value 20 FFH. This means that the peripheral control block will be located at the very top (FF00H to FFFFH) of I/O space. Thus, after reset the relocation register will be located at word location FFFEH in I/O space.

If the user wished to locate the peripheral control block starting at memory location 10000H he would program the peripheral control register with the value 1100H. By doing this, he would move all registers within the integrated peripheral control block to memory locations 10000H to 100FFH. Note that since the relocation register is contained within the peripheral control block, it too would move to word location 100FEH in memory space.

Whenever mapping the 188 peripheral control block to another location, the programming of the relocation register should be done with a byte write (i.e., OUT DX,AL). Any access to the control block is done 16 bits at a time. Thus, internally, the relocation register will get written with 16 bits of the AX register while externally, the BIU will run only one 8 bit bus cycle. If a word instruction is used (i.e., OUT DX,AX), the relocation register will be written on the first bus cycle. The BIU will then run a second bus cycle which is unnecessary. The address of the second bus cycle will no longer be within the control block (i.e., the control block was moved on the first cycle), and therefore, will require the generation of an external ready signal to complete the cycle. For this reason we recommend byte operations to the relocation register. Byte instructions may also be used for the other registers in the control block and will eliminate half of the bus cycles required if a word operation had been specified. Byte operations are only valid on even addresses though, and are undefined on odd addresses.

A.2 Peripheral Control Block Registers

Each of the integrated peripherals' control and status registers are located at a fixed location above the programmed base location of the peripheral control block. There are many locations within the peripheral control block which are not assigned to any peripheral. If a write is made to any of these locations, the bus cycle will be run, but the value will not be stored in any internal location. This means that if a subsequent read is made to the same location, the value written will not be read back.

The processor will run an external bus cycle for any memory or I/O cycle which accesses a location within the integrated control block. This means that the address, data, and control information will be driven on the 80186 external pins just as if a "normal" bus cycle had been run. Any information returned by an external device will be ignored, however, even if the access was to a location which does not correspond to any of the integrated peripheral control registers. The above is also true for the 80188, except that the word access made to the integrated registers will be performed in a single bus cycle internally, while externally, the BIU runs two bus cycles.

The processor internally generates a ready signal whenever any of the integrated peripherals are accessed; thus any external ready signals are ignored whenever an access is made to any location within the integrated peripheral register control block. This ready will also be returned if an access is made to a location within the 256 byte area of the peripheral control block which does not correspond to any integrated peripheral control register. The processor will insert 0 wait states to any access within the integrated peripheral control block except for accesses to the timer registers. ANY access to the timer control and counting registers will incur 1 wait state. This wait state is required to properly multiplex processor and counter element accesses to the timer control registers.

All accesses made to the integrated peripheral control block will be WORD accesses. Any write to the integrated registers will modify all 16 bits of the register, whether the opcode specified a byte write or a word write. A byte read from an even location should cause no problems, but the data returned when a byte read is performed from an odd address within the peripheral control block is undefined. This is true both for the 80186 AND the 80188. As stated above, even though the 80188 has an external 8 bit data bus, internally it is still a 16 bit machine. Thus, the word accesses performed to the integrated registers by the 80188 will each occur in a single bus cycle internally while externally the BIU runs two bus cycles. The DMA controller must not be used for either read or write accesses to the peripheral control block.

APPENDIX B
80186 SYNCHRONIZATION INFORMATION

Many input signals to the 80186 are asynchronous, that is, a specified set up or hold time is not required to insure proper functioning of the device. Associated with each of these inputs is a synchronizer which samples this external asynchronous signal, and synchronizes it to the internal 80186 clock.

B.1 WHY SYNCHRONIZERS ARE REQUIRED

Every data latch requires a certain set up and hold time in order to operate properly. At a certain window within the specified set up and hold time, the part will actually try to latch the data. If the input makes a transition within this window, the output will not attain a stable state within the given output delay time. The size of this sampling window is typically much smaller than the actual window specified by the data sheet, however part to part variation could move this window around within the specified window in the data sheet.

Even if the input to a data latch makes a transition while a data latch is attempting to latch this input, the output of the latch will attain a stable state after a certain amount of time, typically much longer than the normal strobe to output delay time. Figure B-1 shows a normal input to output strobed transition and one in which the input signal makes a transition during the latch's sample window. In order to synchronize an asynchronous signal, all one needs to do is to sample the signal into one data latch, wait a certain amount of time, then latch it into a second data latch. Since the time between the strobe into the first data latch and the strobe into the second data latch allows the first data latch to attain a steady state (or to resolve the asynchronous signal), the second data latch will be presented with an input signal which satisfies any set up and hold time requirements it may have.

Thus, the output of this second latch is a synchronous signal with respect to its strobe input.

A synchronization failure can occur if the synchronizer fails to resolve the asynchronous transition within the time between the two latch's strobe signals. The rate of failure is determined by the actual size of the sampling window of the data latch, and by the amount of time between the strobe signals of the two latches. Obviously, as the sampling window gets smaller, the number of times an asynchronous transition will occur during the sampling window will drop. In addition, however, a smaller sampling window is also indicative of a faster resolution time for an input transition which manages to fall within the sampling window.

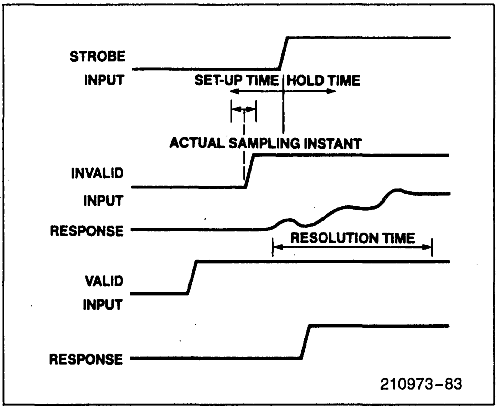


Figure B-1. Valid and Invalid Latch Input Transitions and Responses

B.2 80186 SYNCHRONIZERS

The 80186 contains synchronizers on the $\overline{\text{RES}}$, $\overline{\text{TEST}}$, TmrIn0-1, DRQ0-1, NMI, INT0-3, ARDY, and HOLD input lines. Each of these synchronizers use the two stage synchronization technique described above (with some minor modifications for the ARDY line, see section 3.1.6). The sampling window of the latches is designed to be in the tens of pico-seconds, and should allow operation of the synchronizers with a mean time between failures of over 30 years assuming continuous operation.

APPENDIX C
80186 EXAMPLE DMA INTERFACE CODE

```
$mod186
name                        assembly_example_80186_DMA_support
;
; This file contains an example procedure which initializes the 80186 DMA
;       controller to perform the DMA transfers between the 80186 system and the
;       8272 Floppy Disk Controller (FDC). It assumes that the 80186
;       peripheral control block has not been moved from its reset location.
;
arg1                    equu    word ptr [BP + 4]
arg2                    equ     word ptr [BP + 6]
arg3                    equ     word ptr [BP + 8]
DMA_FROM_LOWER          equ     0FFC0h              ; DMA register locations
DMA_FROM_UPPER          equ     0FFC2h
DMA_TO_LOWER            equ     0FFC4h
DMA_TO_UPPER            equ     0FFC6h
DMA_COUNT               equ     0FFC8h
DMA_CONTROL             equ     0FFCAh
DMA_TO_DISK_CONTROL     equ     01486h              ; destination synchronization
                                                    ; source to memory, incremented
                                                    ; destination to I/O
                                                    ; no terminal count
                                                    ; byte transfers
DMA_FROM_DISK_CONTROLequ        0A046h              ; source synchronization
                                                    ; source to I/O
                                                    ; destination to memory, incr
                                                    ; no terminal count
                                                    ; byte transfers
FDC_DMA                 equ     6B8h                ; FDC DMA address
FDC_DATA                equ     688h                ; FDC data register
FDC_STATUS              equ     680h                ; FDC status register

cgroup                  group   code
code                    segment                     public 'code'
                        public  set_dma_
                        assume  cs:cgroup
;
; set_dma (offset,to) programs the DMA channel to point one side to the
;       disk DMA address, and the other to memory pointed to by ds:offset. If
;       'to' = 0 then will be a transfer from disk to memory; if
;       'to' = 1 then will be a transfer from memory to disk. The parameters to
;       the routine are passed on the stack.
;
set_dma_                proc    near
                        enter   0,0                 ; set stack addressability
                        push    AX                  ; save registers used
                        push    BX
                        push    DX
                        test    arg2,1              ; check to see direction of
                                                    ; transfer
                        jz      from_disk
; performing a transfer from memory to the disk controller
;
                        mov     AX,DS               ; get the segment value
                        rol     AX,4                ; gen the upper 4 bits of the
                                                    ; physical address in the lower 4
                                                    ; bits of the register
```

210973-84

```
                        mov     BX,AX                   ; save the result...
                        mov     DX,DMA_FROM_UPPER       ; prgm the upper 4 bits of the
                        out     DX,AX                   ; DMA source register
                        and     AX,0FFF0h               ; form the lower 16 bits of the
                                                        ; physical address
                        add     AX,arg1                 ; add the offset
                        mov     DX,DMA_FROM_LOWER       ; prgm the lower 16 bits of the
                        out     DX,AX                   ; DMA source register
                        jnc     no_carry_from           ; check for carry out of addition
                        inc     BX                      ; if carry out, then need to adj
                        mov     AX,BX                   ; the upper 4 bits of the pointer
                        mov     DX,DMA_FROM_UPPER
                        out     DX,AX
no_carry_from:
                        mov     AX,FDC_DMA              ; prgm the low 16 bits of the DMA
                        mov     DX,DMA_TO_LOWER         ; destination register
                        out     DX,AX
                        xor     AX,AX                   ; zero the up 4 bits of the DMA
                        mov     DX,DMA_TO_UPPER         ; destination register
                        out     DX,AX
                        mov     AX,DMA_TO_DISK_CONTROL; prgm the DMA ctl reg
                        mov     DX,DMA_CONTROL          ; note: DMA may begin immediatly
                        out     DX,AX                   ; after this word is output
                        pop     DX
                        pop     BX
                        pop     AX
                        leave
                        ret
from_disk:
;
;   performing a transfer from the disk to memory
;
                        mov     AX,DS
                        rol     AX,4
                        mov     DX,DMA_TO_UPPER
                        out     DX,AX
                        mov     BX,AX
                        and     AX,0FFF0h
                        add     AX,arg1
                        mov     DX,DMA_TO_LOWER
                        out     DX,AX
                        jnc     no_carry_to
                        inc     BX
                        mov     AX,BX
                        mov     DX,DMA_TO_UPPER
                        out     DX,AX
no_carry_to:
                        mov     AX,FDC_DMA

                        mov     DX,DMA_FROM_LOWER
                        out     DX,AX
                        xor     AX,AX
                        mov     DX,DMA_FROM_UPPER
                        out     DX,AX
                        mov     AX,DMA_FROM_DISK_CONTROL
                        mov     DX,DMA_CONTROL
```

210973-85

```
                        out     DX,AX
                        pop     DX
                        pop     BX
                        pop     AX
                        leave
                        ret
set_dma_                endp

code                    ends
                        end
```

210973-86

APPENDIX D
80186 EXAMPLE TIMER INTERFACE CODE

```
$mod186
name                        example_80186_timer_code
;
;  this file contains example 80186 timer routines. The first routine
;       sets up the timer and interrupt controller to cause the timer
;       to generate an interrupt every 10 milliseconds, and to service
;       interrupt to implement a real time clock. Timer 2 is used in
;       this example because no input or output signals are required.
;       The code example assumes that the peripheral control block has
;       not been moved from its reset location (FF00-FFFF in I/O space).
;
arg1                equu    word ptr [BP + 4]
arg2                equ     word ptr [BP + 6]
arg3                equ     word ptr [BP + 8]
timer_2int          equ     19                          ; timer 2 has vector type 19
timer_2control      equ     0FF66h
timer_2max_ctl      equ     0FF62h
timer_int_ctl       equ     0FF32h                      ; interrupt controller regs
eoi_register        equ     0FF22h
interrupt_stat      equ     0FF30h

data                segment                             public 'data'
                    public  hour_,minute_,second_,msec_
msec_               db      ?
hour_               db      ?
minute_             db      ?
second_             db      ?
data                ends

cgroup              group   code
dgroup              group   data

code                segment                             public 'code'
                    public  set_time_
                    assume  cs:code,ds:dgroup
;
;  set_time(hour,minute,second) sets the time variables, initializes the
;       80186 timer2 to provide interrupts every 10 milliseconds, and
;       programs the interrupt vector for timer 2
;
set_time_           proc    near
                    enter   0,0                         ; set stack addressability
                    push    AX                          ; save registers used
                    push    DX
                    push    SI
                    push    DS

                    xor     AX,AX                       ; set the interrupt vector
                                                        ; the timers have unique
                                                        ; interrupt
                                                        ; vectors even though they share
                                                        ; the same control register
                    mov     DS,AX

                    mov     SI,4 * timer2_int
```

210973-87

```
                        mov     word ptr DS:[SI],offset timer_2_interrupt_routine
                        inc     SI
                        inc     SI
                        mov     DS:[SI],CS
                        pop     DS

                        mov     AX,arg1                 ; set the time values
                        mov     hour_,AL
                        mov     AX,arg2
                        mov     minute_,AL
                        mov     AX,arg3
                        mov     second_,AL
                        mov     msec_,0

                        mov     DX,timer2_max_ctl       ; set the max count value
                        mov     AX,20000                ; 10 ms / 500 ns (timer 2 counts
                                                        ; at 1/4 the CPU clock rate)
                        out     DX,AX
                        mov     DX,timer2_control       ; set the control word
                        mov     AX,1110000000000001b    ; enable counting
                                                        ; generate interrupts on TC
                                                        ; continuous counting
                        out     DX,AX

                        mov     DX,timer_int_ctl        ; set up the interrupt controller
                        mov     AX,0000b                ; unmask interrupts
                                                        ; highest priority interrupt
                        out     DX,AX
                        sti                             ; enable processor interrupts

                        pop     SI
                        pop     DX
                        pop     AX
                        leave
                        ret
set_time_               endp

timer2_interrupt_routine proc   far
                        push    AX
                        push    DX

                        cmp     msec_,99                ; see if one second has passed
                        jae     bump_second             ; if above or equal...
                        inc     msec_
                        jmp     reset_int_ctl
bump_second:
                        mov     msec_,0                 ; reset millisecond
                        cmp     second_,59              ; see if one minute has passed
                        jae     bump_minute
                        inc     second_
                        jmp     reset_int_ctl
bump_minute:
                        mov     second_,0
                        cmp     minute_,59              ; see if one hour has passed
                        jae     bump_hour
                        inc     minute_
                        jmp     reset_int_ctl                                   210973-88
                        pop     DX
                        pop     AX
                        ret
timer2_interrupt_routine endp
code                    ends
                        end
                                210973-89
```

```
bump_hour:
                                mov     minute_,0
                                cmp     hour_,12                ; see if 12 hours have passed
                                jae     reset_hour
                                inc     hour_
                                jmp     reset_int_ctl
reset_hour:
                                mov     hour_,1
reset_int_ctl:

                                mov     DX,eoi_register
                                mov     AX,8000h                ; non-specific end of interrupt
                                out     DX,AX

                                pop     DX
                                pop     AX
                                iret
timer2_interrupt_routine        endp
code                            ends
                                end
$mod186
name                            example_80186_baud_code
;
; this file contains example 80186 timer routines. The second routine
;       sets up the timer as a baud rate generator. In this mode,
;       Timer 1 is used to continually output pulses with a period of
;       6.5 usec for use with a serial controller at 9600 baud
;       programmed in divide by 16 mode (the actual period required
;       for 9600 baud is 6.51 usec). This assumes that the 80186 is
;       running at 8 MHz. The code example also assumes that the
;       peripheral control block has not been moved from its reset
;       location (FF00-FFFF in I/O space).
;
timer1_control                  equ     0FF5Eh
timer1_max_cnt                  equ     0FF5Ah

code                            segment                         public 'code'
                                assume  cs:code
;
; set_baud() initializes the 80186 timer1 as a baud rate generator for
;       a serial port running at 9600 baud
;
set_baud_                       proc    near
                                push    AX                      ; save registers used
                                push    DX

                                mov     DX,timer1_max_cnt       ; set the max count value
                                mov     AX,13                   ; 500ns * 13 = 6.5 usec
                                out     DX,AX
                                mov     DX,timer1_control       ; set the control word
                                mov     AX,1100000000000001b    ; enable counting
                                                                ; no interrupt on TC
                                                                ; continuous counting
                                                                ; single max count register
                                out     DX,AX

                                pop     DX
                                pop     AX
```

210973-90

```
                              ret
set_baud_                     endp
code                          ends
                              end
$mod186
name                          example_80186_count_code
;
;  this file contains example 80186 timer routines. The third routine
;          sets up the timer as an external event counter. In this mode,
;          Timer 1 is used to count transitions on its input pin. After
;          the timer has been set up by the routine, the number of
;          events counted can be directly read from the timer count
;          register at location FF58H in I/O space. The timer will
;          count a maximum of 65535 timer events before wrapping
;          around to zero. This code example also assumes that the
;          peripheral control block has not been moved from its reset
;          location (FF00-FFFF in I/O space).
;
timer1_control                equ     0FF5Eh
timer1_max_cnt                equ     0FF5Ah
timer1_cnt_reg                equ     0FF58H

code                          segment                       public 'code'
                              assume  cs:code
;
;  set_count() initializes the 80186 timer1 as an event counter
;
set_count_                    proc    near
                              push    AX                    ;  save registers used
                              push    DX

                              mov     DX,timer1_max_cnt     ;  set the max count value
                              mov     AX,0                  ;  allows the timer to count
                                                            ;  all the way to FFFFH
                              out     DX,AX
                              mov     DX,timer1_control     ;  set the control word
                              mov     AX,1100000000000101b  ;  enable counting
                                                            ;  no interrupt on TC
                                                            ;  continuous counting
                                                            ;  single max count register
                                                            ;  external clocking
                              out     DX,AX

                              xor     AX,AX                 ;  zero AX
                              mov     DX,timer1_cnt_reg     ;  and zero the count in the timer
                              out     DX,AX                 ;  count register

                              pop     DX
                              pop     AX
                              ret

set_count_                    endp
code                          ends
                              end
```

210973-91

APPENDIX E
80186 EXAMPLE INTERRUPT CONTROLLER INTERFACE CODE

```
$mod186
name                    example_80186_interrupt_code
;
; This routine configures the 80186 interrupt controller to provide
;       two cascaded interrupt inputs (through an external 8259A
;       interrupt controller on pins INT0/INT2) and two direct
;       interrupt inputs (on pins INT1 and INT3). The default priority
;       levels are used. Because of this, the priority level programmed
;       into the control register is set the 111, the level all
;       interrupts are programmed to at reset.
;
int0_control            equ     0FF38H
int_mask                equ     0FF28H
;
code                    segment                          public 'code'
                        assume  CS:code
set_int_                proc    near
                        push    DX
                        push    AX

                        mov     AX,0100111B              ; cascade mode
                                                         ; interrupt unmasked
                        mov     DX,int0_control
                        out     DX,AX

                        mov     AX,01001101B             ; now unmask the other external
                                                         ; interrupts
                        mov     DX,int_mask
                        out     DX,AX
                        pop     AX
                        pop     DX
                        ret
set_int_                endp
code                    ends
                        end
$mod186
name                    example_80186_interrupt_code
;
; This routine configures the 80186 interrupt controller into slave
;       mode. This code does not initialize any of the 80186
;       integrated peripheral control registers, nor does it initialize
;       the external 8259A interrupt controller.
;
relocation_reg          equ     0FFFEH
;
code                    segment                          public 'code'
                        assume  CS:code
set_rmx_                proc    near
                        push    DX
                        push    AX

                        mov     DX,relocation_reg
                        in      AX,DX                    ; read old contents of register
                        or      AX,0100000000000000B     ; set the Slave/Master mode bit
                        out     DX,AX
```

210973-92

APPENDIX F
80186/8086 EXAMPLE SYSTEM INITIALIZATION CODE

```
name                    example_80186_system_init
;
;  This file contains a system initialization routine for the 80186
;       or the 8086. The code determines whether it is running on
;       an 80186 or an 8086, and if it is running on an 80186, it
;       initializes the integrated chip select registers.
;
restart                 segment  at                   OFFFFh
;
;  This is the processor reset address at OFFFFOH
;
                        org      0
                        jmp      far ptr initialize
restart                 ends
;
                        extrn    monitor:far
init_hw                 segment  at                   OFFF0h
                        assume   CS:init_hw
;
;  This segment initializes the chip selects. It must be located in the
;       top 1K to insure that the ROM remains selected in the 80186
;       system until the proper size of the select area can be programmed.
;
UMCS_reg                equu     OFFA0H               ;  chip select register locations
LMCS_reg                equ      OFFA2H
PACS_reg                equ      OFFA4H
MPCS_reg                equ      OFFA8H
UMCS_value              equ      OF038H               ;  64K, no wait states
LMCS_value              equ      07F8H                ;  32K, no wait states
PACS_value              equ      007EH                ;  peripheral base at 400H, 2 ws
MPCS_value              equ      81B8H                ;  PCS5 and 6 supplies,
                                                      ;  peripherals in I/O space
initialize              proc     far
                        mov      AX,2                 ;  determine if this is an
                        mov      CL,33                ;  8086 or an 80186 (checks
                        shr      AX,CL                ;  to see if the multiple bit
                        test     AX,1                 ;  shift value was ANDed)
                        jz       not_80186

                        mov      DX,UMCS_reg          ;  program the UMCS register
                        mov      AX,UMCS_value
                        out      DX,AX

                        mov      DX,LMCS_reg          ;  program the LMCS register
                        mov      AX,LMCS_value
                        out      DX,AX

                        mov      DX,PACS_reg          ;  set up the peripheral chip
                                                      ;  selects (note the mid-range
                                                      ;  memory chip selects are not
                                                      ;  needed in this system, and
                                                      ;  are thus not initialized

                        mov      AX,PACS_value
                        out      DX,AX
                        mov      DX,MPCS_reg
                        mov      AX,MPCS_value
                        out      DX,AX
;
;  Now that the chip selects are all set up, the main program of the
;       computer may be executed.
;
not_80186:
                        jmp      far ptr monitor
initialize              endp
init_hw                 ends
                        end
```

210973-93

210973-94

APPENDIX G
80186 WAIT STATE PERFORMANCE

Because the 80186 contains separate bus interface and execution units, the actual performance of the processor will not degrade at a constant rate as wait states are added to the memory cycle time from the processor. The actual rate of performance degradation will depend on the type and mix of instructions actually encountered in the user's program.

Shown below are two 80186 assembly language programs, and the actual execution time for the two programs as wait states are added to the memory system of the processor. These programs show the two extremes to which wait states will or will not affect system performance as wait states are introduced.

Program 1 is very memory intensive. It performs many memory reads and writes using the more extensive memory addressing modes of the processor (which also take a greater number of bytes in the opcode for the instruction). As a result, the execution unit must constantly wait for the bus interface unit to fetch and perform the memory cycles to allow it to continue. Thus, the execution time of this type of routine will grow quickly as wait states are added, since the execution time is almost totally limited to the speed at which the processor can run bus cycles.

Note also that this program execution time calculated by merely summing up the number of clock cycles given in the data sheet will typically be less than the actual number of clock cycles actually required to run the program. This is because the numbers quoted in the data sheet assume that the opcode bytes have been prefetched and reside in the 80186 prefetch queue for immediate access by the execution unit. If the execution unit cannot access the opcode bytes immediately upon request, dead clock cycles will be inserted in which the execution unit will remain idle, thus increasing the number of clock cycles required to complete execution of the program.

On the other hand, program 2 is more CPU intensive. It performs many integer multiplies, during which time the bus interface unit can fill up the instruction prefetch queue in parallel with the execution unit performing the multiply. In this program, the bus interface unit can perform bus operations faster than the execution unit actually requires them to be run. In this case, the performance degradation is much less as wait states are added to the memory interface. The execution time of this program is closer to the number of clock cycles calculated by adding the number of cycles per instruction because the execution unit does not have to wait for the bus interface unit to place an opcode byte in the prefetch queue as often. Thus, fewer clock cycles are wasted by the execution unit laying idle for want of instructions. Table G-1 lists the execution times measured for these two programs as wait states were introduced with the 80186 running at 8 MHz.

Table G-1

# of Wait States	Program 1		Program 2	
	Exec Time (μsec)	Perf Degr	Exec Time (μsec)	Perf Degr
0	505		294	
1	595	18%	311	6%
2	669	12%	337	8%
3	752	12%	347	3%

```
$mod186
name                    example_wait_state_performance
;
; This file contains two programs which demonstrate the 80186 performance
;       degradation as wait states are inserted. Program 1 performs a
;       transformation between two types of characters sets, then copies
;       the transformed characters back to the original buffer (which is 64
;       bytes long. Program 2 performs the same type of transformation, however
;       instead of performing a table lookup, it multiplies each number in the
;       original 32 word buffer by a constant (3, note the use of the integer
;       immediate multiply instruction). Program "nothing" is used to measure
;       the call and return times from the driver program only.
;
cgroup                  group   code
dgroup                  group   data
data                    segment                 public 'data'
```

210973–95

```
t_table             db        256 dup (?)
t_string            db        64 dup (?)
m_array             dw        32 dup (?)
data                ends

code                segment                                public 'code'
                    assume    CS:cgroup,DS:dgroup
                    public    bench_1,bench_2,nothing_,wait_state_,set_timer_
bench_1             proc      near
                    push      SI                           ;  save registers used
                    push      CX
                    push      BX
                    push      AX

                    mov       CX,64                        ;  translate 64 bytes
                    mov       SI,0
                    mov       BH,0

loop_back:
                    mov       BL,t_string[SI]              ;  get the byte
                    mov       AL,t_table[BX]               ;  translate byte
                    mov       t_string[SI],AL              ;  and store it
                    inc       SI                           ;  increment index
                    loop      loop_back                    ;  do the next byte

                    pop       AX
                    pop       BX
                    pop       CX
                    pop       SI
                    ret
bench_1             endp

bench_2             proc      near
                    push      AX                           ;  save registers used
                    push      SI
                    push      CX

                    mov       CX,32                        ;  multiply 32 numbers
                    mov       SI,offset m_array

loop_back_2:
                    imul      AX,word ptr [SI],3           ;  immediate multiply
                    mov       word ptr [SI],AX
                    inc       SI
                    inc       SI
                    loop      loop_back_2

                    pop       CX
                    pop       SI
                    pop       AX
                    ret
bench_2_            endp
```

210973-96

```
nothing_                proc     near
                        ret
nothing_                endp
;
; wait_state(n) sets the 80186 LMCS register to the number of wait states
;       (0 to 3) indicated by the parameter n (which is passed on the stack).
;       No other bits of the LMCS register are modified.
;
wait_state_             proc     near
                        enter    0,0                    ; set up stack frame
                        push     AX                     ; save registers used
                        push     BX
                        push     DX

                        mov      BX,word ptr [BP + 4]   ; get argument
                        mov      DX,0FFA2h              ; get current LMCS register
contents
                        in       AX,DX

                        and      AX,0FFFCh              ; and off existing ready bits
                        and      BX,3                   ; insure ws count is good
                        or       AX,BX                  ; adjust the ready bits
                        out      DX,AX                  ; and write to LMCS

                        pop      DX
                        pop      BX
                        pop      AX
                        leave                           ; tear down stack frame
                        ret
wait_state_             endp
;
; set_timer() initializes the 80186 timers to count microseconds. Timer 2
;       is set up as a prescaler to timer 0, the microsecond count can be read
;         directly out of the timer 0 count register at location FF50H in I/O
;         space.
;
set_timer_              proc     near
                        push     AX
                        push     DX

                        mov      DX,0ff66h              ; stop timer 2
                        mov      AX,4000h
                        out      DX,AX

                        mov      DX,0ff50h              ; clear timer 0 count
                        mov      AX,0
                        out      DX,AX

                        mov      DX,0ff52h              ; timer 0 counts up to 65535
                        mov      AX,0
                        out      DX,AX
```

210973-97

```
                        mov     DX,0ff56h               ;  enable timer 0
                        mov     AX,0c009h
                        out     DX,AX

                        mov     DX,0ff60h               ;  clear timer 2 count
                        mov     AX,0
                        out     DX,AX

                        mov     DX,0ff62h               ;  set maximum count of timer 2
                        mov     AX,2
                        out     DX,AX

                        mov     DX,0ff66h               ;  re-enable timer 2
                        mov     AX,0c001h
                        out     DX,AX

                        pop     DX
                        pop     AX
                        ret
set_timer_              endp
code                    ends
                        end
```

210973-98

APPENDIX H
80186 NEW INSTRUCTIONS

The 80186 performs many additional instructions to those of the 8086. These instructions appear shaded in the instruction set summary at the back of the 80186 data sheet. This appendix explains the operation of these new instructions. In order to use these new instructions with the 8086/186 assembler, the "$mod186" switch must be given to the assembler. This can be done by placing the line: "$mod186" at the beginning of the assembly language file.

PUSH IMMEDIATE

This instruction allows immediate data to be pushed onto the processor stack. The data can be either an immediate byte or an immediate word. If the data is a byte, it will be sign extended to a word before it is pushed onto the stack (since all stack operations are word operations).

PUSHA, POPA

These instructions allow all of the general purpose 80186 registers to be saved on the stack, or restored from the stack. The registers saved by this instruction (in the order they are pushed onto the stack) are AX, CX, DX, BX, SP, BP, SI, and DI. The SP value pushed onto the stack is the value of the register before the first PUSH (AX) is performed; the value popped for the SP register is ignored.

This instruction does not save any of the segment registers (CS, DC, SS, ES), the instruction pointer (IP), the flag register, or any of the integrated peripheral registers.

IMUL BY AN IMMEDIATE VALUE

This instruction allows a value to be multiplied by an immediate value. The result of this operation is 16 bits long. One operand for this instruction is obtained using one of the 80186 addressing modes (meaning it can be in a register or in memory). The immediate value can be either a byte or a word, but will be sign extended if it is a byte. The 16-bit result of the multiplication can be placed in any of the 80186 general purpose or pointer registers.

This instruction requires three operands: the register in which the result is to be placed, the immediate value, and the second operand. Again, this second operand can be any of the 80186 general purpose registers or a specified memory location.

SHIFTS/ROTATES BY AN IMMEDIATE VALUE

The 80186 can perform multiple bit shifts or rotates where the number of bits to be shifted is specified by an immediate value. This is different from the 8086, where only a single bit shift can be performed, or a multiple shift can be performed where the number of bits to be shifted is specified in the CL register.

All of the shift/rotate instructions of the 80186 allow the number of bits shifted to be specified by an immediate value. Like all multiple bit shift operations performed by the 80186, the number of bits shifted is the number of bits specified modulus 32 (i.e., the maximum number of bits shifted by the 80186 multiple bit shifts is 31).

These instructions require two operands: the operand to be shifted (which may be a register or a memory location specified by any of the 80186 addressing modes) and the number of bits to be shifted.

BLOCK INPUT/OUTPUT

The 80186 adds two new input/output instructions: INS and OUTS. These instructions perform block input or output operations. They operate similarly to the string move instructions of the processor.

The INS instruction performs block input from an I/O port to memory. The I/O address is specified by the DX register; the memory location is pointed to by the DI register. After the operation is performed, the DI register is adjusted by 1 (if a byte input is specified) or by 2 (if a word input is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The ES segment register is used for memory addressing, and cannot be overridden. When preceded by a REPeat prefix, this instruction allows blocks of data to be moved from an I/O address to a block of memory. Note that the I/O address in the DX register is not modified by this operation.

The OUTS instruction performs block output from memory to an I/O port. The I/O address is specified by the DX register; the memory location is pointed to by the SI register. After the operation is performed, the SI register is adjusted by 1 (if a byte output is specified) or by 2 (if a word output is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The DS segment register is used for memory addressing, but can be overridden by using a segment override prefix. When preceded by a REPeat prefix, this instruction allows blocks of data to be moved from a block of memory to an I/O address. Again note that the I/O address in the DX register is not modified by this operation.

Like the string move instruction, these two instructions require two operands to specify whether word or byte operations are to take place. Additionally, this determination can be supplied by the mnemonic itself by adding a "B" or "W" to the basic mnemonic, for example:

```
INSB          ;perform byte input

REP OUTSW     ;perform word block output
```

BOUND

The 80186 supplies a BOUND instruction to facilitate bound checking of arrays. In this instruction, the calculated index into the array is placed in one of the general purpose registers of the 80186. Located in two adjacent word memory locations are the lower and upper bounds for the array index. The BOUND instruction compares the register contents to the memory locations, and if the value in the register is not between the values in the memory locations, an interrupt type 5 is generated. The comparisons performed are SIGNED comparisons. A register value equal to either the upper bound or the lower bound will not cause an interrupt.

This instruction requires two arguments: the register in which the calculated array index is placed, and the word memory location which contains the lower bound of the array (which can be specified by any of the 80186 memory addressing modes). The memory location containing the upper bound of the array must follow immediately the memory location containing the lower bound of the array.

ENTER AND LEAVE

The 80186 contains two instructions which are used to build and tear down stack frames of higher level, block structured languages. The instruction used to build these stack frames is the ENTER instruction. The algorithm for this instruction is:

```
PUSH BP             /*save the previous  frame
                    pointer*/
if level=0 then
   BP:=SP;
else   temp1:=SP;/*save current frame  pointer
                    */
  temp2:= level - 1;
  do while temp2>0/*copy down previous  level
                    frame*/
    BP:= BP - 2;    /*pointers*/
    PUSH [BP];
  BP:=temp1;
  PUSH BP;          /*put current level  frame
                    pointer*/

/*in the save area*/
SP:=SP - disp;      /*create space on the  stack
                    for*/

/*local variables*/
```

Figure H-1 shows the layout of the stack before and after this operation.

This instruction requires two operands: the first value (disp) specifies the number of bytes the local variables of this routine require. This is an unsigned value and can be as large as 65535. The second value (level) is an unsigned value which specifies the level of the procedure. It can be as great as 255.

The 80186 includes the LEAVE instruction to tear down stack frames built up by the ENTER instruction. As can be seen from the layout of the stack left by the ENTER instruction, this involves only moving the contents of the BP register to the SP register, and popping the old BP value from the stack.

Neither the ENTER nor the LEAVE instructions save any of the 80186 general purpose registers. If they must be saved, this must be done in addition to the ENTER and the LEAVE. In addition, the LEAVE instruction does not perform a return from a subroutine. If this is desired, the LEAVE instruction must be explicitly followed by the RET instruction.

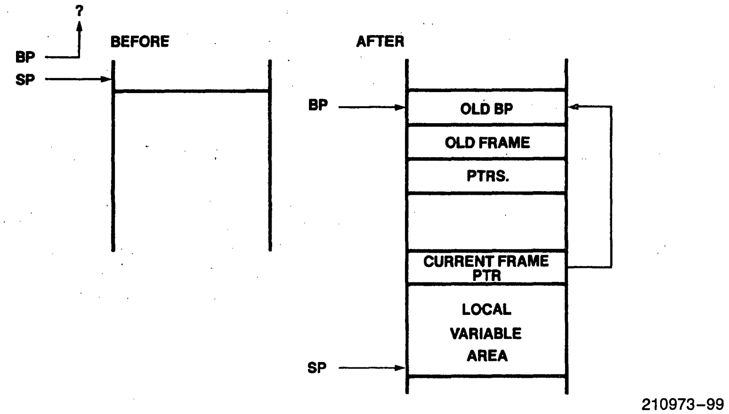


Figure H-1. ENTER Instruction Stack Frame

APPENDIX I
80186/80188 DIFFERENCES

The 80188 is exactly like the 80186, except it has an 8 bit external bus. It shares the same execution unit, timers, peripheral control block, interrupt controller, chip select, and DMA logic. The differences between the two caused by the narrower data bus are:

- The 80188 has a 4 byte prefetch queue, rather than the 6 byte prefetch queue present on the 80186. The reason for this is since the 80188 fetches opcodes one byte at a time, the number of bus cycles required to fill the smaller queue of the 80188 is actually greater than the number of bus cycles required to fill the queue of the 80186. As a result, a smaller queue is required to prevent an inordinate number of bus cycles being wasted by prefetching opcodes to be discarded during a jump.
- AD8–AD15 on the 80186 are transformed to A8–A15 on the 80188. Valid address information is present on these lines throughout the bus cycle of the 80188. Valid address information is not guaranteed on these lines during idle T states.
- $\overline{\text{BHE}}$/S7 is always defined HIGH by the 80188, since the upper half of the data bus is non-existent.
- The DMA controller of the 80188 only performs byte transfers. The $\overline{\text{B}}$/W bit in the DMA control word is ignored.
- Execution times for many memory access instructions are increased because the memory access must be funnelled through a narrower data bus. The 80188 also will be more bus limited than the 80186 (that is, the execution unit will be required to wait for the opcode information to be fetched more often) because the data bus is narrower. The execution time within the processor, however, has not changed between the 80186 and 80188.

Another important point is that the 80188 internally is a 16-bit machine. This means that any access to the integrated peripheral registers of the 80188 will be done in 16-bit chunks, NOT in 8-bit chunks. All internal peripheral registers are still 16-bits wide, and only a single read or write is required to access the registers. When a word access is made to the internal registers, the BIU will run two bus cycles externally.

Access to the control block may also be done with byte operations. Internally the full 16-bits of the AX register will be written, while externally, only one bus cycle will be executed.

intel®

APPLICATION
NOTE

AP-258

February 1986

High Speed Numerics with the 80186/80188 and 8087

STEVE FARRER
APPLICATIONS ENGINEER

Order Number: 231590-001

1.0 INTRODUCTION

From their introduction in 1982, the highly integrated 16-bit 80186 and its 8-bit external bus version, the 80188, have been ideal processor choices for high-performance, low-cost embedded control applications. The integrated peripheral functions and enhanced 8086 CPU of the 80186 and 80188 allow for an easy upgrade of older generation control applications to achieve higher performance while lowering the overall system cost through reduced board space, and a simplified production flow.

More and more controller applications need even higher performance in numerics, yet still require the low-cost and small form factor of the 80186 and 80188. The 8087 Numerics Data Coprocessor satisfies this need as an optional add-on component.

The 8087 Numeric Data Coprocessor is interfaced to the 80186 and 80188 through the 82188 IBC (Integrated Bus Controller). The IBC provides a highly integrated interface solution which replaces the 8288 used in 8086–8087 systems. The IBC incorporates all the necessary bus control for the 8087 while also providing the necessary logic to support the interface between the 80186/8 and the 8087.

This application note discusses the design considerations associated with using the 8087 Numeric Data Coprocessor with the 80186 and 80188. Sections two, three, and four contain an overview of the integrated circuits involved in the numerics configuration. Section five discusses the interfacing aspects between the 80186/8 and the 8087, including the role of the 82188 Integrated Bus Controller and the operation of the integrated peripherals on the 80186/8 with the 8087. Section six compares the advantages of using an 8087 Numeric Data Coprocessor over software routines written for the host processor as well as the advantage of using an 80186/8 numerics system over an 8086/8088 numerics system.

Except where noted, all future references to the 80186 will apply equally to the 80188.

2.0 OVERVIEW OF THE 80186

The 80186 and 80188 are highly integrated microprocessors which effectively combine up to 20 of the most common system components onto a single chip. The 80186 and 80188 processors are designed to provide both higher performance and a more highly integated solution to the total system.

Higher integration results from integrating system peripherals onto the microprocessor. The peripherals consist of a clock generator, an interrupt controller, a DMA controller, a counter/timer unit, a programmable wait state generator, programmable chip selects, and a bus controller. (See Figure 1.)

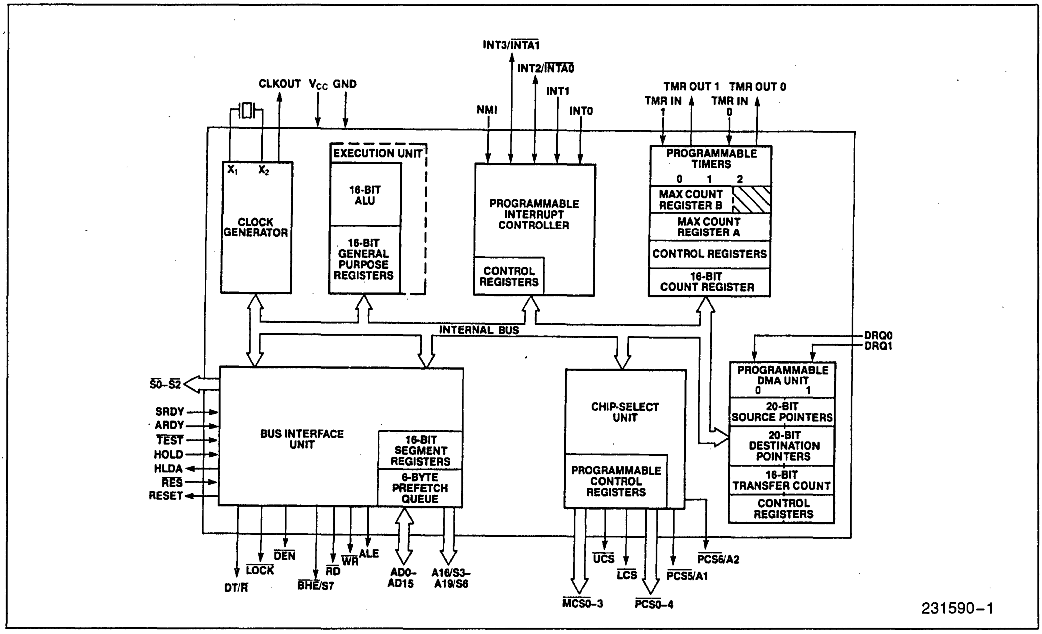


Figure 1. 80186/8 Block Diagram

Higher performance results from enhancements to both general and specific areas of the 8086 CPU, including faster effective address calculation, improvement in the execution speed of many instructions, and the inclusion of new instructions which are designed to produce optimum 80186 code.

The 80186 and 80188 are completely object code compatible with the 8086 and 8088. They have the same basic register set, memory organization, and addressing modes. The differences between the 80186 and 80188 are the same as the differences between the 8086 and 8088: the 80186 has a 16-bit architecture and 16-bit bus interface; the 80188 has a 16-bit internal architecture and an 8-bit data bus interface. The instruction execution times of the two processors differ accordingly: for each non-immediate 16-bit data read/write instruction, 4 additional clock cycles are required by the 80188.

3.0 NUMERICS OVERVIEW

3.1 The Benefits of Numeric Coprocessing

The 8086/8 and 80186/8 are general purpose microprocessors, designed for a very wide range of applications. Typically, these applications need fast, efficient data movement and general purpose control instructions. Arithmetic on data values tends to be simple in these applications. The 8086/8 and 80186/8 fulfill these needs in a low cost, effective manner.

However, some applications require extremely fast and complex math functions which are not provided by a general purpose processor. Such functions as square root, sine, cosine, and logarithms are not directly available in a general purpose processor. Software routines required to implement these functions tend to be slow and not very accurate. Integer data types and their arithmetic operations (i.e., add, subtract, multiply and divide) which are directly available on general purpose processors, still may not meet the needs for accuracy, speed and ease of use.

Providing fast, accurate, complex math can be quite complicated, requiring large areas of silicon on integrated circuits. A general data processor does not provide these features due to the extra cost burden that less complex general applications must take on. For such features, a special numeric data processor is required — one which is easy to use and has a high level of support in hardware and software.

3.2 Introduction to the 8087

The 8087 is a numeric data coprocessor which is capable of performing complex mathematical functions while the host processor (i.e. the main CPU) performs more general tasks. It supports the necessary data types and operations and allows use of all the current hardware and software support for the 8086/8 and 80186/8 microprocessors. The fact that the 8087 is a coprocessor means it is capable of operating in parallel with the host CPU, which greatly improves the processing power of the system.

The 8087 can increase the performance of floating-point calculations by 50 to 100 times, providing the performance and precision required for small business and graphics applications as well as scientific data processing.

The 8087 numeric coprocessor adds 68 floating-point instructions and eight 80-bit floating-point registers to the basic 8086 programming architecture. All the numeric instructions and data types of the 8087 are used by the programmer in the same manner as the general data types and instructions of the host.

The numeric data formats and arithmetic operations provided by the 8087 support the proposed IEEE Microprocessor Floating Point Standard. All of the proposed IEEE floating point standard algorithms, exception detection, exception handling, infinity arithmetic and rounding controls are implemented. The IEEE standard makes it easier to use floating point and helps to avoid common problems that are inherent to floating point.

3.3 Escape Instructions

The coprocessing capabilities of the 8087 are achieved by monitoring the local bus of the host processor. Certain instructions within the 8086 assembly language known as ESCAPE instructions are defined to be coprocessor instructions and, as such, are treated differently.

The coprocessor monitors program execution of the host processor to detect the occurrence of an ESCAPE instruction. The fetching of instructions is monitored via the data bus and bus cycle status S2–S0, while the execution of instructions is monitored via the queue status lines QS0 and QS1.

All ESCAPE instructions start with the high-order 5-bits of the instruction opcode being 11011. They have two basic forms, the memory reference form and the non-memory reference form. The non-memory form, shown in Figure 2A, initiates some activity in the coprocessor using the nine available bits of the ESCAPE instruction to indicate which function to perform.

Memory reference forms of the ESCAPE instruction, shown in Figure 2B, allow the host to point out a memory operand to the coprocessor using any host memory

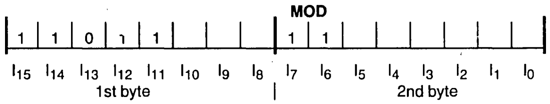


Figure 2A. Non-Memory Reference ESCAPE Instructions

addressing mode. Six bits are available in the memory reference form to identify what to do with the memory operand.

Memory reference forms of ESCAPE instructions are identified by bits 7 and 6 of the byte following the ESCAPE opcode. These two bits are the MOD field of the 8086/8 or 80186/8 effective address calculation byte. Together with the R/M field (bits 2 through 0), they determine the addressing mode and how many subsequent bytes remain in the instruction.

3.4 Host Response to Escape Instructions

The host performs one of two possible actions when encountering an ESCAPE instruction: do nothing (operation is internal to 8087) or calculate an effective address and read a word value beginning at that address (required for all LOADS and STORES). The host ignores the value of the word read and hence the cycle is referred to as a "Dummy Read Cycle." ESCAPE instructions do not change any registers in the host other than advancing the IP. If there is no coprocessor or the coprocessor ignores the ESCAPE instruction, the ESCAPE instruction is effectively a NOP to the host. Other than calculating a memory address and reading a word of memory, the host makes no other assumptions regarding coprocessor activity.

The memory reference ESCAPE instructions have two purposes: to identify a memory operand and, for certain instructions, to transfer a word from memory to the coprocessor.

3.5 Coprocessor Response to Escape Instructions

The 8087 performs basically three types of functions when encountering an ESCAPE instruction: LOAD (read from memory), STORE (write to memory), and EXECUTE (perform one of the internal 8087 math functions).

When the host executes a memory reference ESCAPE instruction intended to cause a read operation by the 8087, the host always reads the low-order word of any 8087 memory operand. The 8087 will save the address and data read. To read any subsequent words of the operand, the 8087 must become a local bus master.

When the 8087 has the local bus, it increments the 20-bit physical address it saved to address the remaining words of the operand.

When the ESCAPE instruction is intended to cause a write operation by the 8087, the 8087 will save the address but ignore the data read. Eventually, it will get control of the local bus and perform successive writes incrementing the 20-bit address after each word until the entire numeric variable has been written.

ESCAPE instructions intended to cause the execution of a coprocessor calculation do not require any bus activity. Numeric calculations work off of an internal register stack which has been initialized using a LOAD operation. The calculation takes place using one or two of the stack positions specified by the ESCAPE instruction. The result of the operation is also placed in one of the stack positions specified by the ESCAPE instruction. The result may then be returned to memory using a STORE instruction, thus allowing the host processor to access it.

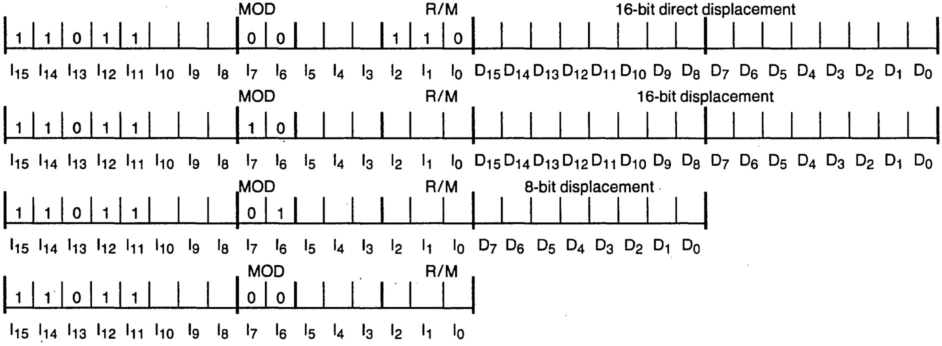

MOD
R/M
16-bit direct displacement
1 1 0 1 1 0 0 1 1 0
I15 I14 I13 I12 I11 I10 I9 I8 I7 I6 I5 I4 I3 I2 I1 I0 D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
MOD
R/M
16-bit displacement
1 1 0 1 1 1 0
I15 I14 I13 I12 I11 I10 I9 I8 I7 I6 I5 I4 I3 I2 I1 I0 D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
MOD
R/M
8-bit displacement
1 1 0 1 1 0 1
I15 I14 I13 I12 I11 I10 I9 I8 I7 I6 I5 I4 I3 I2 I1 I0 D7 D6 D5 D4 D3 D2 D1 D0
MOD
R/M
1 1 0 1 1 0 0
I15 I14 I13 I12 I11 I10 I9 I8 I7 I6 I5 I4 I3 I2 I1 I0

Figure 2B. Memory Reference ESCAPE Instruction Forms

4.0 OVERVIEW OF THE 82188 INTEGRATED BUS CONTROLLER

4.1 Introduction

The 82188 Integrated Bus Controller (IBC) is a highly integrated version of the 8288 Bus Controller. The IBC provides command and control timing signals for bus control and all of the necessary logic to interface the 80186 to the 8087.

4.2 Bus Control Signals

The bus command and control signals consist of RD, WR, DEN, DT/R, and ALE. The timings and levels are driven following the latching of valid signals on the status lines S0–S2. When S0–S2 change state from passive to active, the IBC begins cycling through a state machine which drives the corresponding control and command lines for the bus cycle. As with the 8288, an address enable input (AEN) is present to allow tri-stating when other bus masters supply their own bus control signals.

4.3 Bus Arbitration

The IBC also has the ability to convert bus arbitration protocols of RQ/GT to HOLD-HLDA. This allows the 82586 Local Area Network (LAN) Coprocessor, the 82730 Text Coprocessor, and other coprocessors using the HOLD-HLDA protocol to be interfaced to the 8086/8 as well as allowing the 80186/8 to be interfaced to the 8087. In addition to converting arbitration protocols, the IBC makes it possible to arbitrate between two bus masters using HOLD-HLDA with a third using RQ/GT.

4.4 Interface Logic

In addition to all the bus control and arbitration features, the IBC provides logic to connect the queue status to the 8087, a chip-select for the 8087, and the necessary READY synchronization required between the 8087 and the 80186/8.

5.0 DESIGNING THE SYSTEM

5.1 Circuit Schematics of the 80186/8-82888-8087 System

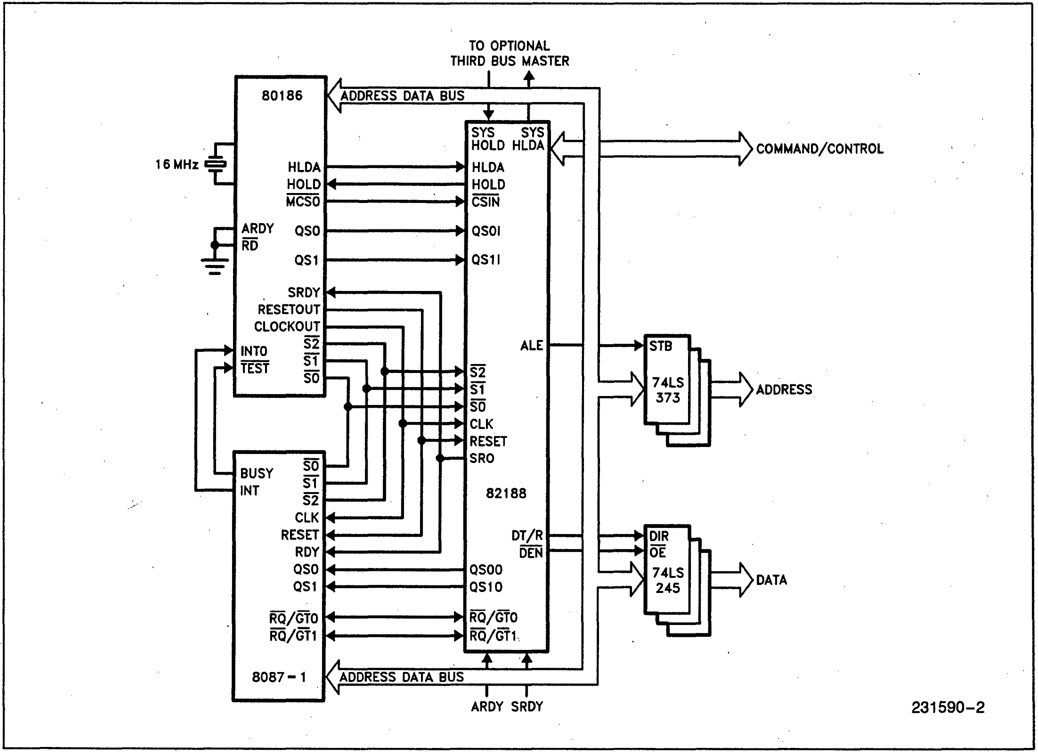


Figure 3. 80186/8-82188-8087 Circuit Diagram

5.2 Queue Status

The 8087 tracks the instruction execution of the 80186 by keeping an internal instruction queue which is identical to the processor's instruction queue. Each time the processor performs an instruction fetch, the 8087 latches the instruction into its own queue in parallel with the processor. Each time the processor removes the first byte of an instruction from the queue, the 8087 removes the byte at the top of the 8087 queue and checks to see if the byte is an ESCAPE prefix. If it is, the 8087 decodes the following bytes in parallel with the processor to determine which numeric instruction the bytes represent. If the first byte of the instruction is not an ESCAPE prefix, the 8087 discards it along with the subsequent bytes of the non-numeric instruction as the 80186 removes them from the queue for execution.

The 8087 operates its internal instruction queue by monitoring the two queue status lines from the CPU. This status information is made available by the CPU by placing it into queue status mode. This requires strapping the $\overline{\text{RD}}$ pin on the 80186 to ground. When $\overline{\text{RD}}$ is tied to ground, ALE and $\overline{\text{WR}}$ become QS0 (Queue Status #0) and QS1 (Queue Status #1) respectively.

Table 1. Queue Status Decoding

QS1	QS0	Queue Operation
0	0	No queue operation
0	1	First byte from queue
1	0	Subsequent byte from queue
1	1	Reserved

Each time the 80186 begins decoding a new instruction, the queue status lines indicate "first byte of instruction taken from the queue". This signals the 8087 to check for an ESCAPE prefix. As the remaining bytes of the instruction are removed, the queue status indicates "subsequent byte removed from queue". The 8087 uses this status to either continue decoding subsequent bytes, if the first byte was an ESCAPE prefix, or to discard the subsequent bytes if the first byte was not an ESCAPE prefix.

The QS0(ALE) and QS1($\overline{\text{WR}}$) pins of the 80186 are fed directly to the 82188 where they are latched and delayed by one-half-clock. The delayed queue status from the 82188 is then presented directly to the 8087.

The waveforms of the queue status signals are shown in Figure 4. The critical timings are the setup time into the 82188 from the 80186 and the setup and hold time into the 8087 from the 82188. The calculations for an 8 MHz system are as follows:

$.5T_{CLCL} - T_{CHQSV}$ (186 max) $\geq T_{QIVCL}$ (82188 min) ;setup to 82188
$.5(125 \text{ ns}) - 35 \geq 15 \text{ ns}$

$T_{CLCL} - T_{CLQOV}$ (82188 max) $\geq T_{QVCL}$;setup to 8087
$(125 \text{ ns}) - 50 \geq 10 \text{ ns}$

T_{CLQOV} (82188 min) $\geq T_{CLQX}$ (8087 min) ;hold to 8087
$5 \geq 5 \text{ ns}$

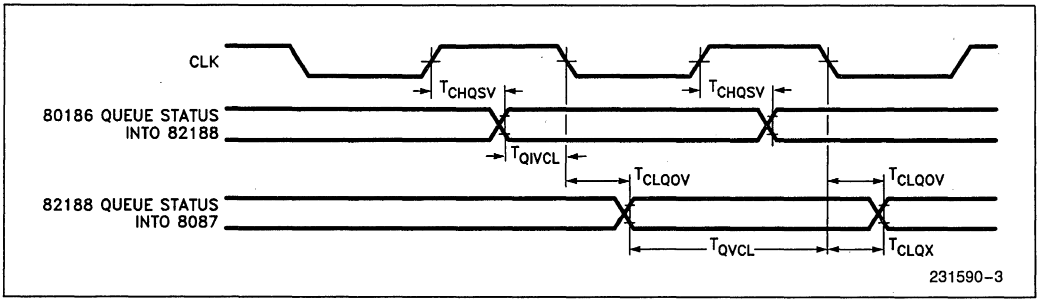


Figure 4. Queue Status Timing

5.3 Bus Control Signals

When the 80186 is in Queue Status mode, another component must generate the ALE, $\overline{RD}$, and $\overline{WR}$ signals. The 82188 provides these signals by monitoring the CPU bus cycle status ($\overline{S0}$–$\overline{S2}$). Also provided are $\overline{DEN}$ and DT/$\overline{R}$ which may be used for extra drive capability on the control bus. With the exception of ALE, all control signals on the 82188 are almost identical to their corresponding 80186 control signals. This section discusses the differences between the 80186 and the 82188 control signals for the purpose of upgrading an 80186 design to an 80186–8087 design. For original 80186–8087 designs, there is no need to compare control signal timings of the 82188 with the 80186.

5.3.1 ALE

The ALE (Address Latch Enable) signal goes active one clock phase earlier on the 80186 than on the 82188. Timing of the ALE signal on the 82188 is closer to that of the 8086 and 8288 bus controller because the bus cycle status is used to generate the ALE pulse. ALE on the 80186 goes active before the bus cycle status lines are valid.

The inactive edge of ALE occurs in the same clock phase for both the 80186 and the 82188. The setup and hold times of the 80186 address relative to the 82188 ALE signal are shown in Figure 5 and are calculated for an 8 MHz system as follows:

Setup Time

$$\text{For 80186} = T_{AVCH}\ (186\ \text{min}) + T_{CHLL}\ (82188\ \text{min}) = 10 + 0 = 10\ \text{ns.}$$

$$\text{For 8087} = 0.5\ (T_{CLCL}) - T_{CLAV}\ (8087\ \text{max}) + T_{CHLL}\ (82188\ \text{min}) = 0.5\ (125) - 55 + 0 = 7.5$$

Hold Time

$$= 0.5\ (T_{CLCL}) - T_{CHLL}\ (82188\ \text{max}) + T_{CLAZ}\ (186\ \text{min}) = 0.5\ (125) - 30 + 10 = 42.5\ \text{ns.}$$

NOTE:
The hold time calculation is the same for both the 80186 and 8087.

These timings provide adequate setup and hold times for a 74LS373 address latch.

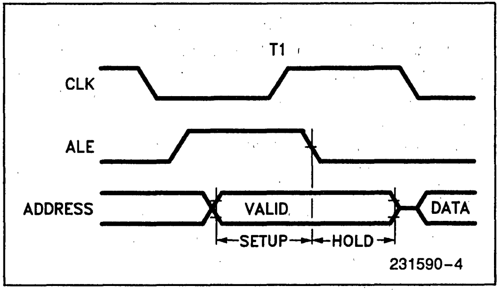
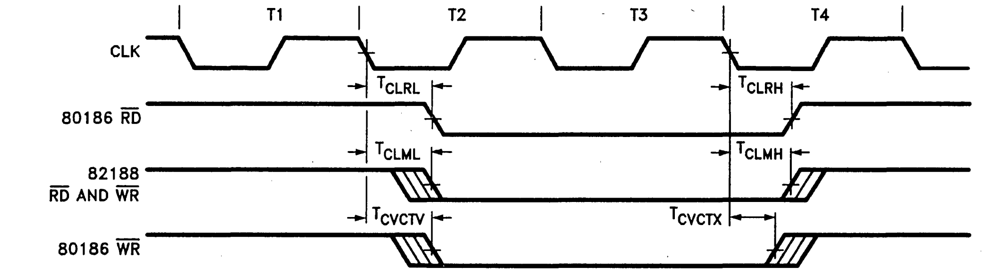


Figure 5. Address Latch Timings



$T_{CLRL} = T_{CLML} = T_{CVCTV}$ = 10 to 70 ns
$T_{CLRH} = T_{CLMH}$ = 10 to 55 ns
T_{CVCTX} = 5 to 55 ns

231590-5

Figure 6. Read and Write Timings

5.3.2 Read and Write

The read and write signals of the 82188 have identical timings to those of the 80186 with one exception: the 82188 $\overline{WR}$ inactive edge may not go inactive quite as early as the 80186. This spec is, in fact, a tighter spec than the 80186 $\overline{WR}$ timing and should make designs easier. The timings for $\overline{RD}$ and $\overline{WR}$ are shown in Figure 6 for both the 80186 and the 82188.

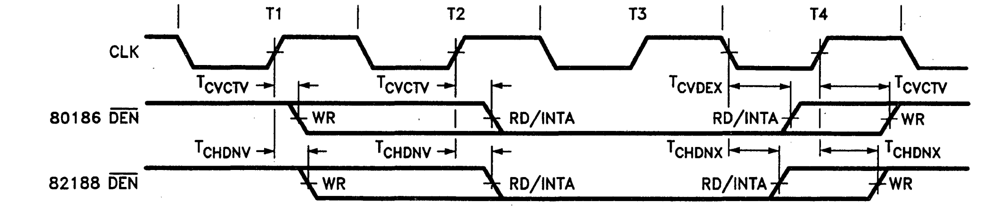
5.3.3 $\overline{DEN}$

The $\overline{DEN}$ signal on the 82188 is identical to the $\overline{DEN}$ signal on the 80186 but with a tighter timing specification. This makes designs easier with the 82188 and makes upgrades from 80186 bus control to 82188 bus control more straightforward. The timings for $\overline{DEN}$ on both the 80186 and 82188 are shown in Figure 7.

5.3.4 DT/$\overline{R}$

The operation of the DT/$\overline{R}$ signal varies somewhat between the 80186 and the 82188. The 80186 DT/$\overline{R}$ signal will remain in an active high state for all write cycles and will default to a high state when the system bus is idle (i.e., no bus activity). The 80186 DT/$\overline{R}$ goes low only for read cycles and does so only for the duration of the bus cycle. At the end of the read cycle, assuming the following cycle is a non-read, the DT/$\overline{R}$ signal will default back to a high state. Back-to-back read cycles will result in the DT/$\overline{R}$ signal remaining low until the end of the last read cycle.

The DT/$\overline{R}$ signal on the 82188 operates differently by making transitions only at the start of a bus cycle. The 82188 DT/$\overline{R}$ signal has no default state and therefore will remain in whichever state the previous bus cycle required. The 82188 DT/$\overline{R}$ signal will only change states when the current bus cycle requires a state different from the previous bus cycle.



231590-6

T_{CVCTV} = 10 to 70 - clock edge to $\overline{DEN}$ active/inactive
T_{CVDEX} = 10 to 70 - falling edge of T4 to $\overline{DEN}$ inactive
T_{CHDNV} = 10 to 55 - rising edge of clock to $\overline{DEN}$ active
T_{CHDNX} = 10 to 55 - clock edge to $\overline{DEN}$ inactive

Figure 7. Data Control Timings

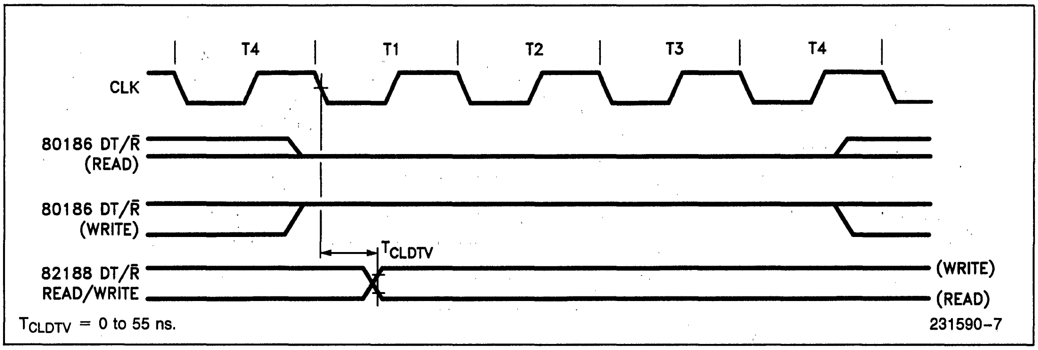


Figure 8. Data Transmit & Receive Timings

5.4 Chip Selects

5.4.1 INTRODUCTION

Chip-select circuitry is typically accomplished by using a discrete decoder to decode two or more of the upper address lines. When a valid address appears on the address bus, the decoder generates a valid chip-select. With this method, any bus master capable of placing an address on the system bus is able to generate a chip-select. An example of this is shown in Figure 9 where an 8086/8087 system uses a common decoder on the address bus. Note the decoder is able to operate regardless of which processor is in control of the bus.

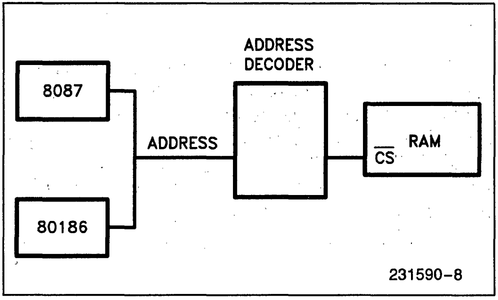


Figure 9. Typical 8086/8087 System

With high integration processors like the 80186 and 80188, the chip-select decoder is integrated onto the processor chip. The integrated chip-selects on the 80186 enable direct processor connection to the chip-enable pins on many memory devices, thus eliminating an external decoder. But because the integrated chip-selects decode the 80186's internal bus, an external bus master, such as the 8087, is unable to activate them. The 82188 IBC solves this problem by supplying a chip-select mechanism which may be activated by both the host processor and a second processor.

5.4.2 $\overline{\text{CSI}}$ AND $\overline{\text{CSO}}$ OF THE 82188

The $\overline{\text{CSI}}$ (chip select in) and $\overline{\text{CSO}}$ (chip select out) pins of the 82188 provide a way for a second bus master to select memory while also making use of the 80186 integrated chip-selects. The $\overline{\text{CSI}}$ pin of the 82188 connects directly to one of the 80186's chip-selects while $\overline{\text{CSO}}$ connects to the memory device designated for the chip-selects range. An example of this is shown in Figure 10.

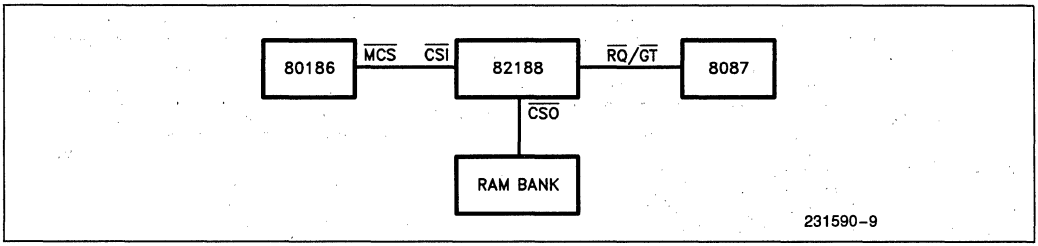


Figure 10. Typical 80186/82188/8087 System

When the 80186 has control of the bus, the circuit acts just as a buffer and the memory device gets selected as if the circuit had not been there. Whenever $\overline{\text{CSI}}$ goes active, $\overline{\text{CSO}}$ goes active. When a second bus master, such as the 8087, takes control of the bus, $\overline{\text{CSO}}$ goes active and remains active until the 8087 passes control back to the processor. At this time $\overline{\text{CSO}}$ is deactivated.

A functional block diagram of the $\overline{\text{CSI}}$–$\overline{\text{CSO}}$ circuit is shown in Figure 11. A grant pulse on the $\overline{\text{RQ}}/\overline{\text{GT}}0$ line gives control to the 8087 and also causes the $\overline{\text{8087CONTROL}}$ signal to go active, which in turn causes $\overline{\text{CSO}}$ to go active. The $\overline{\text{8087CONTROL}}$ signal goes inactive when either a release is received on $\overline{\text{RQ}}/\overline{\text{GT}}0$, indicating that the 8087 is relinquishing control to the main processor, or a grant is received on the $\overline{\text{RQ}}/\overline{\text{GT}}1$ line, indicating that the 8087 is relinquishing control to a third processor. Both actions signify that the 8087 is relinquishing the bus. If $\overline{\text{CSO}}$ goes inactive because a third processor took control of the bus, then $\overline{\text{CSO}}$ will go active again for the 8087 when a release pulse is transmitted on the $\overline{\text{RQ}}/\overline{\text{GT}}1$ line to the 8087. This release pulse occurs as a result of SYSHLDA going inactive from the third processor.

5.4.3 SYSTEM DESIGN EXAMPLE

To provide the 8087 access to data in low memory through an integrated chip-select, the $\overline{\text{LCS}}$ pin should be disconnected from the bank that it is currently selecting and fed directly into the 82188 $\overline{\text{CSI}}$. The $\overline{\text{CSI}}$ output should be connected to the banks which the $\overline{\text{LCS}}$ formerly selected. The $\overline{\text{LCS}}$ will still select the same banks because $\overline{\text{CSO}}$ goes active whenever $\overline{\text{CSI}}$ goes active. But now the 8087, when taking control of the bus, may also select these banks.

Care must be taken in locating the 8087 data area because it must reside in the area in which the chip-select is defined. If the 8087 generates an address outside of the $\overline{\text{LCS}}$ range, the $\overline{\text{CSO}}$ will still go active, but the address will erroneously select a part of the lower bank. Note also that this chip-select limits the size of the 8087 data area to the maximum size memory which can be selected with one chip-select. However, this does not place a limit on instruction code size or non-8087 data size. All 80186 and 8087 instructions are fetched by the processor and therefore do not require that the 8087 be

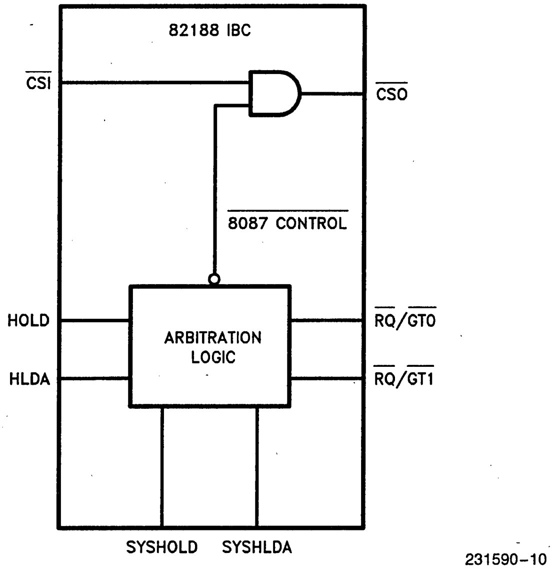

82188 IBC
$\overline{\text{CSI}}$
$\overline{\text{CSO}}$
$\overline{\text{8087 CONTROL}}$
ARBITRATION LOGIC
HOLD
HLDA
$\overline{\text{RQ}}/\overline{\text{GT0}}$
$\overline{\text{RQ}}/\overline{\text{GT1}}$
SYSHOLD
SYSHLDA
231590–10

Figure 11. 82188 Chip Select Circuitry

able to address them. Likewise, non-8087 data is never accessed by the 8087 and therefore does not require an 8087 chip-select.

5.5 Wait State & Ready Logic

The 8087 must accurately track every instruction fetch the 80186 performs so that each op-code may be read from the system bus by the 8087 in parallel with the processor. This means that for instruction code areas, the 80186 cannot use internally generated wait states. All ready logic for these areas must be generated externally and sent into the 82188. The 82188 then presents a synchronous ready out (SRO) signal to both the 80186 and the 8087.

5.5.1 INTERNAL WAIT STATES WITH INSTRUCTION FETCHES

If internal wait states are used by the processor with the 8087 at zero wait states, then the 8087 will latch opcodes using a four clock bus cycle while the processor is using between five and seven clocks on each bus cycle. If the wait states are truly necessary to latch valid data from memory, then a four clock bus cycle will force the 8087 to latch invalid data. The invalid data may then be possibly interpreted to be an ESCAPE prefix when, in reality, it is not. The reverse may also hold true in that the 8087 may not recognize an ESCAPE prefix when it is fetched. These conditions could cause a system to hang (i.e., cease to operate), or operate with erroneous results.

If the memory is fast enough to allow latching of valid data within a four clock bus cycle, then the 80186 internal wait states will not cause the system to hang. Both processors will receive valid data during their respective bus cycles. The 8087 will finish its bus cycle earlier than the processor, but this is of no consequence to system operation. The 8087 will synchronize with the processor using the status lines S0–S2 at the start of the next instruction fetch.

5.5.2 INTERNAL WAIT STATES WITH DATA & I/O CYCLES

With the exception of "Dummy Read Cycles" and instruction fetches, all memory and I/O bus cycles executed by the host processor are ignored by the 8087. Coprocessor synchronization is not required for untracked bus cycles and, therefore, internally generated wait states do not affect system operation. All of the I/O space and any part of memory used strictly for data may use the internal wait state generator on the 80186.

Memory used for 8087 data is somewhat different. Here, as in the case of code segment areas, the system must rely on an external ready signal or else the memory must be fast enough to support zero wait state operation. Without an external ready signal, the 8087 will always perform a four clock bus cycle which, when used with slow memories, results in the latching of invalid data.

Internal wait states will not affect system operation for data cycles performed by the 8087. In this case the 8087 has control of the bus and the two processors operate independently.

One type of data cycle has not yet been considered. Each time a numerics variable is accessed, the host processor runs a "Dummy Read Cycle" in order to calculate the operand address for the 8087. The 8087 latches the address and then takes control of the bus to fetch any subsequent bytes which are necessary. If the 8087 variables are located at even addresses, then an internally generated wait state will not present any problems to the system. If any numeric variables are located at odd addresses, then the interface between the host and coprocessor becomes asynchronous causing erroneous results.

The erroneous results are due to the 80186 running two back-to-back bus cycles with wait states while the 8087 runs two back-to-back bus cycles without wait states. The start of the second bus cycle is completely uncoordinated between the two processors and the 8087 is unable to latch the correct address for subsequent transfers. For this reason, 8087 variables in a 80186 system must always lie on even boundaries when using the internal wait state generator to access them.

Numeric variables in an 80188 system must never be in a section of memory which uses the internal wait state generator. The 80188 will always perform consecutive bus cycles which would be equivalent to the 80186 performing an odd addressed "Dummy Read Cycle."

5.5.3 AUTOMATIC WAIT STATES AT RESET

The 80186 automatically inserts three wait states to the predefined upper memory chip select range upon power up and reset. This enables designers to use slow memories for system boot ROM if so desired. If slow ROM's are chosen, then no further programming is necessary. If fast ROM's are chosen, then the wait state logic may simply be reprogrammed to the appropriate number of wait states.

The automatic wait states have the possibility of presenting the same problem as described in section 5.5.1 if the boot ROM needs one or more wait states. Under these conditions the 8087 would be forced to latch invalid opcodes and possibly mistake one for an ESCAPE instruction.

If the boot ROM requires wait states, then some sort of external ready logic is necessary. This allows both processors to run with the same number of wait states and insures that they always receive valid data.

If the boot ROM does not require wait states, then there is no need to design external ready logic for the upper chip select region. But if 8087 code is present in the upper memory chip select region, the situation described in section 3.4 regarding "Dummy Read Cycles" must be considered.

The 82188 solves this problem by inserting three wait states on the SRO line to the 8087 for the first 256 bus cycles. By doing this the 82188 inserts the same number of wait states to both processors keeping them synchronized. The initialization code for the 80186 must program the upper memory chip select to look at external ready and to insert zero wait states within these first 256 bus cycles. At the end of the 256 bus cycles, the 82188 stops inserting wait states and both processors run at zero wait states.

5.5.4 EXTERNAL READY SYNCHRONIZATION

The 80186 and 8087 sample READY on different clock edges. This implies that some sort of external synchronization is required to insure that both processors sample the same ready state. Without the synchronization, it would be possible for the external signal to change state between samples. The 80186 may sample ready high while the 8087 samples ready low. This would lead to the two processors running different length bus cycles and possibly cause the system to hang.

The 82188 provides ready synchronization through the ARDY and SRDY inputs. Once a valid transition is recorded, the 82188 presents the results on the SRO output and holds the output in that state until both processors have had a chance to sample the signal.

5.6 BUS ARBITRATION

In order for the 8087 to read and write numeric data to and from memory, it must have a means of taking control of the local bus. With the 8086/88 this is accomplished through a request-grant exchange protocol. The 80186, however, makes use of HOLD/HOLD ACKNOWLEDGE protocol to exchange control of the bus with another processor. The 82188 supplies the necessary conversion to interface $\overline{\text{RQ}}/\overline{\text{GT}}$ to HOLD/HLDA signals. The $\overline{\text{RQ}}/\overline{\text{GT}}$ signal of the 8087 connects directly to the 82188's $\overline{\text{RQ}}/\overline{\text{GT}}$0 input while the 82188's HOLD and HLDA pins connect to the 80186's HOLD and HLDA pins.

When the 8087 requires control of the bus, the 8087 sends a request on the $\overline{\text{RQ}}/\overline{\text{GT}}$0 line to the 82188. The 82188 responds by sending a HOLD request to the 80186. When HLDA is received back from the 80186, the 82188 sends a grant back to the 8087 on the same $\overline{\text{RQ}}/\overline{\text{GT}}$0 line.

The 82188 also has provisions for adding a third busmaster to the system which uses HOLD/HLDA protocol. This is accomplished by using the 82188 SYSHOLD, SYSHLDA, and $\overline{\text{RQ}}/\overline{\text{GT}}$1 signals. The third processor requests the bus by pulling the SYSHOLD line high. The 82188 will route (and translate if necessary) the requests to the current bus master. If the 8087 has control, the 82188 will request control via the $\overline{\text{RQ}}/\overline{\text{GT}}$1 line which should be connected to the 8087's $\overline{\text{RQ}}/\overline{\text{GT}}$1 line.

The 8087 will relinquish control by getting off the bus and sending a grant pulse on the $\overline{\text{RQ}}/\overline{\text{GT}}$1 line. The 82188 responds by sending a SYSHLDA to the third processor. The third processor lowers SYSHOLD when it has finished on the bus. The 82188 routes this in the form of a release pulse on the $\overline{\text{RQ}}/\overline{\text{GT}}$1 line to the 8087. The 8087 then continues bus activity where it left off. The maximum latency from SYSHOLD to SYSHLDA is equal to the 80186 latency + 8087 latency + 82188 latency.

5.7 SPEED REQUIREMENTS

One of the most important timing specs associated with the 80186-8087 interface is the speed at which the system should run. The 8087 was designed to operate with a 33% duty cycle clock whereas the 80186 and 80188 were designed to operate with a 50% duty cycle clock. In order to run both parts off the same clock, the 8087 must run at a slower speed than is typically implied by its dash number in the 8086/88 family.

To determine the speed at which an 8087 may run (with a 50% duty cycle clock), the minimum low and high times of the 8087 must be examined. The maximum of these two minimum specs becomes the half-period of the 50% duty cycle system clock. For example, the 8087-1 provides worst case spec compatibility with the 80186 at system clock-speeds of up to 8 MHz. The clock waveforms are shown in Figure 12 using 10 MHz timings.

The minimum clock low time spec (T_{CLCH}) of the 10 MHz 8087 is 53 ns. The clock low time of an 8 MHz 80186 is specified to be:

$$\frac{1}{2}(T_{CLCL}) - 7.5$$

Solving for T_{CLCL} of the 80186 using T_{CLCH} of the 8087 yields the following:

$$\frac{1}{2}(T_{CLCL}) - 7.5 = T_{CLCH}$$

$$(T_{CLCL}) = 2(T_{CLCH} + 7.5)$$

$$T_{CLCL} = 121 \text{ ns}$$

The calculation shows minimum cycle time of the 80186 to be 121 ns. This time translates into a maximum frequency of 8.26 MHz.

6.0 BENCHMARKS

6.1 Introduction

The following benchmarks compare the overall system performance of an 8086, 80188, and an 80186 in numeric applications. Results are shown for all three processors in systems with the 8087 coprocessor and in systems using an 8087 software emulator. Three FORTRAN benchmark programs are used to demonstrate the large increase in floating-point math performance provided by the 8087 and also the increase in performance due to the enhanced 80186 and 80188 host processors.

The 8086 results were measured on an Intellec® Series III Microcomputer Development System with an iSBC® 86/12 board and an iSBC 337 multimodule. Typically, one wait state for memory read cycles and two wait states for memory write cycles are experienced in this environment.

The 80186 and 80188 results were measured on a prototype board which allowed zero wait state operation at 8 MHz. The benchmarks measured using the 8087 showed little sensitivity to wait states. Instructions executed on the 8087 tend to be long in comparison to the amount of bus activity required and, therefore, are not affected much by wait states.

The benchmarks measured using the software emulator are much more bus intensive and average from 10 to 15 percent performance degradation for one wait state.

All execution times shown here represent 8 MHz operation. The 8086 results were measured at 5 MHz and extrapolated to achieve 8 MHz execution times.

6.2 Interest Rate Calculations

Routines were written in FORTRAN-86 to calculate the final value of a fund given the annual interest and the present value. It is assumed that the interest will be compounded daily, which requires the calculation of the yearly effective rate. This value, which is the equivalent annual interest if the interest were compounded daily, is determined by the following formula:

```
yer = (1 + (ir/np))**np - 1
```

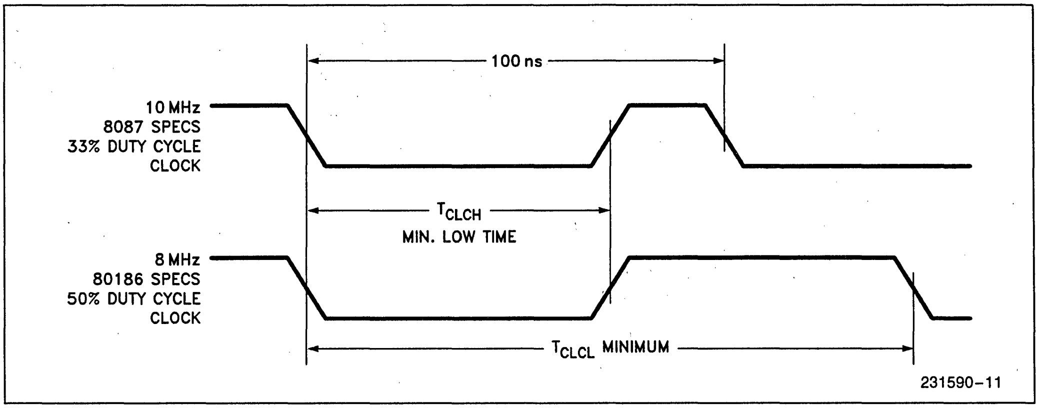


Figure 12. Clock Cycle Timing

where:

yer is the yearly effective rate
ir is the annual interest rate
np is the number of compounding periods per annum

Once the yer is determined, the final value of the fund is determined by the formula:

$$fv = (1 + yer) * pv$$

where:

pv is the present value
fv is the future value

Results are obtained using single-precision, double-precision, and temporary real precision operands when:

ir is set to 10% (0.1)
np is set to 365 (for daily compounding)
pv is set to $2,000,000

THE RESULTS:

	yer	Final Value
Single-Precision (32-bit)	10.514%	$2,210,287.50
Double-Precision (64-bit)	10.516%	$2,210,311.57
Temporary Real Precision	10.516%	$2,210,311.57

The difference between the final single-precision and double-precision values is $24.07; the difference in the final value between the double-precision and the temporary real precision is 0.000062 cents. Since the 8087 performs all internal calculations on 80-bit floating point numbers (temp real format), temporary real precision operations perform faster than single- or double-precision. No data conversions are required when loading or storing temporary real values in the 8087. Thus, for business applications, the double-precision computing of the 8087 is essential for accurate results, and the performance advantage of using the 8087 turns out to be as much as 100 times the equivalent software emulation program.

6.3 Matrix Multiply Benchmark Routine

A routine was written in FORTRAN-86 to compute the product of two matrices using a simple row/column inner-product method. Execution times were obtained for the multiplication of 32×32 matrices using double precision. The results of the benchmark are shown in Figure 14.

The results show the 8087 coprocessor systems performing from 23 to 31 times faster than the equivalent software emulation program. Both the 80188/87 and the 80186/87 systems outperform the 8086/87 system by 34 to 75 percent. This difference is mainly attributed to the fact that the matrix program largely consists of effective address calculations used in array accessing. The hardware effective address calculator of the 80186 and 80188 allow each array access to improve by as much as three times the 8086 effective address calculation.

6.4 Whetstone Benchmark Routine

The Whetstone benchmark program was developed by Harry Curnow for the Central Computer Agency of the British government. This benchmark has received high visibility in the scientific community as a measurement of main frame computer performance. It is a "synthetic" program. That is, it does not solve a real problem, but rather contains a mix of FORTRAN statements which reflect the frequency of such statements as measured in over 900 actual programs. The program computes a performance metric: "thousands of Whetstone instructions per second (KIPS)."

Simple variable and array addressing, fixed- and floating- point arithmetic, subroutine calls and parameter passing, and standard mathematical functions are performed in eleven separate modules or loops of a prescribed number of iterations.

Table 2. Interest Rate Benchmark Results

	8087 Software Emulator			8087 Coprocessor		
	80188	8086	80186	80188	8086	80186
Single Precision	70.3 ms	62.8 ms	43.4 ms	.70 ms	.66 ms	.61 ms
Double Precision	72.1 ms	62.9 ms	44.4 ms	.71 ms	.66 ms	.61 ms
Temp Real Precision	72.6 ms	63.0 ms	44.8 ms	.69 ms	.65 ms	.59 ms
Average	71.7 ms	62.9 ms	44.2 ms	.70 ms	.66 ms	.60 ms

The original coding of the Whetstone benchmark was written in Algol-60 and used single-precision values. It was rewritten in FORTRAN with single-precision values to exactly reflect the original intent. Another version was created using double-precision values. The results are shown in Table 3.

The results show the 8087 systems with the 80186 and 80188 outperforming the equivalent software emulation by 60 to 83 times. Additionally, the 80186 coupled with the 8087 outperformed the 8086/87 system by 22 percent.

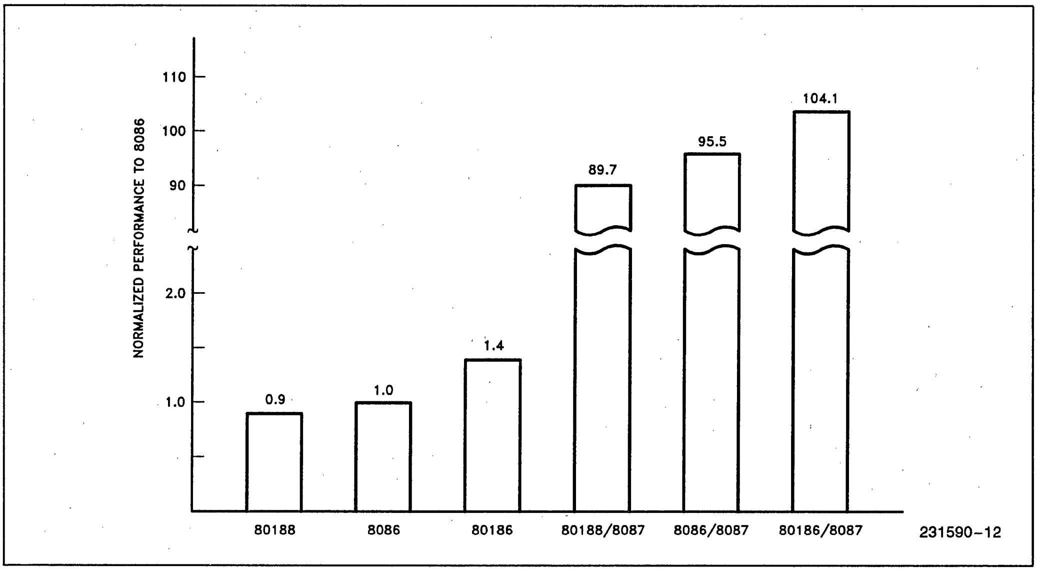


Figure 13. Interest Rate Benchmark Results

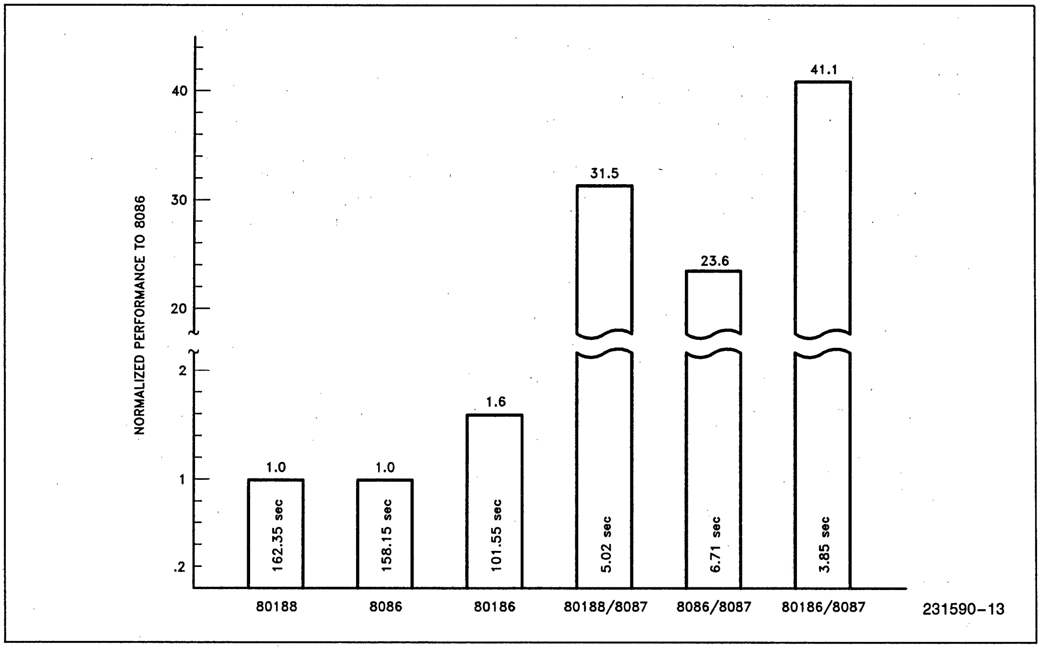


Figure 14. Double Precision Matrix Multiplication

Table 3. Whetstone Benchmark Results

Units = KIPS	8087 Software Emulator			8087 Coprocessor		
	80188	8086	80186	80188	8086	80186
Single Precision	2	2.3	3.3	165.8	178.0	197.6
Double Precision	2	2.2	3.2	151.7	152.0	185.2

6.5 Benchmark Conclusions

These benchmarks show that the 8087 Numeric Data Coprocessor, coupled with either the 80186 or the 80188, can increase the performance of a numeric application by 75 to 100 times the equivalent software emulation program.

Applications which require array accessing with effective address calculations will benefit even more by using the 80188 and 80186 as the host processor as compared to the 8086. The results of the matrix multiplication show both the 80188 and 80186 outperforming the 8086 by 34 and 75%, respectively, in an 8087 system. In general, an 80186/8087 system will offer a 10% to a 75% improvement over an equivalent 8086/8087 system, depending on the instruction mix.

7. CONCLUSION

For controller applications which require high performance in numerics and low system cost, the 16-bit 80186 or 8-bit 80188 coupled with the 8087 offers an ideal solution. The integrated features of the 80186 and 80188 offer a low system cost through reduced board space and a simplified production flow while the 8087 fulfills the performance requirements of numeric applications.

The 82188 IBC provides a straightforward, highly integrated solution to interfacing the 80188 or 80186 to the 8087. The bus control timings of the 82188 are compatible with the 80186 and 80188, allowing easy upgrades from existing designs. The 82188 features present a highly integrated solution to both new and old designs.

The coprocessing capabilities of the 8087 bring performance improvements of 75 to 100 times the equivalent 80186 or 80188 software emulation program and an 80186/8087 system will offer a 10% to a 75% improvement over an equivalent 8086/8087 system depending on the instruction mix.

In addition a growing base of high-level language support (FORTRAN, Pascal, C, Basic, PL/M, etc.) from Intel and numerous third-party software vendors facilitates the timely and efficient generation of application software.

REFERENCES:

82188 Data Sheet #231051
80186 Data Sheet #210451
80188 Data Sheet #210706
iAPX 86/88 80186/188 Users Manual
    Programmers Reference #210911
    Hardware Reference #210912
AP-113 "Getting Started with the
    Numeric Data Processor #207865

APPLICATION NOTE

AP-286

October 1986

80186/188 Interface to Intel Microcontrollers

PARVIZ KHODADADI
APPLICATIONS ENGINEER

Order Number: 231784-001

1.0 INTRODUCTION

Systems which require I/O processing and serial data transmission are very software intensive. The communication task and I/O operations consume a lot of the system's intelligence and software. In many conventional systems the central processing unit carries the burden of all the communication and I/O operations in addition to its main routines, resulting in a slow and inefficient system.

In an ideal system, tasks are divided among processors to increase performance and achieve flexibility. One attractive solution is the combination of the Intel highly integrated 80186 microprocessor and the Intel 8-bit microcontrollers such as the 80C51, 8052, or 8044. In such a system, the 80186 provides the processing power and the 1 Mbyte memory addressability, while the controller provides the intelligence for the I/O operations and data communication tasks. The 80186 runs application programs, performs the high level communication tasks, and provides the human interface. The microcontroller performs 8-bit math and single bit boolean operations, the low level communication tasks, and I/O processing.

This application note describes an efficient method of interfacing the 16-bit 80186 high integration microprocessor to the 80C51, 8052, or the microcontroller-based 8044 serial communication controller. The interface hardware shown in Figure 1.1, is very simple and may be implemented with a programmable logic device or a gate-array. The 80186 and the microcontroller may run asynchronously and at different speeds. With this technique data transfers up to 200 Kbytes per second can be achieved between a 12 MHz microcontroller and an 8 MHz 80186.

The 8-bit 80188 high integration microprocessor can also be used with the same interface technique. The performance of the interface is the same since an 8-bit bus is used.

Interface to the 8044, 80C51, and the 8052 is identical because they have identical pinouts (some pins have alternate functions). As an example, the software procedures for the 8044/80186 interface, which is the building block for the application driver, is supplied in this Application Note.

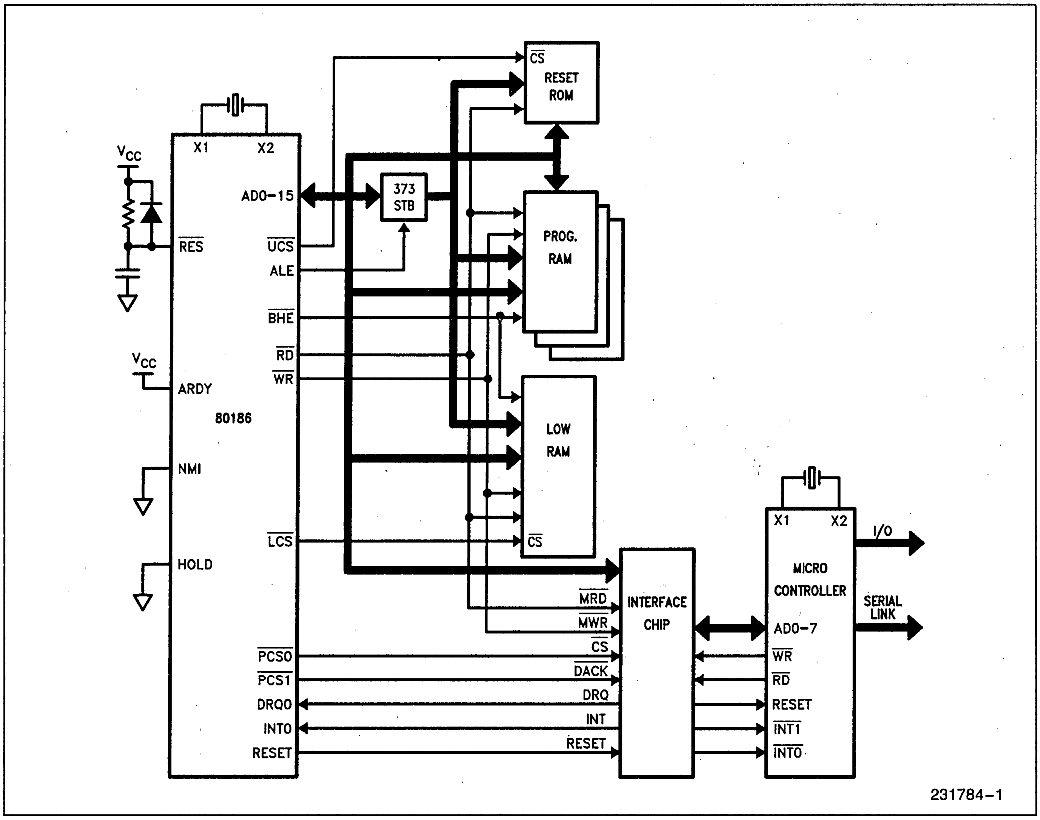


Figure 1.1. 80186/Microcontroller Based System

1.1 System Overview

The 80186 and the microcontrollers are processors. They each access memory and have address/data, read, and write signals. There are three common ways to interface multiple processors together:

1) First In First Out (FIFO)

2) Dual Port RAM (DPRAM)

3) Slave Port

The FIFO interface, compared to DPRAM, requires less TTL and is easier to interface; however, FIFOs are expensive. The DPRAM interface is also expensive and even more complex. When DPRAM is used, the address/data lines of each processor must be buffered, and hardware logic is needed to arbitrate access to DPRAM. The slave port interface given here is cheaper and easier than both FIFO and DPRAM alternatives.

The 80186 processor, when interfaced to this circuit, views the microcontroller as a peripheral chip with 8-bit data bus and no address lines (see Figure 1.1). It can read status and send commands to the microcontroller at any time. The microcontroller becomes a slave coprocessor while keeping its processing power and serial communication capabilities.

The microcontrollers, with the interface hardware, have a high level command interface like many other data communication peripherals. For example, the 80186 can send the microcontroller commands such as Transmit or Configure. This means the designer does not have to write low level software to perform these tasks, and it offloads the 80186 to serve other functions in the application.

1.2 Application Examples:

The combination of the 80186 and a microcontroller basically provides all the functions that are needed in a system: a 16-bit CPU, 8-bit CPU, DMA controller, I/O ports, and a serial port. The 80C51 and the 8052 have an on-chip asynchronous channel, while the 8044 has an intelligent SDLC serial channel. In addition, many other functions such as timers, counters, and interrupt controllers are integrated in both the 80186 and the microcontrollers.

Applications of the system described above are in the area of robotics, data communication networks, or serial communication backplanes. A typical example is copiers. Different segments of the copy machine like the motor, paper feed, diagnostics, and error/warning displays are all controlled by microcontrollers. Each segment receives orders from and replies to the central processor which consists of the 80186 interfaced with a microcontroller.

Another common application is in the area of process controllers. An example is a central control unit for a multiple story building which controls the heating, cooling, and lighting of each room in each floor. In each room a microcontroller performs the above functions based on the orders received from the central processor. Depending on the throughput and type of the serial communication required, the 8044 or the 80C51 (8052) may be selected for the application.

2.0 OVERVIEW OF THE 80186, 80C51, 8052, AND 8044

This section briefly discusses the features of the microcontrollers and the 80186. For more information about these products please refer to the Intel Microcontroller and Microsystem components hand-books. Readers familiar with the above products may skip this section.

2.1 The 80186 Internal Architecture

The 80186 contains an enhanced version of Intel's popular 8086 CPU integrated with many other features common to most systems (Figure 2.1). The 16-bit CPU can access up to 1 Mbyte of memory and execute instructions faster than the 8086. With speed selection of 8, 10, and 12.5 MHz, this highly integrated product is the most popular 16-bit microprocessor for embedded control applications.

The on-chip DMA controller has two channels which can each be shared by multiple devices. Each channel is capable of transferring data up to 3.12 Mbytes per second (12.5 MHz speed). It offers the choice of byte or word transfer. It can be programmed to perform a burst transfer of a block of data, transfer data per specified time interval, or transfer data per external request.

The on-chip interrupt controller responds to both external interrupts and interrupts requested by the on-chip peripherals such as the timers and the DMA channels. It can be configured to generate interrupt vector addresses internally like the microcontrollers or externally like the popular 8259 interrupt controller. It can be configured to be a slave controller to an external interrupt controller (iRMX 86 mode) or be master for one or two 8259s which in turn may be masters for up to 8 more 8259s. When configured in master mode, each channel can support up to 64 external interrupts (128 total).

Three 16-bit timers are also integrated on the chip. Timer 0 and timer 1 can be configured to be 16-bit counters and count external events. If configured as timers, they can be started by software or by an external event. Timer 0 and 1 each contain a timer output pin. Transitions on these pins occur when the timers reach one of the two possible maximum counts. Timer

2 can be used as a prescaler for timer 0 and 1 or can be used to generate DMA requests to the on-chip DMA channel.

Finally, the integrated clock generator, the wait state generator, and the chip select logic reduce the external logic necessary to build a processing system.

2.2 The MCS-51 Internal Architecture

The 80C51BH, as shown in Figure 2.2, consists of an 8-bit CPU which can access up to 64 Kbytes of data memory (RAM) and 64 Kbytes of program memory (ROM). In addition, 4 Kbytes of ROM and 128 bytes of RAM are built onto the chip.

The on-chip interrupt controller supports five interrupts with two priority levels. There are two timers integrated in the 80C51. Timer 0 and 1 can be configured as 8-bit or 16-bit timers or event counters.

Finally the integrated full duplex asynchronous serial channel provides the human interface or communication capability with other microcontrollers. The UART supports data rates up to 500 kHz (with 15 MHz crystal) and can distinguish between address bytes and data bytes.

The 8052 has the same features as the 80C51 except it has 8 Kbytes of on-chip ROM and 256 bytes of on-chip RAM. In addition the 8052 has another timer which may be configured as the baud rate generator for the serial port.

2.3 The 8044 Internal Architecture

The 8044 has all the features of the 80C51. In addition the on-chip RAM size is increased to 192 bytes and an intelligent HDLC/SDLC serial channel (SIU) replaces the 80C51 serial port (see Figure 2.3). It supports data rates up to 2.4 Mbps when an external clock is used and 375 Kbps when the clock is extracted from the data line. The serial port can be used in half duplex point to point, multipoint, or one-way loop configurations.

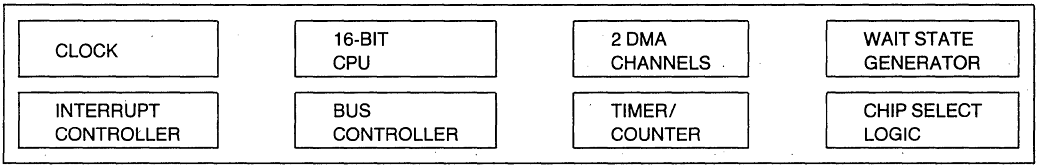


Figure 2.1. 80186 Block Diagram

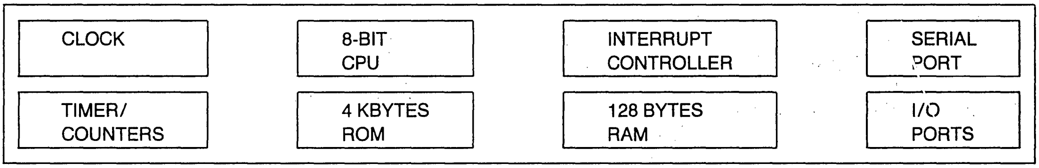


Figure 2.2. 80C51 Block Diagram

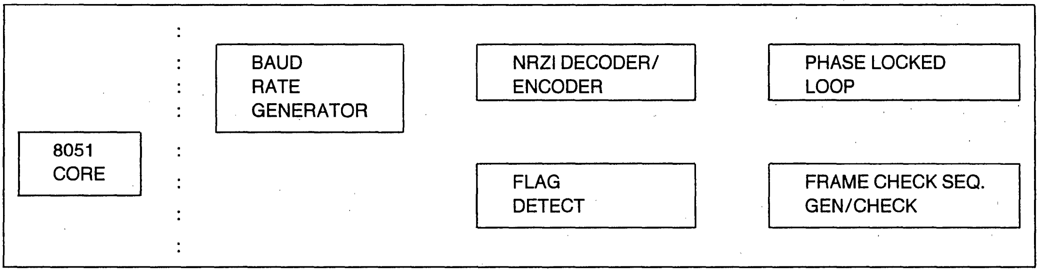


Figure 2.3. 8044 Block Diagram

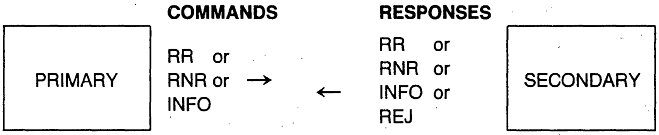


Figure 2.4. 8044 Automatic Response to SDLC Commands

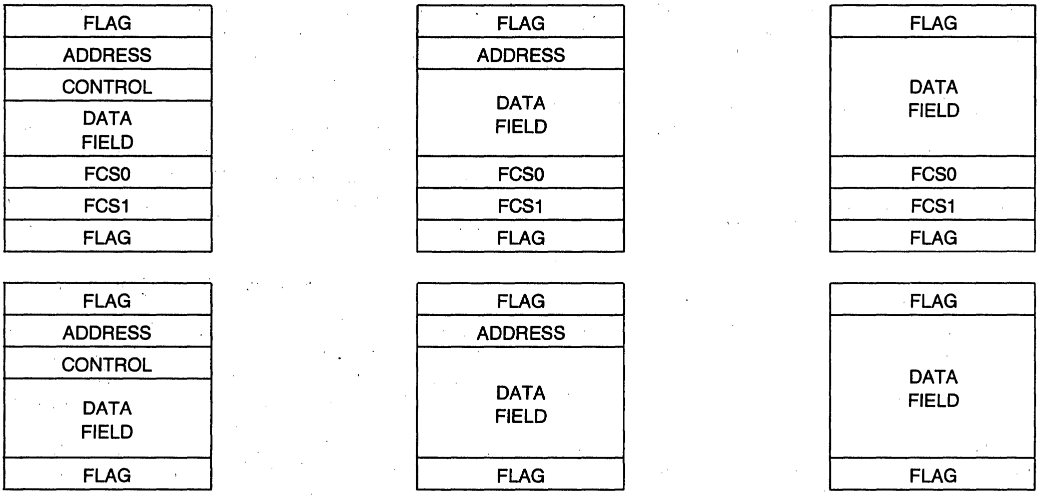


Figure 2.5. The 8044 Frame Formats

The SIU is called an intelligent channel because it responds to some SDLC commands automatically without the CPU intervention when it is set in auto mode. These automatic responses substantially reduce the communication software. Figure 2.4 gives the commands and the automatic responses.

The 8044 supports many types of frames including the standard SDLC format. Figure 2.5 shows the types of frames the 8044 can transmit and receive. If a format with an address byte is chosen, the 8044 performs address filtering during reception and transmits the contents of the station address register during transmission automatically. If a format with FCS bytes is chosen, the 8044 performs Cyclic Redundancy Check (CRC) during reception and calculates the FCS bytes during transmission of a frame in hardware. Two preamble bytes (PFS) may optionally be added to the frames. Formats that include the station address and the control byte are supported both in the auto and flexible modes.

3.0 80186/MICROCONTROLLER INTERACTION

The 80186 communicates with the microcontroller (8044, 80C51 or 8052) through the system's memory and the Command/Data and Status registers. The CPU creates a data structure in the memory, programs the DMA controller with the start address and byte count of the block, and issues a command to the microcontroller. A hypothetical block diagram of a microcontroller when used with the interface hardware is given in Figure 3.1.

Chip select and interrupt lines are used to communicate between the microcontroller and the host. The interrupt is used by the microcontroller to draw the 80186's attention. The Chip Select is used by the 80186 to draw the microcontroller's attention to a new command.

There are two kinds of transfers over the bus: Command/Status and data transfers. Command/Status transfers are always performed by the CPU. Data transfers are requested by the microcontroller and are typically performed by the DMA controller.

The CPU writes commands using CS and WR signals and interrupts the microcontroller. The microcontroller reads the command, decodes it and performs the necessary actions. The CPU reads the status register using CS and RD signals (see Figure 4.1).

To initiate a commnad like TRANSMIT or CONFIGURE, a write operation to the microcontroller is issued by the CPU. A read operation from the CPU gives the status of the microcontroller. Section 5 discusses details on these commands and the status.

Any parameters or data associated with the command are transferred between the system memory and the microcontroller using DMA. The 80186 prepares a data block in memory. Its first byte specifies the length of the rest of the block. The rest of the block is the information field. The CPU programs the DMA controller with the start address of the block, length of the block and other control information and then issues the command to the microcontroller.

When the microcontroller requires access to the memory for parameter or data transfer, it activates the 80186 DMA request line and uses the DMA controller to achieve the data transfer. Upon completion of an operation, the microcontroller interrupts the 80186. The CPU then reads results of the operation and status of the microcontroller.

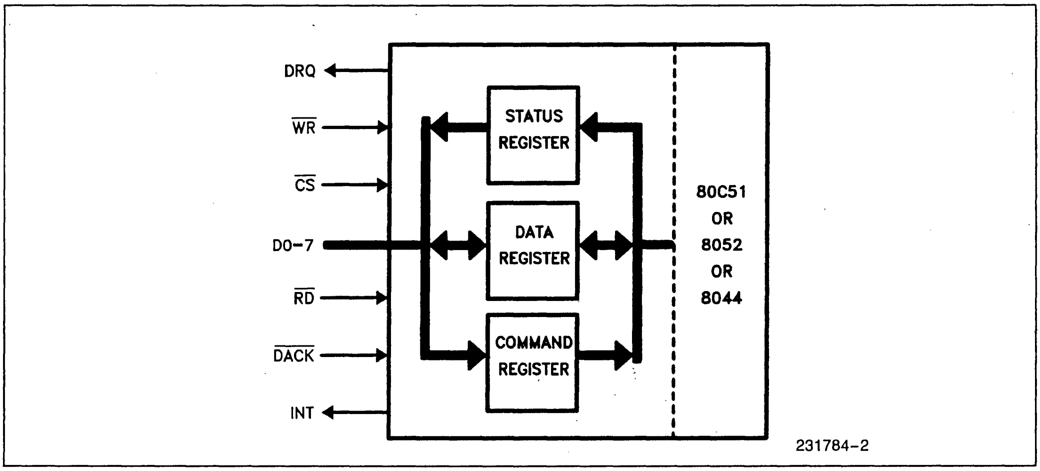


Figure 3.1. Microcontroller Plus the Interface Hardware Block Diagram

4.0 SYSTEM INTERFACE

There are two kinds of transfers over the bus: command/status and data transfers. The command/status transfers are always initiated and performed by the 80186. The data transfers are requested by the microcontroller using the DMA request (DRQ) line. In relatively slow systems the 80186 might also perform the data transfers. In that case, the request from the microcontroller will serve as an interrupt to the CPU. This mode of operation depends on the serial data rate.

The system interface performs command/status transfers, data/parameter transfers, and interrupts. This section describes the interface between the 80186 and a microcontroller shown in Figure 1.1. Section 6 describes the interface hardware.

4.1 Command/Status Transfers

The 80186 controls the microcontroller by writing into the command/data register and reading from the status register. The CPU writes a command by activating the chip select (PCS0), putting the command onto the data bus, and activating the WR signal. The command byte is latched into the command/data register, and the microcontroller is interrupted. In the interrupt service routine, the microcontroller reads the command byte from the command/data register, decodes the command byte, and activates the DRQ for data or parameter transfer if the decoded command requires such transfer.

At the end of parameter transfer the microcontroller updates the status register and interrupts the 80186.

4.2 Data/Parameter Transfer

Data/parameter transfers are controlled by a pair of REQUEST/ACKNOWLEDGE lines: DMA Request line (DRQ) and DMA Acknowledge line (DACK). Data and parameters are transferred via the Command/Data register to or from memory.

In order to request a transfer from memory, the microcontroller activates the DRQ pin. The DRQ signal goes active after a read operation by the microcontroller. In response, the 80186 DMA controller performs a byte transfer from the memory to the Command/Data register. Data is transferred on the bus and written into the Command/Data register on the rising edge of the 80186 WR signal (MWR), which is activated by the DMA controller. Figure 4.2 shows the write timing.

In order to request a transfer to memory, the microcontroller activates the DRQ signal and outputs the data into the Command/Data latch. When the microcontroller WR signal goes active, DRQ is set. In response, the DMA performs the data transfer and resets the DRQ signal. Figure 4.3 shows the read timing.

4.3 Interrupt

The microcontroller reports on completion of an event by updating the status register and raising the interrupt signal assuming this signal is initially low. The interrupt is cleared by the command from the CPU where the INTERRUPT ACKNOWLEDGE bit is set (MD7). The INTA bit is the most significant bit of the command byte. Figure 4.4 and 4.5 show the interrupt timing. Note that it is the responsibility of the CPU to clear the interrupt in order to prevent a deadlock.

80186 Pin Name			Function
$\overline{\text{CS}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	
1 0	X 1	X 1	No Transfer to/from Command/Status
0	0	0	Illegal
0	0	1	Read from Status Register
0	1	0	Write to Command/Data Register
$\overline{\text{DACK}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	
1 0	X 1	X 1	No Transfer
0	0	0	Illegal
0	0	1	Data Read from DMA Channel
0	1	0	Data Write to DMA Channel

NOTE:
Only one of CS, DACK may be active at any time.

Figure 4.1. Data Bus Control Signals and Their Functions

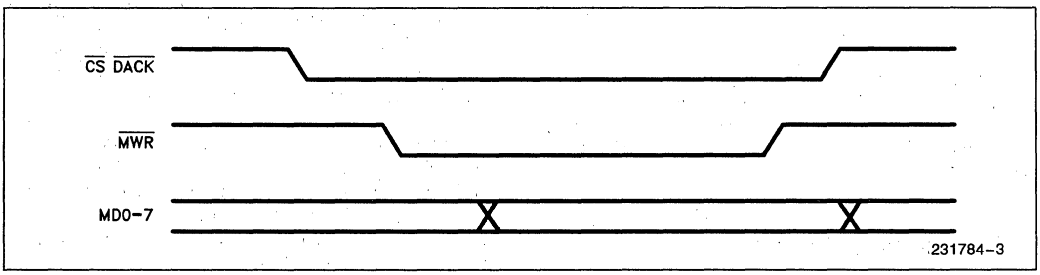


Figure 4.2. Write Timing

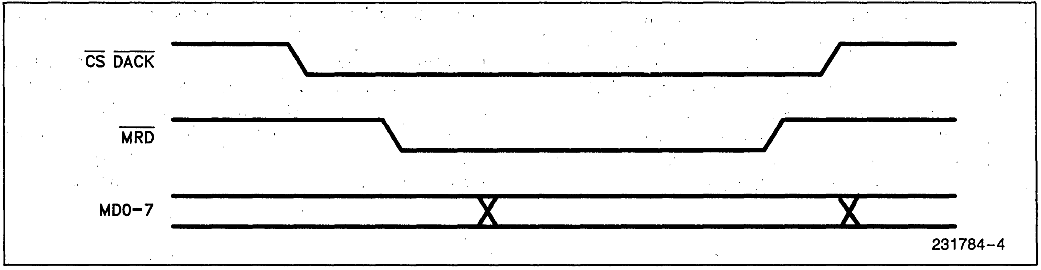


Figure 4.3. Read Timing

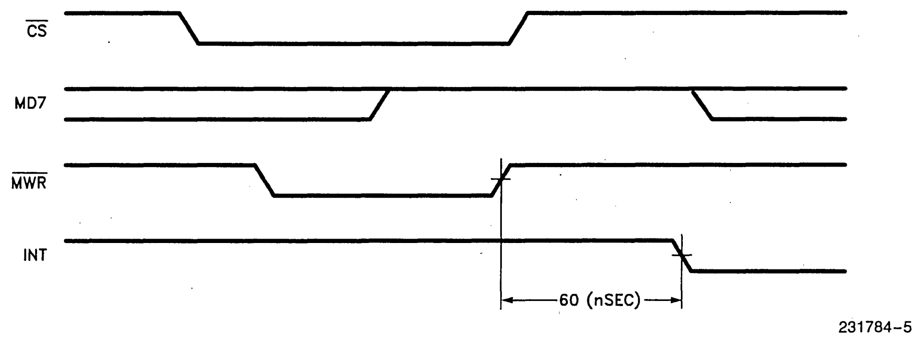


Figure 4.4. Interrupt Timing (Going Inactive)

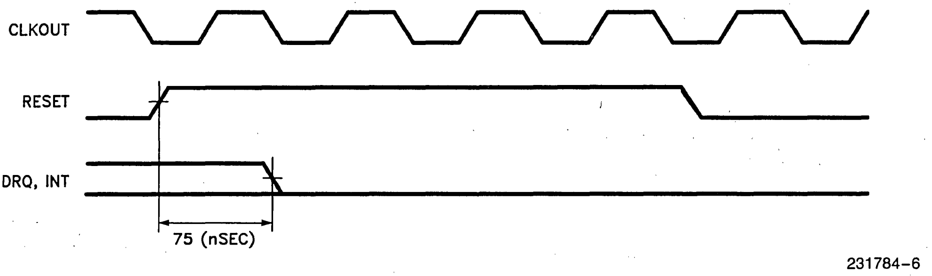


Figure 4.5. Reset Timing

5.0 COMMANDS AND STATUS

This section specifies the format of the commands and status. The commands and status given here are similar to most common coprocessors and data communication peripherals (e.g., the 82588 and 82586). The user may add more commands or redefine the formats for his/her own specific application.

5.1 Commands

A command is given to the microcontroller by writing it into the Command/Data register and interrupting the microcontroller. The command can be issued at any time; but in case it is not accepted, the operation is treated like a NOP and will be ignored (although the INT will be updated).

Format:

7	6	5	4	3	2	1	0
INTA	X	X	X	OPERATION			

5.1.1 ACKNOWLEDGING INTERRUPT (BIT 7)

The INTA bit, if set, causes the interrupt hardware signal and the interrupt bit to be cleared. This is the only way to clear the interrupt bit and reset the 80186 interrupt signal other than by a hardware reset.

5.1.2 OPERATIONS (BITS 0–3)

The OPERATION field initiates a specific operation. The microcontroller executes the following commands in software:

NOP
ABORT
TRANSMIT*
CONFIGURE*
DUMP*
RECEIVE*
TRA-DISABLE
REC-DISABLE
*Requires DMA operation.

The above operations except ABORT are executed only when the microcontroller is not executing any other operation. Abort is accepted only when the CPU is performing a DMA operation.

Operations that require parameter transfer (e.g., CONFIGURE and DUMP) or data transfer (e.g., TRANSMIT and RECEIVE) are called parametric operations. The remaining are called non-parametric operations.

An operation is initiated by writing into the command register. This causes the microcontroller to execute the command decode instructions. Some of the operations cause the microcontroller to read parameters from memory. The parameters are organized in a block that starts with an 8-bit byte count. The byte count specifies the length of the rest of the block. Before beginning the operation, the DMA pointer of the DMA channel must point to the byte count. There is no restriction on the memory structure of the parameter block as long as the microcontroller receives the next byte of the block for every DMA request it generates. Transferring the bytes is the job of the 80186 DMA controller.

The microcontroller requests the byte-count and determines the length of the parameter block. It then requests the parameters.

Upon completion of the operation, (when interrupt is low) the microcontroller updates the status, raises the interrupt signal, and goes idle.

NOP

This operation does not affect the microcontroller. It has no parameters and no results.

ABORT

This operation attempts to abort the completion of an operation under execution. It is valid for CONFIGURE, TRANSMIT, DUMP, and RECEIVE. It is ignored for any of the above if transfer of parameters has already been accomplished. The microcontroller, upon reception of the ABORT command, stops the DMA operation and issues an Execution-Aborted interrupt.

TRANSMIT

This operation transmits one message. A message may be transmitted as an SDLC frame by the 8044, or in ASYNC protocol by the 80C51 or the 8052 serial port.

Figure 5.1 shows the format of the Transmit block. A typical transmit operation parameter block includes the destination address and the control byte in the information field. As an example, see the 8044 transmit block in Figure 7.2.

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">BYTE COUNT</td></tr>
<tr><td colspan="8">FIRST INFO BYTE</td></tr>
<tr><td colspan="8"></td></tr>
<tr><td colspan="8">LAST INFO BYTE</td></tr>
</table>

Figure 5.1. Format of Transmit Block

The transmit operation will either complete the execution or be aborted by a specific ABORT operation. A Transmit-Done or Execution-Aborted interrupt is issued upon completion of this operation.

CONFIGURE

This operation configures the microcontroller's internal registers. The length and the part of the configuration block that is modified are determined by the first two bytes of the command parameter (see Figure 5.2). The FIRST BYTE specifies the first register in the configure block that will be configured, and the BYTE COUNT specifies the number of registers that will be configured starting with the FIRST BYTE. For example, if the FIRST BYTE is 1 and the BYTE COUNT is the length of the configure block, then all of the registers are updated. If FIRST BYTE is 4 and BYTE COUNT is 2, then only the fourth register in the configure block is updated. Minimum byte count is 2.

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">BYTE COUNT</td></tr>
<tr><td colspan="8">FIRST BYTE</td></tr>
<tr><td colspan="8">FIRST REGISTER</td></tr>
<tr><td colspan="8"></td></tr>
<tr><td colspan="8">LAST REGISTER</td></tr>
</table>

Figure 5.2. Format of Configure Block

A Configure-Done interrupt is issued when the operation is done unless ABORT was issued during the DMA operation.

DUMP

This operation causes dumping of a set of microcontroller internal registers to system memory. Figure 7.4 shows the format of the 8044 DUMP block.

The DUMP operation will either complete the execution or be aborted by a specific ABORT operation. A Dump-Done or Execution-Aborted interrupt is issued upon completion of this operation.

RECEIVE

This operation enables the reception of frames. It is ignored if the microcontroller's serial channel is already in reception mode.

The serial port receives only frames that pass the address filtering. The microcontroller transfers the received information and the byte count to the system memory using DMA. The completion of frame reception causes a Receive-Done event.

REC-DISABLE

This operation causes reception to be disabled. If transfer of data to the 80186 memory has already begun, then it is treated like the ABORT command. This operation has no parameters. REC-DISABLE is accepted only when the microcontroller's serial port is in receive mode.

TRA-DISABLE

This operation causes the transmission process to be aborted. If the microcontroller is fetching data from 80186 memory, then it is treated like the ABORT command. This operation has no parameters. It is accepted only when the serial port is in transmit mode.

5.1.3 ILLEGAL COMMANDS

Parametric and non-parametric commands except ABORT will be rejected (interrupt will not be set) if the microcontroller is already executing a command.

ABORT is rejected if issued when the microcontroller is not requesting DMA operation, or a non-Parametric execution is performed, or transfer of parameters/data has already been accomplished.

DMA operations shall not be aborted by any non-parametric or parametric command except by the ABORT command.

REC-DISABLE and TRA-DISABLE will not be accepted if the serial channel is idle.

5.2 Status

The microcontroller provides the information about the last operation that was executed, via the status register.

The microcontroller reports on these events by updating a status register and raising the INTERRUPT signal. Information from the status register is valid provided the interrupt signal is high or bit 0 of the status being read is set.

Format:

7	6	5	4 3 2	1	0
CTS*	RTS*	E	EVENT	DMA	INT

*8044 only

5.2.1 INTERRUPT (BIT 0)

The interrupt bit is set together with the hardware interrupt signal. Setting the INT bit indicates the occurrence of an event. This bit is cleared by any command whose INTA bit is set. Status is valid only when this bit is set.

5.2.2 DMA OPERATION (BIT 1)

The DMA bit, when set, indicates that a DMA operation is in progress. This bit is set if the commnad received by the microcontroller requires data or parameter transfer. If this bit is clear, DRQ will be inactive. The DMA bit, when cleared, indicates the completion of a DMA operation.

5.2.3 ERROR (BIT 5)

The E bit, if set, indicates that the event generated for the operation that was completed contains a warning, or the operation was not accepted.

5.2.4 REQUEST TO SEND (BIT 6)

The RTS bit, if clear, indicates that the serial channel is requesting a transmission.

5.2.5 CLEAR TO SEND (BIT 7)

The CTS bit indicates that, if the RTS bit is clear, the serial port is active and transmitting a frame.

5.2.6 EVENT (BITS 2–4)

The event field specifies why the microcontroller needs the attention of the 80186.

The following events may occur:

CONFIGURE-DONE
TRANSMIT-DONE
DUMP-DONE
RECEIVE-DONE
RECEPTION-DISABLED
TRANSMISSION-DISABLED
EXECUTION-ABORTED

CONFIGURE-DONE

This event indicates the completion of a CONFIGURE operation.

TRANSMIT-DONE

This event indicates the completion of the TRANSMIT operation.

If the E bit is set, it indicates that the transmit buffer was already full.

DUMP-DONE

This event indicates that the DUMP operation is completed.

RECEIVE-DONE

This event indicates that a frame has been received and stored in memory.

The format of the received message is indicated in Figure 5.3.

7 6 5 4 3 2 1 0

FIRST INFO BYTE
LAST INFO BYTE
RECEIVED BYTE COUNT

Figure 5.3. Format of Receive Block

Following the byte count, a few more bytes relating to the received frame such as the source address and the control byte may be transferred to the system memory using DMA. As an example, see the 8044 receive block in Figure 7.3.

Note that the format of a frame received by the microcontroller serial channel is configured by the CONFIGURE command.

If the E bit is set, buffer overrun has occurred.

RECEPTION-DISABLED

This event is issued as a result of a RCV-DISABLE operation that causes part of a frame to be disabled.

If the E bit is set, the serial port was already disabled, and the RCV-DISABLE is not accepted.

TRANSMISSION-DISABLED

This event is issued as a result of a TRA-DISABLE operation that causes transmission of a frame to be disabled.

The E bit, if set, indicates that the TRA-DISABLE operation was not accepted since the serial port was already idle, or transmission of a frame has already been accomplished.

EXECUTION-ABORTED

This event indicates that the execution of the last operation was aborted by the ABORT command.

If the E bit is set, ABORT was issued when the microcontroller was not executing any commands.

6.0 HARDWARE DESCRIPTION

The interface hardware shown in Figures 6.1 and 6.2 are identical. The difference is the status register. In Figure 6.2, an external latch is used to latch the status byte. This hardware is recommended if an extra I/O port on the microcontroller is required for some other applications, or external program and data memory is required for the microcontroller. The hardware shown in Figure 6.1 makes use of one of the microcontroller's I/O ports (Port 1) to latch the status to minimize hardware. The discussion of Sections 1 through 5 apply to both schematics.

6.1 Reset

After an 80186 hardware reset, the microcontroller is also reset. The on-chip registers are initialized as explained in the Intel Microcontroller Handbook. The reset signal also clears the 80186 interrupt and the microcontroller interrupt signals by resetting FF3 (Flip-Flop 3) and FF2 (Flip-Flop 2). Figure 4.5 shows the RESET timing.

6.2 Sending Commands

A bidirectional latched transceiver (74ALS646) is used for the Command/Data register. When the 80186 writes a command to the Command/Data register, it interrupts the microcontroller. The interrupt is generated only when bit 7 (INTA) of the command byte is set. When the 80186 PCS0 and WR signals go active to write the command, FF2 will be set and FF3 will be cleared. The output of FF3 is the interrupt to the 80186 and the INT status bit. The INT bit is cleared immediately to indicate that the status is no longer valid. The output of FF2 is the interrupt to the microcontroller. A high to low transition on this line will interrupt the microcontroller. The interrupt signal will be cleared as soon as the microcontroller reads the command from the Command/Data register.

6.3 DMA Transfers

In the interrupt service routine the command is decoded. If it requires a DMA transfer, the microcontroller sets the DMA bit of the status register which activates the DMA request signal. DRQ active causes the 80186 on-chip DMA to perform a fetch and a deposit bus cycle. The first DMA cycle clears the DRQ signal (FF1 is cleared). When the microcontroller performs a read or write operation, the output of the FF1 will be set, and DRQ goes active again.

The DMA controller transfers a byte from system memory to the Command/Data register. Data is latched when the 80186 PCS1 and WR signals go active. PCS1 and WR active also clear FF1. The microcontroller monitors the output of FF1 by polling the P3.3 pin. When FF1 is cleared the microcontroller reads the byte from the Command/Data register. The P3.3 pin is also the interrupt pin. If a slow rate of transfer is acceptable, every DMA transfer can be interrupt driven to allow the microcontroller to perform other tasks.

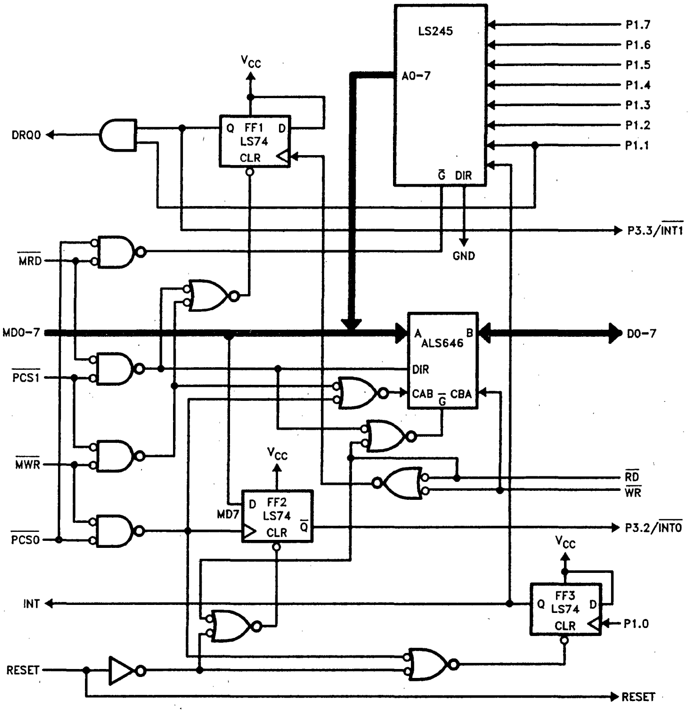
The DMA controller transfers a byte from the Command/Data register to system memory by activating the 80186 PCS1 and RD signals. PCS1 and RD active also clear FF1. When FF1 is cleared the microcontroller writes the next byte to the Command/Data register.

When all the data is transferred, the microcontroller clears the DMA status bit to disable DRQ. It then updates the status, sets the INT bit, and interrupts the 80186.

If the interface hardware in Figure 6.1 is used P1.1 is the DMA status bit and P1.0 is the INT bit. The microcontroller enables or disables them by writing to port 1. In Figure 6.2, DRQ or INT is disabled or enabled by writing to the 74LS374 status register. Note that the INT status bit is cleared by the hardware when the 80186 writes a command.

6.4 Reading Status

The command is written and the status is read with the same chip select (PCS0), although the status is read through the 74LS245 transceiver and the command is written to the Command/Data register.



231784-7

Figure 6.1. Hardware Interface (Port 1 is the Status Register)

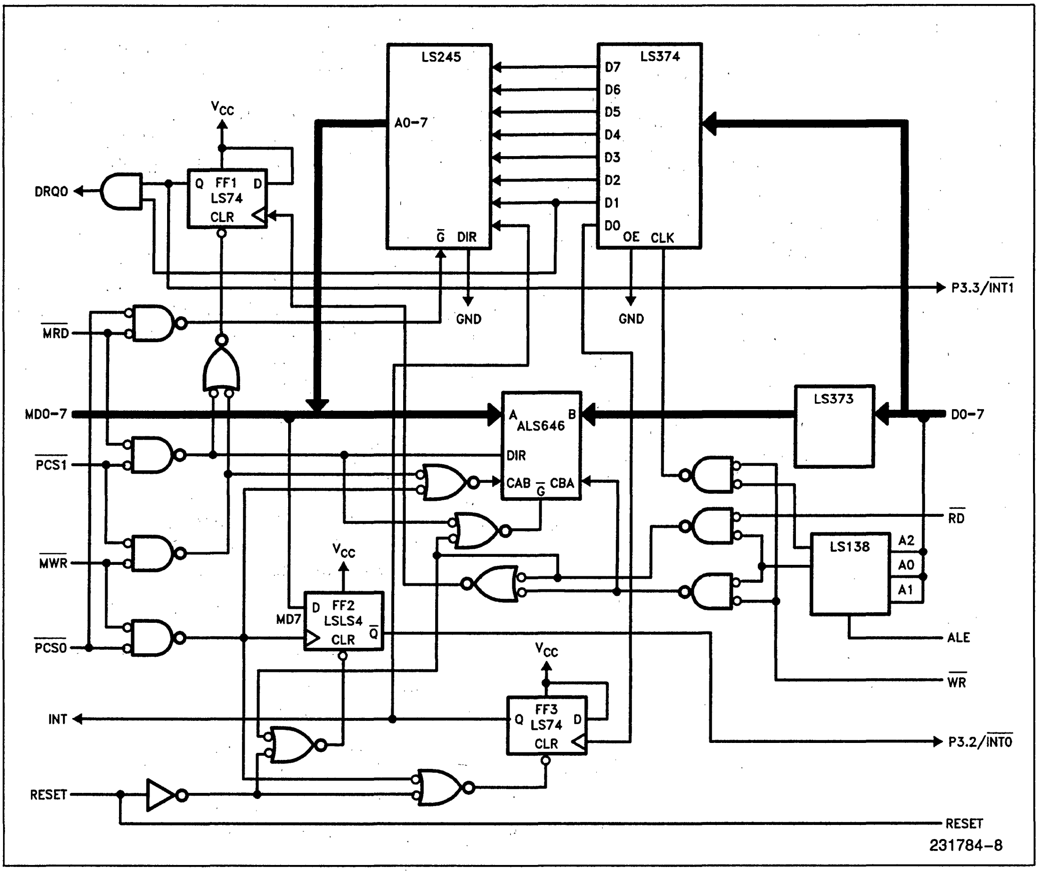


Figure 6.2. Hardware Interface

The microcontroller updates the status byte whenever a change occurs in the status and outputs the result to the status register. In order to read status, the 80186 activates the PCS0 line, and then activates the RD line. The contents of the status are put on the data bus, through the 74LS245 transceiver.

For systems that require two DMA channels, a second pair of DRQ1/DACK1 signals may easily be added to the hardware. In that case one of the status bits (DMA2) ANDed with the output of FF1 will serve as the second DMA request signal (DRQ1). DACK1 can be generated with the 80186 PCS2.

7.0 8044/80186 INTERFACE

This section shows how to make use of the status and commands described in section 5 and the hardware given in Figure 6.1 to interface the 80186 with the 8044. The 8044 code to implement these functions is shown in Appendix A.

7.1 Configuring the 8044

This operation configures the 8044 registers. The format of the configure block is shown in Figure 7.1. The part of the configuration block that is modified is determined by the first two bytes of the command parameter. The FIRST BYTE specifies the first register in the configure block that will be configured, and the BYTE COUNT specifies the number of registers that will be configured starting with the FIRST BYTE. For example, if the FIRST BYTE is 1 and the BYTE COUNT is 13, then all of the registers are updated. If FIRST BYTE is 4 and BYTE COUNT is 2, then transmit buffer start register is configured.

The configure command performs the following: 1) configures the interrupts and assigns their priorities; 2) assigns the start address and length of the transmit and receive buffers; 3) sets the station address; 4) sets the clock option and the frame format.

For other microcontrollers the format of the configure block should be modified accordingly. For example, the 80C51 serial port registers (e.g., T2CON, SCON) replace the 8044 SIU registers in the configure block.

7	6	5	4	3	2	1	0
BYTE COUNT							
FIRST BYTE							
STS							
SMD							
STATION ADDRESS							
TRANSMIT BUFFER START							
TRANSMIT BUFFER LENGTH							
RECEIVE BUFFER START							
RECEIVE BUFFER LENGTH							
INTERRUPT PRIORITY							
INTERRUPT ENABLE							
TIMER/COUNTER MODE							
TIMER/COUNTER MODE							
PROCESSOR STATUS WORD							

Figure 7.1. Format of the 8044 Configure Block

7.2 Transmitting a Message with the 8044

A message is a block of data which represents a text file or a set of instructions for a remote node or an application program which resides on the 8044 program memory. A message can be a frame (packet) by itself or can be comprised of multiple frames. An SDLC frame is the smallest block of data that the 8044 transmits. The 8044 can receive commands from the 80186 to transmit and receive messages. The 8044 on-chip CPU can be programmed to divide messages into frames if necessary. Maximum frame size is limited by the transmit or receive buffer.

To transmit a message, the 80186 prepares a transmit data block in memory as shown in Figure 7.2. Its first byte specifies the length of the rest of the block. The next two bytes specify the destination address of the node the message is being sent to and the control byte of the message. The 80186 programs the DMA controller with the start address of the block, length of the block and other control information and then issues the Transmit command to the 8044.

Upon receiving the command, the 8044 fetches the first byte of the block using DMA to determine the length of the rest of the block. It then fetches the destination address and the control byte using DMA.

The 8044 fetches the rest of the message into the on-chip transmit buffer. The size and location of the transmit buffer in the on-chip RAM is configured with the Configure command. The 8044 CPU then enables the Serial Interface Unit (SIU) to transmit the data as an SDLC frame. The SIU sends out the opening flag, the station address, the SDLC control byte, and the contents of transmit buffer. It then transmits the calculated CRC bytes and the closing flag. The 8044 CPU and the SIU operate concurrently. The CPU can fetch bytes from system memory or execute a command such as TRANSMIT-DISABLE while the SIU is active.

Upon completion of transmission, the SIU updates the internal registers and interrupts the 8044 CPU. The 8044 then updates the status and interrupts the 80186. Note that baud rate generation, zero bit insertion, NRZI encoding, and CRC calculation are automatically done by the SIU.

7.3 Receiving a Message with the 8044

To receive a message, the 80186 allocates a block of memory to store the message. It sets the DMA channel and sends the Receive command to the 8044.

Upon reception of the command, the 8044 enables its serial channel. The 8044 receives and passes to memory all frames whose address matches the individual or broadcast address and passes the CRC test.

The SIU performs NRZI decoding and zero bit deletion, then stores the information field of the received frame in the on-chip receive buffer. At the end of reception, the CPU requests the transfer of data bytes to 80186 memory using DMA. After transferring all the bytes, the 8044 transfers the data length, source address, and control byte of the received frame to the memory (see Figure 7.3). Upon completion of the transfers, the 8044 updates the status register and raises the interrupt signal to inform the 80186.

If the SIU is not ready when the first byte of the frame arrives, then the whole frame is ignored. Disabling reception after the first byte was passed to memory causes the rest of the frame to be ignored and an interrupt with Receive-Aborted event to be issued.

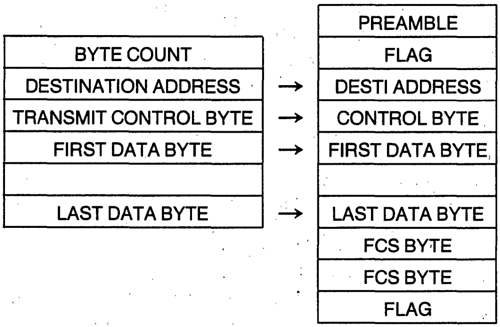


Figure 7.2. The 8044 Transmit Frame Structure and Location of Data Element in System Memory

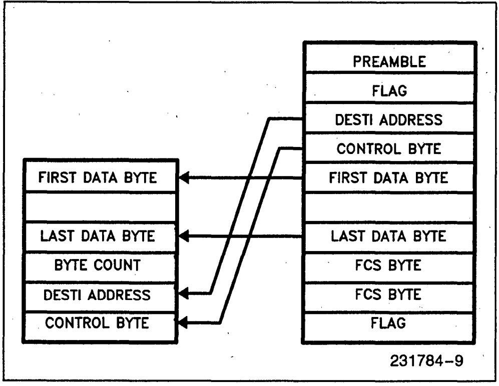


Figure 7.3. The 8044 Receive Frame Structure and Location of Received Data Element in System Memory

7.4 Dumping the 8044 Registers

Upon reception of the Dump command, the 8044 transfers the contents of its internal registers to the system memory (See Figure 7.4).

7 6 5 4 3 2 1 0
STS REG.
SMD REG.
STAD REG.
TBS REG.
TBL REG.
TCB REG.
RBS REG.
RBL REG.
RCB REG.
RFL REG.
PSW REG.
IP REG.
IE REG.
TMOD REG.
TCON REG.

Figure 7.4. Format of the 8044 Dumped Registers

7.5 Aborting an Operation

To abort a DMA operation, the 80186 sends an Abort command to the Command/Data latch and interrupts the 8044. During a DMA operation, the 8044 puts the external interrupt to high priority; therefore, the Abort interrupt will suspend the execution of the operation in progress and update the status register with the Execution-Aborted event. It then returns the 8044 program counter to a location before the aborted operation started. The Abort software procedure given in Appendix A gives the details of the execution of the ABORT command.

7.6 Disabling the Transmission or Reception

Transmission of a frame is aborted if the 80186 sends a TRANSMIT-DISABLE command to the 8044. The command causes the 8044 to clear the Transmit Buffer

Full (TBF) bit. During transmission, if the TBF bit is cleared, the SIU will discontinue the transmission and interrupt the 8044 CPU.

The RECEIVE-DISABLE command causes the 8044 to clear the Receive Buffer Empty (RBE) bit. The SIU aborts the reception, if the RBE bit is cleared by the CPU.

When transmission or reception of a frame is discontinued, the SIU interrupts the 8044 CPU. The CPU then updates the status and interrupts the 80186.

7.7 Handling Interrupts

When the 80186 sends a command, it sets the 8044 external interrupt flag. The 8044 services the interrupt at its own convenience. In the interrupt service routine the 8044 executes the appropriate instructions for a given command. During execution of a command the 8044 ignores any command, except ABORT, sent by the 80186 (see section 5.1.2). This is accomplished by clearing the interrupt flag before the 8044 returns from the interrupt service routine. During DMA operations the 8044 sets the external interrupt to high priority. An interrupt with high priority can suspend execution of an interrupt service routine with low priority. The ABORT command given by the 80186 will interrupt the execution of the DMA transfer in progress. Upon completion of ABORT, execution of the last operation will not be resumed (see Appendix A). Note that any other command given during the DMA operation will also abort the operation in progress and should be avoided.

8.0 8044 IN EXPANDED OPERATION

To increase the number of information bytes in a frame, the 8044 can be operated in Expanded mode. In Expanded operation the system memory can be used as the transmit and receive buffer instead of the 8044 internal RAM. AP-283, "Flexibility in Frame Size Operation with the 8044", describes Expanded operation in detail.

8.1 Transmitting a Message in Expanded Operation

In Expanded operation the 8044 transmits the frame while it is fetching the data from the system memory using DMA. An internal transmit buffer is not necessary. The system memory can be used as the transmit buffer by the 8044.

Upon receiving the Transmit command, the 8044 enables the SIU and fetches the first data byte from the Command/Data register. The SIU transmits the opening flag, station address, and the control byte if the frame format includes these fields. It then transmits the fetched data. The 8044 CPU fetches the next byte while the previously fetched byte is being transmitted by the SIU. The CPU fetches the remaining bytes using DMA, then the SIU transmits them simultaneously until the end of message is reached. The SIU then transmits the FCS bytes, the closing flag and interrupts the 8044 CPU. The 8044 updates the status with the Transmit-Done event and interrupts the 80186. If the DMA does not keep up with transmission, the transmission is an underrun.

8.2 Receiving a Message in Expanded Operation

In Expanded operation the DMA controller transfers data to the system memory while the 8044 SIU is receiving them.

To receive a message, the 80186 allocates a block of memory for storing the message. It sets the DMA channel and sends the Receive command to the 8044.

Upon reception of the command, the 8044 enables its serial channel and waits for a frame. The SIU performs flag detection, address filtering, zero bit deletion, NRZI decoding, and CRC checking as it does in Normal operation.

After the SIU receives the first byte of the frame, the 8044 CPU requests the transfer of the byte to memory using DMA. The 80186 DMA moves the information byte into the system memory while the SIU is receiving the next byte. The next byte is transferred to the memory after the SIU receives it. When the entire frame is received, the SIU checks the received Frame Check Sequence bytes. If there is no CRC error, the SIU updates the 8044 registers and interrupts the 8044 CPU. The CPU updates the status and interrupts the 80186.

9.0 CONCLUSION

This application note describes an efficient way to interface the 80186 and the 80188 microprocessors to the Intel 8-bit microcontrollers like the 80C51, 8052, and 8044. To illustrate this point the 80186 microprocessor interface to the 8044 microcontroller based serial communication chip was described. The hardware interface given here is very general and can interface the 8-bit microcontrollers to a variety of Intel microprocessors and DMA controllers. The microcontrollers with this interface hardware have the same benefits as both the Intel UPI-41/42 family and data communication peripheral chips such as the 82588 and the 82568 LAN controllers. Like the Intel UPI chips, they can be easily interfaced to microprocessors, and like the data communication peripherals, they execute high level commands. A similar approach can be used to interface Intel microprocessors to the 16-bit 8096 microcontroller.

APPENDIX A SOFTWARE

The software modules shown here implement the execution of commands and status explained in sections 5 and 7. The 80186 software provides procedures to send commands and read status. The 8044 software decodes and executes the commands, updates the status, and interrupts the 80186. The procedures given here are called by higher level software drivers. For example, an 80186 application program may use the Transmit command to send a block of data to an application program that resides in the 8044 ROM or in another remote node. The application programs and the drivers that perform the communication tasks run asynchronously since all communication tasks are interrupt-driven.

Figure A-1 shows how to assign the ports and control registers for an 80186-based system. The software is written for an Intel iSBC® 186/51 computer board. The 8044 hardware is connected to the computer board iSBX™ connector.

Figure A-2 shows the 80186 command procedures. These procedures are used by the data link driver.

Figure A-3 shows how the DMA controller is loaded and initialized for data and parameter transfer from the 80186 memory to the 8044. This procedure is used by the TRANSMIT and CONFIGURE commands.

Figure A-4 shows how the DMA controller is loaded and initialized for data and parameter transfer from the 8044 to the 80186 memory. This procedure is used by the RECEIVE and DUMP commands.

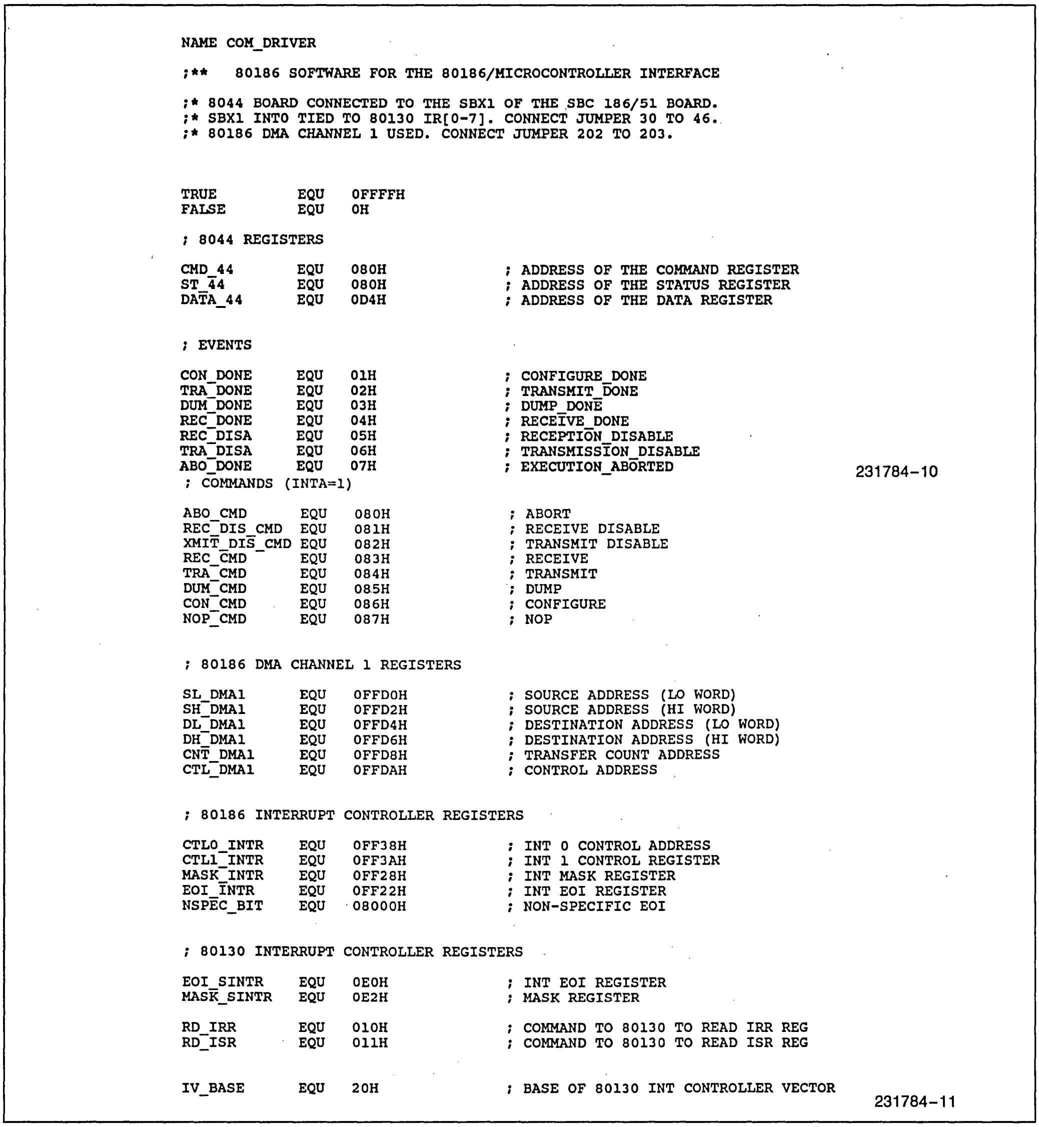
Figure A-5 shows an interrupt service routine which handles interrupts resulting from various events. Note that this routine is not complete. The user should write the software to respond to events.

Figure A-6 shows an example of the 80186 software. It shows how to start various operations. This is not a data link driver, but it gives the procedures needed to write a complete driver.

Figure A-7 shows how to initialize the 8044. The user application program should be inserted here.

Figures A-8 through A-13 show the 8044 external interrupt service routine. In this routine a command received from the 80186 is decoded, and one of the command procedures shown in Figures A-9 through A-13 is executed.

Figure A-14 shows the serial channel (SIU) interrupt service routine. Note that execution of TRANSMIT, RECEIVE, and TRANSMIT-DISABLE commands are completed in this routine.



```
NAME COM_DRIVER

;**    80186 SOFTWARE FOR THE 80186/MICROCONTROLLER INTERFACE

;* 8044 BOARD CONNECTED TO THE SBX1 OF THE SBC 186/51 BOARD.
;* SBX1 INT0 TIED TO 80130 IR[0-7]. CONNECT JUMPER 30 TO 46.
;* 80186 DMA CHANNEL 1 USED. CONNECT JUMPER 202 TO 203.


TRUE            EQU    0FFFFH
FALSE           EQU    0H

; 8044 REGISTERS

CMD_44          EQU    080H                ; ADDRESS OF THE COMMAND REGISTER
ST_44           EQU    080H                ; ADDRESS OF THE STATUS REGISTER
DATA_44         EQU    0D4H                ; ADDRESS OF THE DATA REGISTER


; EVENTS

CON_DONE        EQU    01H                 ; CONFIGURE_DONE
TRA_DONE        EQU    02H                 ; TRANSMIT_DONE
DUM_DONE        EQU    03H                 ; DUMP_DONE
REC_DONE        EQU    04H                 ; RECEIVE_DONE
REC_DISA        EQU    05H                 ; RECEPTION_DISABLE
TRA_DISA        EQU    06H                 ; TRANSMISSION_DISABLE
ABO_DONE        EQU    07H                 ; EXECUTION_ABORTED
; COMMANDS (INTA=1)

ABO_CMD         EQU    080H                ; ABORT
REC_DIS_CMD     EQU    081H                ; RECEIVE DISABLE
XMIT_DIS_CMD    EQU    082H                ; TRANSMIT DISABLE
REC_CMD         EQU    083H                ; RECEIVE
TRA_CMD         EQU    084H                ; TRANSMIT
DUM_CMD         EQU    085H                ; DUMP
CON_CMD         EQU    086H                ; CONFIGURE
NOP_CMD         EQU    087H                ; NOP


; 80186 DMA CHANNEL 1 REGISTERS

SL_DMA1         EQU    0FFD0H              ; SOURCE ADDRESS (LO WORD)
SH_DMA1         EQU    0FFD2H              ; SOURCE ADDRESS (HI WORD)
DL_DMA1         EQU    0FFD4H              ; DESTINATION ADDRESS (LO WORD)
DH_DMA1         EQU    0FFD6H              ; DESTINATION ADDRESS (HI WORD)
CNT_DMA1        EQU    0FFD8H              ; TRANSFER COUNT ADDRESS
CTL_DMA1        EQU    0FFDAH              ; CONTROL ADDRESS

; 80186 INTERRUPT CONTROLLER REGISTERS

CTL0_INTR       EQU    0FF38H              ; INT 0 CONTROL ADDRESS
CTL1_INTR       EQU    0FF3AH              ; INT 1 CONTROL REGISTER
MASK_INTR       EQU    0FF28H              ; INT MASK REGISTER
EOI_INTR        EQU    0FF22H              ; INT EOI REGISTER
NSPEC_BIT       EQU    08000H              ; NON-SPECIFIC EOI

; 80130 INTERRUPT CONTROLLER REGISTERS

EOI_SINTR       EQU    0E0H                ; INT EOI REGISTER
MASK_SINTR      EQU    0E2H                ; MASK REGISTER

RD_IRR          EQU    010H                ; COMMAND TO 80130 TO READ IRR REG
RD_ISR          EQU    011H                ; COMMAND TO 80130 TO READ ISR REG


IV_BASE         EQU    20H                 ; BASE OF 80130 INT CONTROLLER VECTOR
```

231784-10

231784-11

Figure A-1. Port and Register Definitions for 80186 System

```
;*********************************************************************
; INTERRUPT TABLE

INTERRUPTS      SEGMENT  AT 0

              ORG    (IV_BASE+1)*4H

IV_INTRO      LABEL   DWORD           ; IR1 VECTOR

INTERRUPTS     ENDS

;*******************************************************************

STACK          SEGMENT   STACK  'STACK'

THE_STACK      DW      200H    DUP(?)
TOS      LABEL         WORD

STACK          ENDS

;*******************************************************************

DATA           SEGMENT   PUBLIC  'DATA'

REC_BUFFER      DB      1024   DUP(?)

CON_BUFFER      DB      08H,01H,00H,0D0H,55H,20H,05H,30H,05H

DUM_BUFFER      DB      0FH     DUP(?)

TRA_BUFFER      DB      07H,55H,11H,01H,02H,03H,04H,05H

CMND_FLAG      DW      FALSE

DATA           ENDS
```

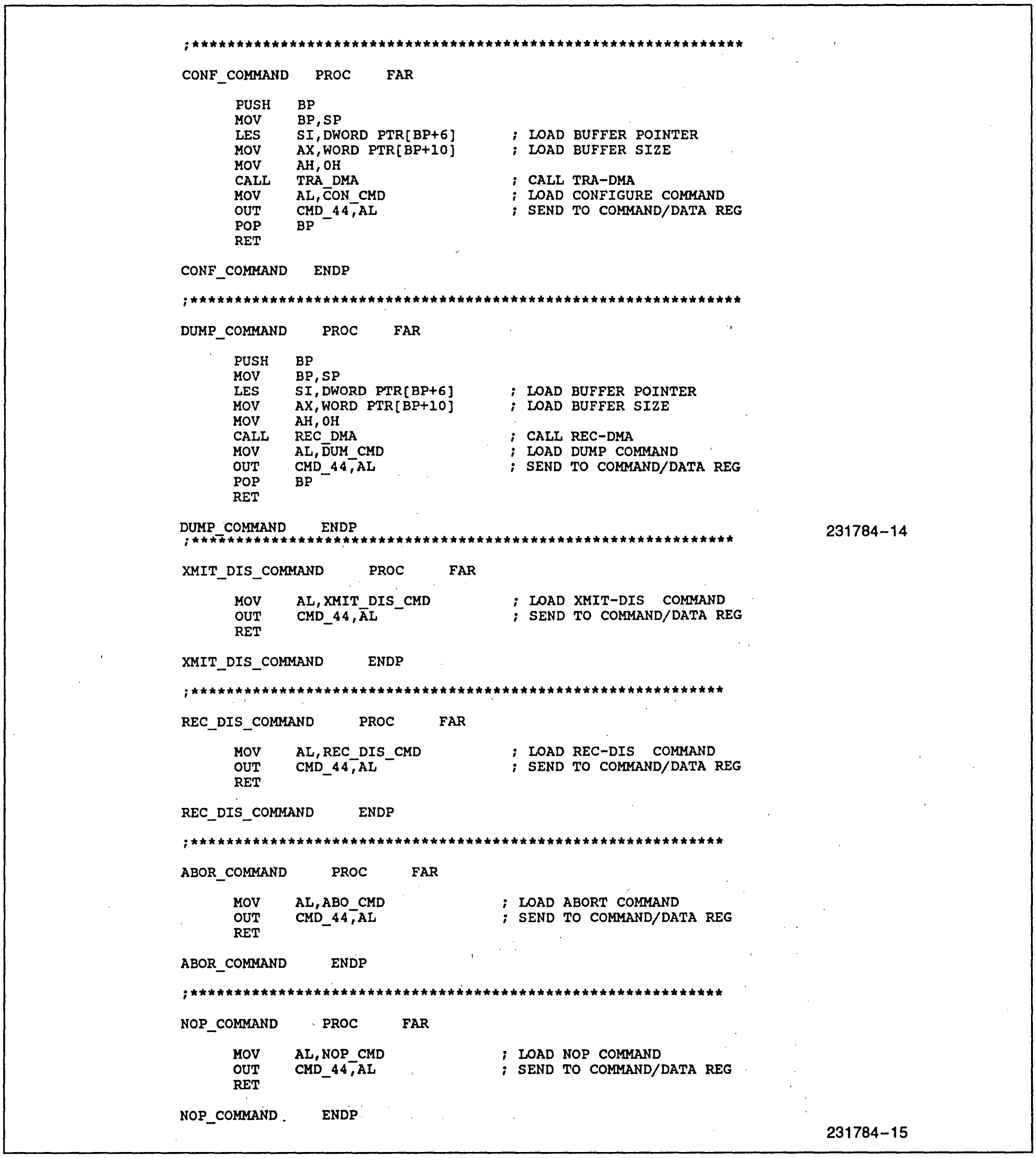
231784-12

Figure A-1. Port and Register Definitions for 80186 System (Continued)

```
;*************************************************************

CODE        SEGMENT  PUBLIC  'CODE'

ASSUME      CS:CODE,
&           DS:DATA,
&           ES:NOTHING,
&           SS:STACK

;*************************************************************

RECV_COMMAND     PROC    FAR

      PUSH   BP
      MOV    BP,SP
      LES    SI,DWORD PTR [BP+6]      ; LOAD BUFFER POINTER
      MOV    AX,WORD PTR[BP+10]       ; LOAD BUFFER SIZE
      MOV    AH,0H
      CALL   REC_DMA                  ; CALL REC-DMA
      MOV    AL,REC_CMD               ; LOAD RECEIVE COMMAND
      OUT    CMD_44,AL                ; SEND TO COMMAND/DATA REG
      POP    BP
      RET

RECV_COMMAND     ENDP

;***************************************************************

XMIT_COMMAND    PROC     FAR

      PUSH   BP
      MOV    BP,SP
      LES    SI,DWORD PTR [BP+6]      ; LOAD BUFFER POINTER
      MOV    AX,WORD PTR[BP+10]       ; LOAD BUFFER SIZE
      MOV    AH,0H
      CALL   TRA_DMA                  ; CALL TRA-DMA
      MOV    AL,TRA_CMD               ; LOAD TRANSMIT COMMAND
      OUT    CMD_44,AL                ; SEND TO COMMAND/DATA REG
      POP    BP
      RET

XMIT_COMMAND    ENDP
```

231784-13

Figure A-2. Setup and Execution of Commands



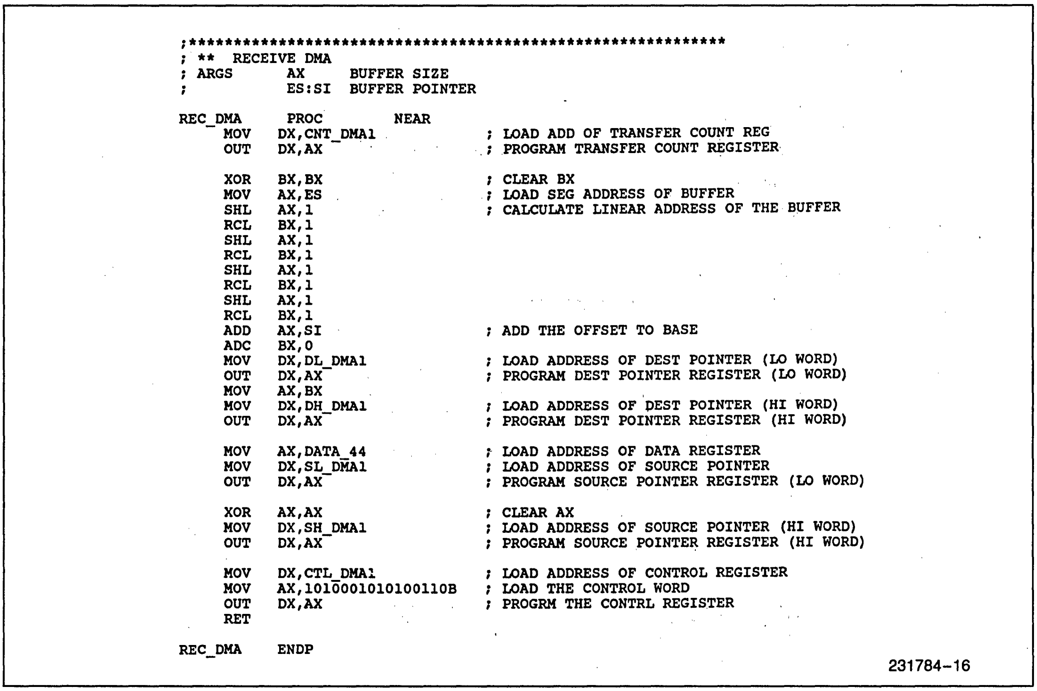
```
;*****************************************************************

CONF_COMMAND    PROC     FAR

      PUSH    BP
      MOV     BP,SP
      LES     SI,DWORD PTR[BP+6]       ; LOAD BUFFER POINTER
      MOV     AX,WORD PTR[BP+10]       ; LOAD BUFFER SIZE
      MOV     AH,0H
      CALL    TRA_DMA                  ; CALL TRA-DMA
      MOV     AL,CON_CMD               ; LOAD CONFIGURE COMMAND
      OUT     CMD_44,AL                ; SEND TO COMMAND/DATA REG
      POP     BP
      RET

CONF_COMMAND    ENDP

;*****************************************************************

DUMP_COMMAND     PROC     FAR

      PUSH    BP
      MOV     BP,SP
      LES     SI,DWORD PTR[BP+6]       ; LOAD BUFFER POINTER
      MOV     AX,WORD PTR[BP+10]       ; LOAD BUFFER SIZE
      MOV     AH,0H
      CALL    REC_DMA                  ; CALL REC-DMA
      MOV     AL,DUM_CMD               ; LOAD DUMP COMMAND
      OUT     CMD_44,AL                ; SEND TO COMMAND/DATA REG
      POP     BP
      RET

DUMP_COMMAND     ENDP
```

231784-14

```
;*****************************************************************

XMIT_DIS_COMMAND     PROC      FAR

      MOV     AL,XMIT_DIS_CMD          ; LOAD XMIT-DIS  COMMAND
      OUT     CMD_44,AL                ; SEND TO COMMAND/DATA REG
      RET

XMIT_DIS_COMMAND     ENDP

;***************************************************************

REC_DIS_COMMAND      PROC      FAR

      MOV     AL,REC_DIS_CMD           ; LOAD REC-DIS  COMMAND
      OUT     CMD_44,AL                ; SEND TO COMMAND/DATA REG
      RET

REC_DIS_COMMAND      ENDP

;***************************************************************

ABOR_COMMAND     PROC      FAR

      MOV     AL,ABO_CMD              ; LOAD ABORT COMMAND
      OUT     CMD_44,AL               ; SEND TO COMMAND/DATA REG
      RET

ABOR_COMMAND     ENDP

;***************************************************************

NOP_COMMAND     PROC      FAR

      MOV     AL,NOP_CMD              ; LOAD NOP COMMAND
      OUT     CMD_44,AL               ; SEND TO COMMAND/DATA REG
      RET

NOP_COMMAND     ENDP
```

231784-15

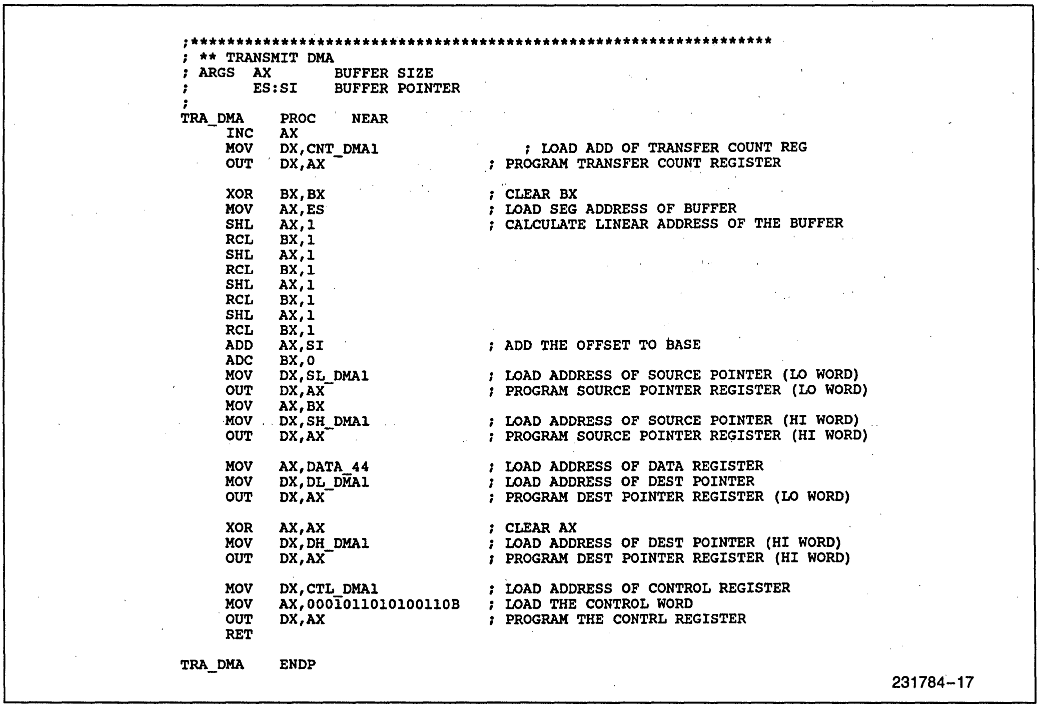
Figure A-2. Setup and Execution of Commands (Continued)



```
;****************************************************************
; **  RECEIVE DMA
; ARGS      AX      BUFFER SIZE
;           ES:SI   BUFFER POINTER

REC_DMA     PROC        NEAR
     MOV    DX,CNT_DMA1             ; LOAD ADD OF TRANSFER COUNT REG
     OUT    DX,AX                   ; PROGRAM TRANSFER COUNT REGISTER

     XOR    BX,BX                   ; CLEAR BX
     MOV    AX,ES                   ; LOAD SEG ADDRESS OF BUFFER
     SHL    AX,1                    ; CALCULATE LINEAR ADDRESS OF THE BUFFER
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     ADD    AX,SI                   ; ADD THE OFFSET TO BASE
     ADC    BX,0
     MOV    DX,DL_DMA1              ; LOAD ADDRESS OF DEST POINTER (LO WORD)
     OUT    DX,AX                   ; PROGRAM DEST POINTER REGISTER (LO WORD)
     MOV    AX,BX
     MOV    DX,DH_DMA1              ; LOAD ADDRESS OF DEST POINTER (HI WORD)
     OUT    DX,AX                   ; PROGRAM DEST POINTER REGISTER (HI WORD)

     MOV    AX,DATA_44              ; LOAD ADDRESS OF DATA REGISTER
     MOV    DX,SL_DMA1              ; LOAD ADDRESS OF SOURCE POINTER
     OUT    DX,AX                   ; PROGRAM SOURCE POINTER REGISTER (LO WORD)

     XOR    AX,AX                   ; CLEAR AX
     MOV    DX,SH_DMA1              ; LOAD ADDRESS OF SOURCE POINTER (HI WORD)
     OUT    DX,AX                   ; PROGRAM SOURCE POINTER REGISTER (HI WORD)

     MOV    DX,CTL_DMA1             ; LOAD ADDRESS OF CONTROL REGISTER
     MOV    AX,1010001010100110B    ; LOAD THE CONTROL WORD
     OUT    DX,AX                   ; PROGRM THE CONTRL REGISTER
     RET

REC_DMA     ENDP
```

231784-16

Figure A-3. Loading and Starting the 80186 DMA Controller



```
;*******************************************************************
; ** TRANSMIT DMA
; ARGS  AX        BUFFER SIZE
;       ES:SI     BUFFER POINTER
;
TRA_DMA     PROC   NEAR
     INC    AX
     MOV    DX,CNT_DMA1                 ; LOAD ADD OF TRANSFER COUNT REG
     OUT    DX,AX                   ; PROGRAM TRANSFER COUNT REGISTER

     XOR    BX,BX                   ; CLEAR BX
     MOV    AX,ES                   ; LOAD SEG ADDRESS OF BUFFER
     SHL    AX,1                    ; CALCULATE LINEAR ADDRESS OF THE BUFFER
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     ADD    AX,SI                   ; ADD THE OFFSET TO BASE
     ADC    BX,0
     MOV    DX,SL_DMA1              ; LOAD ADDRESS OF SOURCE POINTER (LO WORD)
     OUT    DX,AX                   ; PROGRAM SOURCE POINTER REGISTER (LO WORD)
     MOV    AX,BX
     MOV    DX,SH_DMA1              ; LOAD ADDRESS OF SOURCE POINTER (HI WORD)
     OUT    DX,AX                   ; PROGRAM SOURCE POINTER REGISTER (HI WORD)

     MOV    AX,DATA_44              ; LOAD ADDRESS OF DATA REGISTER
     MOV    DX,DL_DMA1              ; LOAD ADDRESS OF DEST POINTER
     OUT    DX,AX                   ; PROGRAM DEST POINTER REGISTER (LO WORD)

     XOR    AX,AX                   ; CLEAR AX
     MOV    DX,DH_DMA1              ; LOAD ADDRESS OF DEST POINTER (HI WORD)
     OUT    DX,AX                   ; PROGRAM DEST POINTER REGISTER (HI WORD)

     MOV    DX,CTL_DMA1             ; LOAD ADDRESS OF CONTROL REGISTER
     MOV    AX,0001011010100110B    ; LOAD THE CONTROL WORD
     OUT    DX,AX                   ; PROGRAM THE CONTRL REGISTER
     RET

TRA_DMA     ENDP
```

231784-17

Figure A-4. Loading and Starting the 80186 DMA Controller

```
;******************************************************************
;  80186 INTERRUPT ROUTINE

INT_186:

     PUSH  AX
     PUSH  DX
     MOV   AX,NSPEC_BIT                  ; SEND NSPEC END OF INT
     MOV   DX,EOI_INTR
     OUT   DX,AX

     MOV   AL,01100001B
     OUT   EOI_SINTR,AL

     IN    AL,ST_44                      ; READ THE STATUS
     AND   AX,0FFH

; DECODE STATUS AND TAKE APPROPRIATE ACTION

     MOV   DX,CTL_DMA1                   ; DISABLE DMA
     IN    AX,DX
     OR    AX,0100B
     AND   AX,NOT 010B
     OUT   DX,AX

     MOV   CMND_FLAG,TRUE

     POP   DX
     POP   AX
     IRET
```

231784-18

Figure A-5. Interrupt Service Routine

```
;*************************************************************

BEGIN:
          CLI
          CLD

; SET ALL REGISTERS SMALL MODEL

          MOV       SP,DATA
          MOV       DS,SP
          MOV       ES,SP
          MOV       SP,STACK
          MOV       SS,SP
          MOV       SP,OFFSET TOS


; SETUP INTERRUPT VECTORS

          PUSH      ES
          XOR       AX,AX
          MOV       ES,AX
          MOV       WORD PTR ES:IV_INTR0 +0, OFFSET INT_186
          MOV       WORD PTR ES:IV_INTR0 +2, CS
          POP       ES


SETUP 80130 INTERRUPT CONTROLLER

          MOV       AL,00010011B            ; ICW1
          OUT       EOI_SINTR,AL
          MUL       AL

          MOV       AL,IV_BASE              ; ICW2
          OUT       MASK_SINTR,AL
          MUL       AL

          MOV       AL,00000000B            ; ICW4
          OUT       MASK_SINTR,AL
          MUL       AL

          MOV       AL,0FCH                 ;MASK
          OUT       MASK_SINTR,AL
```

231784-19

Figure A-6. Example of Executing Commands

```
; SETUP 80186 INTERRUPT CONTROLLER

          MOV     AX,0000000000100000B
          MOV     DX,CTL0_INTR
          OUT     DX,AX

          MOV     DX,CTL1_INTR
          IN      AX,DX
          OR      AX,0000000000101000B
          OUT     DX,AX

          MOV     AX,000EDH                   ; MASK ALL BUT I0
          MOV     DX,MASK_INTR
          OUT     DX,AX
          STI                                 ;ENABLE INTERRUPTS

;***  SEND CONFIURE COMMAND

          PUSH    WORD PTR CON_BUFFER         ; PUSH BUFFER SIZE
          PUSH    DS                          ; PUSH BUFFER SEGMENT REGISTER
          PUSH    OFFSET CON_BUFFER           ; PUSH  OFFSET OF BUFFER
          CALL    CONF_COMMAND                ; CALL CONFIGURE
          ADD     SP,3*2

; WAIT FOR END OF COMMAND

WAIT1:
          CMP     CMND_FLAG,TRUE
          JNE     WAIT1
          MOV     CMND_FLAG,FALSE
```

231784-20

```
;***  SEND DUMP COMMAND

          PUSH    WORD PTR DUM_BUFFER         ; PUSH BUFFER SIZE
          PUSH    DS                          ; PUSH BUFFER SEGMENT REGISTER
          PUSH    OFFSET DUM_BUFFER           ; PUSH  OFFSET OF BUFFER
          CALL    DUMP_COMMAND                ; CALL CONFIGURE
          ADD     SP,3*2

WAIT2:
          CMP     CMND_FLAG,TRUE
          JNE     WAIT2
          MOV     CMND_FLAG,FALSE

;*** SEND TRANSMIT COMMAND

          PUSH    WORD PTR TRA_BUFFER         ; PUSH BUFFER SIZE
          PUSH    DS                          ; PUSH BUFFER SEGMENT REGISTER
          PUSH    OFFSET TRA_BUFFER           ; PUSH  OFFSET OF BUFFER
          CALL    XMIT_COMMAND                ; CALL COMMAND
          ADD     SP,3*2

WAIT3:
          CMP     CMND_FLAG,TRUE
          JNE     WAIT3
          MOV     CMND_FLAG,FALSE

;*** SEND RECEIVE COMMAND

          PUSH    WORD PTR REC_BUFFER         ; PUSH BUFFER SIZE
          PUSH    DS                          ; PUSH BUFFER SEGMENT REGISTER
          PUSH    OFFSET REC_BUFFER           ; PUSH  OFFSET OF BUFFER
          CALL    RECV_COMMAND                ; CALL COMMAND
          ADD     SP,3*2

WAIT4:
          CMP     CMND_FLAG,TRUE
          JNE     WAIT4
          MOV     CMND_FLAG,FALSE


CODE            ENDS
          END   BEGIN
```

231784-21

Figure A-6. Example of Executing Commands (Continued)

```
$DEBUG NOMOD51
$INCLUDE (REG44.PDF)

; THE 8044 SOFTWARE DRIVER FOR THE 80186/8044 INTERFACE.

         ORG    00H              ; LOCATIONS 00 THRU 26H ARE USED
         SJMP   INIT             ; BY INTERRUPT SERVICE ROUTINES.
         ORG    03H              ; VECTOR ADDRESS FOR EXT INT0.
         JMP    EINT0
         ORG    23H              ; VECTOR ADDRESS FOR SERIAL INT
         JMP    SIINT

;********************* INITIALIZATION ***************************

         ORG    26H
INIT:    MOV    TCON,#00000001B  ; EXT INT0: EDGE TRIGGER
         MOV    IE,#00010001B    ; SI=EX0=1
         CLR    P1.1             ; CLEAR DRQ STATUS BIT
         SETB   EA               ; ENABLE INTERRUPTS
DOT:     SJMP   DOT              ; WAIT FOR AN INTERRUPT
```

231784-22

Figure A-7. Initialization Routine

```
;********************EXTERNAL INTERRUPT 0 *********************
EINT0:   CLR    P1.5             ; CLEAR THE E BIT
         MOV    DPTR,#100H       ; LOAD DATA POINTER WITH A DUMY NUMBER
         MOVX   A,@DPTR          ; READ THE COMMAND BYTE.
         ANL    A,#00001111B     ; KEEP THE OPERATION FIELD
         MOV    R2,A             ; SAVE COMMAND

; DECODE COMMAND AND JUMP TO THE APPROPRIATE ROUTINE
;         COMMAND          OPERATION (BITS0-3)
;
;          ABORT              00H
;          REC-DISABLE        01H
;          TRA-DISABLE        02H
;          RECEIVE            03H
;          TRANSMIT           04H
;          DUMP               05H
;          CONFIGURE          06H
;          NOP                07H

         JNB     PX0,J1              ; IF INT0 IS SET TO PRIORITY 1,
         JMP     CABO                ;THEN DMA OPERATION WAS IN PROGRESS.
                                     ; EXECUTE ABORT REGARDLESS OF THE
                                     ;COMMAND ISSUED.
J1:      CJNE   A,#00H,J2            ; EXECUTE ABORT
         JMP    CABO                 ; THIS LINE WILL BE EXECUTED IF ABORT WAS
                                     ;ISSUED WHEN THE 8044 IS NOT EXECUTING
                                     ;ANY COMMANDS.
J2:      CJNE   A,#01H,J3
         JMP    CRDIS                ; EXECUTE RECEIVE-DISCONNECT
J3:      CJNE   A,#0B5H,J4
         JMP    CTDIS                ; EXECUTE TRANSMIT-DISCONNECT
J4:      CJNE   A,#03H,J5
         JMP    CREC                 ; EXECUTE RECEIVE
J5:      CJNE   A,#04H,J6
         JMP    CTRA                 ; EXECUTE TRANSMIT
J6:      CJNE   A,#05H,J7
         JMP    CDUMP                ; EXECUTE DUMP
J7:      CJNE   A,#06H,J8
         JMP    CCON                 ; EXECUTE CONFIGURE
J8:      CJNE   A,#07H,J9
         JMP    CNOP                 ; EXECUTE NOP
J9:      RETI                        ; RETURN. OPERATION NOT RECOGNIZED.
```

231784-23

Figure A-8. External Interrupt Service Routine

```
; **  NOP COMMAND

CNOP:     CLR     IE0                 ; IGNORE PENDING EXT INT0 (IF ANY).
                                      ; ANY INTERRUPT (COMMNAD) DURING
                                      ; EXECUTION OF AN OPERATION IS IGNORED
          RETI                        ; RETURN

; **  ABORT COMMAND

CABO:     JNB     PX0,CABOJ1          ; WAS DMA IN PROGRESS?
          CLR     PX0                 ; YES. EXT INT0: PRIORITY 0
          CLR     P1.1                ; CLEAR DMA REQUEST

          SETB    P1.2                ; UPDATE STATUS WITH
          SETB    P1.3                ;ABORT-DONE EVENT
          SETB    P1.4                ; (STATUS=DDH; E=0)

          CLR     IE0                 ; IGNORE PENDING EXT INT0 (IF ANY).
          CLR     P1.0
          SETB    P1.0                ; SET INT BIT AND INTERRUPT 80186
          JB      P3.2,$              ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
                                      ; EXECUTE THE NEXT "RETI" TWICE
          POP     ACC                 ; POP OUT THE OLD HI BYTE PC
          POP     ACC                 ; POP OUT THE OLD LOW BYTE PC
          MOV     B,#HIGH($+10)       ; HI BYTE ADDRESS OF CABOJ2
          MOV     ACC,#LOW($+7)       ; LOW BYTE ADDRESS OF CABOJ2
          PUSH    ACC                 ; PUSH THE ADDRESS OF THE NEXT
          PUSH    B                   ;"RETI" INSTRUCTION INTO STACK
CABOJ2:   RETI                        ; RETURN

CABOJ1:   NOP                         ; DMA WAS NOT IN PROGRESS
          SETB    P1.5                ; SET THE E BIT

          SETB    P1.2                ; UPDATE STATUS WITH
          SETB    P1.3                ;ABORT-DONE EVENT
          SETB    P1.4                ; (STATUS=FDH; E=1)

          CLR     IE0                 ; IGNORE PENDING EXT INT0 (IF ANY).
          CLR     P1.0
          SETB    P1.0                ; SET INT BIT AND INTERRUPT 80186
          JB      P3.2,$              ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
          RETI                        ; RETURN
```

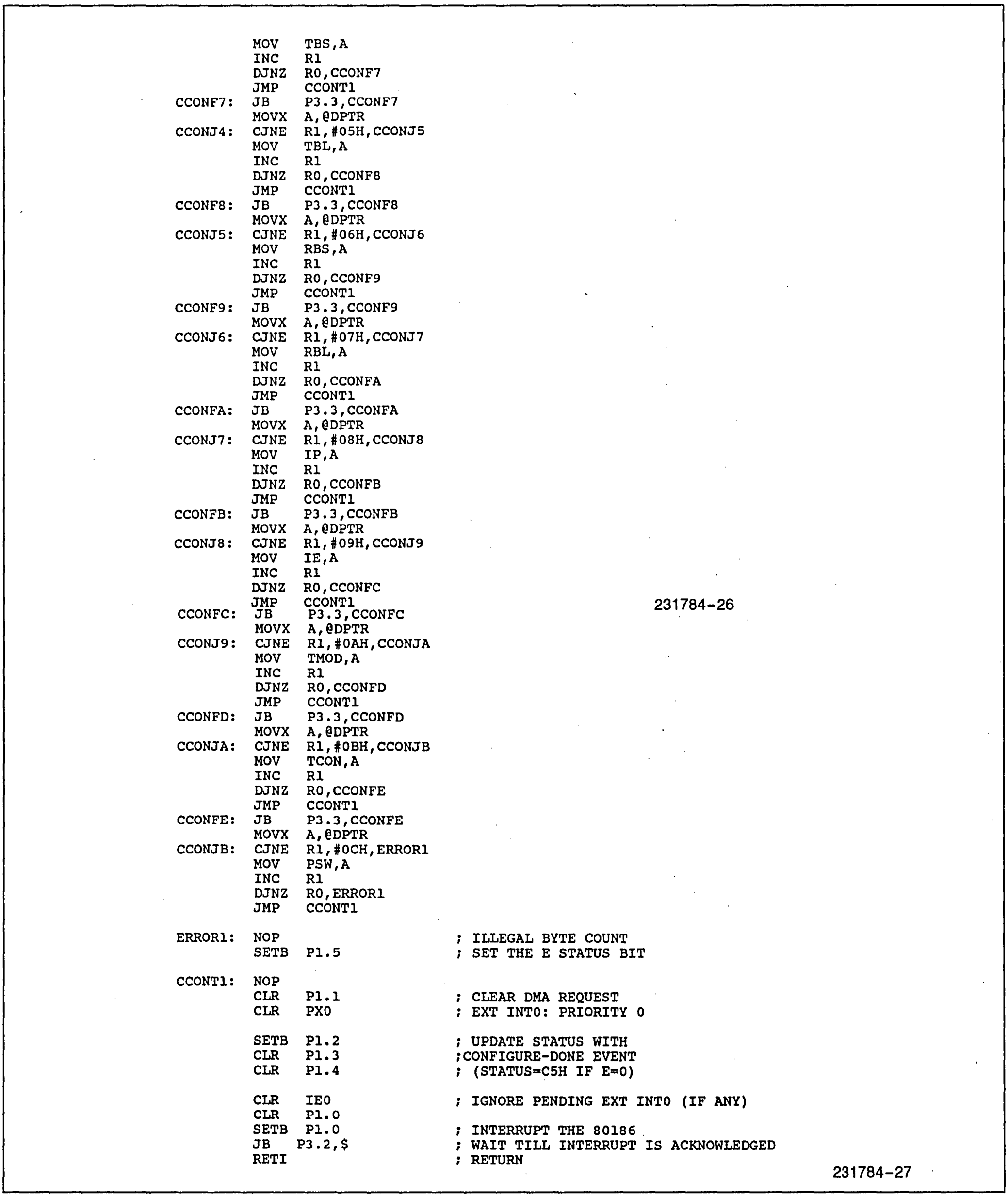
231784-24

Figure A-9. Execution of NOP and ABORT Commands

```
; ** CONFIGURE COMMNAD

CCON:     MOV     DPTR,#100H
          CLR     IE0                 ; IGNORE PENDING EXT INT0 (IF ANY)
          SETB    PX0                 ; EXT INT0: PRIORITY 1
                                      ; PX0 IS SET TO ACCEPT ABORT
                                      ;DURING DMA OPERATION.
          SETB    P1.1                ; ENABLE DMA REQUEST
          JB      P3.3,$              ; WAIT FOR DMA ACK.
          MOVX    A,@DPTR             ; READ FROM COMMAN/DATA REGISTER
          MOV     R0,A                ; LOAD BYTE COUNT
          DEC     R0                  ; DECREMENT BYTE COUNT
          JB      P3.3,$              ; WAIT FOR DMA ACK.
          MOVX    A,@DPTR             ; READ FROM COMMAND/DATA REGISTER
          MOV     R1,A                ; LOAD FIRST-BYTE
          JB      P3.3,$              ; WAIT FOR DMA ACK.
          MOVX    A,@DPTR             ; READ FROM COMMAND/DATA REGISTER
          CJNE    R1,#01H,CCONJ1      ; CHECK THE FIRST-BYTE
          MOV     STS,A               ; UPDATE THE STS REGISTER
          INC     R1                  ; INC. POINTER TO THE CONF. BLOCK
          DJNZ    R0,CCONF4           ; CHECK THE BYTE COUNT
          JMP     CCONT1
CCONF4:   JB      P3.3,CCONF4
          MOVX    A,@DPTR
CCONJ1:   CJNE    R1,#02H,CCONJ2
          MOV     SMD,A
          INC     R1
          DJNZ    R0,CCONF5
          JMP     CCONT1
CCONF5:   JB      P3.3,CCONF5
          MOVX    A,@DPTR
CCONJ2:   CJNE    R1,#03H,CCONJ3
          MOV     STAD,A
          INC     R1
          DJNZ    R0,CCONF6
          JMP     CCONT1
CCONF6:   JB      P3.3,CCONF6
          MOVX    A,@DPTR
CCONJ3:   CJNE    R1,#04H,CCONJ4
```

231784-25

Figure A-10. Execution of CONFIGURE Command



```
          MOV     TBS,A
          INC     R1
          DJNZ    R0,CCONF7
          JMP     CCONT1
CCONF7:   JB      P3.3,CCONF7
          MOVX    A,@DPTR
CCONJ4:   CJNE    R1,#05H,CCONJ5
          MOV     TBL,A
          INC     R1
          DJNZ    R0,CCONF8
          JMP     CCONT1
CCONF8:   JB      P3.3,CCONF8
          MOVX    A,@DPTR
CCONJ5:   CJNE    R1,#06H,CCONJ6
          MOV     RBS,A
          INC     R1
          DJNZ    R0,CCONF9
          JMP     CCONT1
CCONF9:   JB      P3.3,CCONF9
          MOVX    A,@DPTR
CCONJ6:   CJNE    R1,#07H,CCONJ7
          MOV     RBL,A
          INC     R1
          DJNZ    R0,CCONFA
          JMP     CCONT1
CCONFA:   JB      P3.3,CCONFA
          MOVX    A,@DPTR
CCONJ7:   CJNE    R1,#08H,CCONJ8
          MOV     IP,A
          INC     R1
          DJNZ    R0,CCONFB
          JMP     CCONT1
CCONFB:   JB      P3.3,CCONFB
          MOVX    A,@DPTR
CCONJ8:   CJNE    R1,#09H,CCONJ9
          MOV     IE,A
          INC     R1
          DJNZ    R0,CCONFC
          JMP     CCONT1
```

231784-26

```
CCONFC:   JB      P3.3,CCONFC
          MOVX    A,@DPTR
CCONJ9:   CJNE    R1,#0AH,CCONJA
          MOV     TMOD,A
          INC     R1
          DJNZ    R0,CCONFD
          JMP     CCONT1
CCONFD:   JB      P3.3,CCONFD
          MOVX    A,@DPTR
CCONJA:   CJNE    R1,#0BH,CCONJB
          MOV     TCON,A
          INC     R1
          DJNZ    R0,CCONFE
          JMP     CCONT1
CCONFE:   JB      P3.3,CCONFE
          MOVX    A,@DPTR
CCONJB:   CJNE    R1,#0CH,ERROR1
          MOV     PSW,A
          INC     R1
          DJNZ    R0,ERROR1
          JMP     CCONT1

ERROR1:   NOP                         ; ILLEGAL BYTE COUNT
          SETB    P1.5                ; SET THE E STATUS BIT

CCONT1:   NOP
          CLR     P1.1                ; CLEAR DMA REQUEST
          CLR     PX0                 ; EXT INTO: PRIORITY 0

          SETB    P1.2                ; UPDATE STATUS WITH
          CLR     P1.3                ;CONFIGURE-DONE EVENT
          CLR     P1.4                ; (STATUS=C5H IF E=0)

          CLR     IE0                 ; IGNORE PENDING EXT INTO (IF ANY)
          CLR     P1.0
          SETB    P1.0                ; INTERRUPT THE 80186
          JB      P3.2,$              ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
          RETI                        ; RETURN
```

231784-27

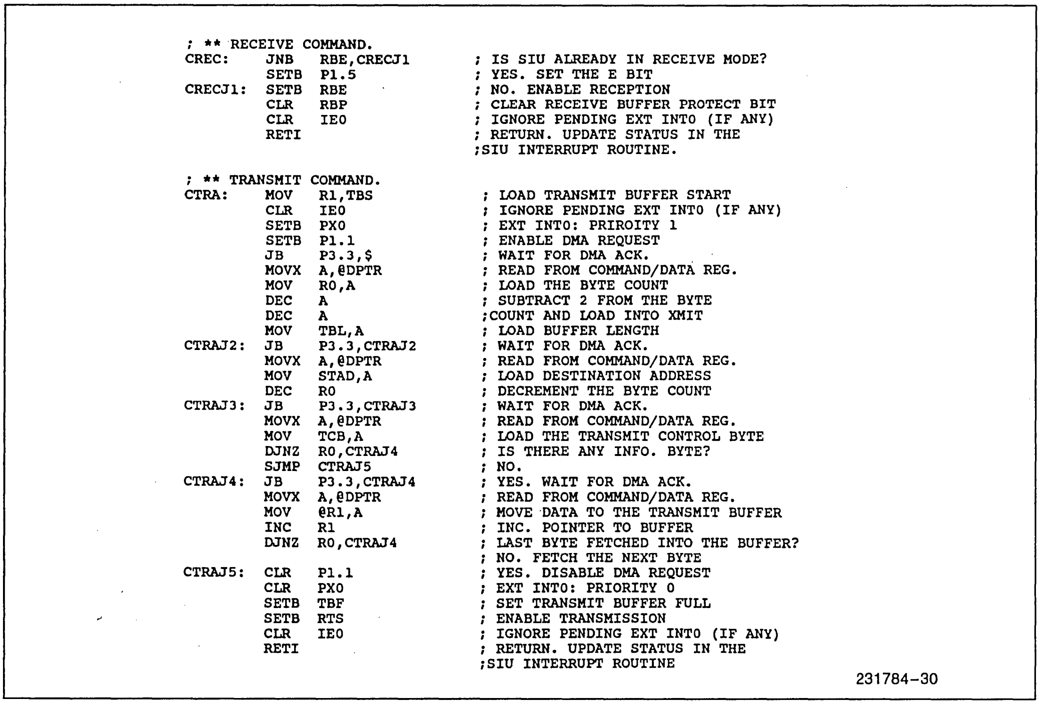
Figure A-10. Execution of CONFIGURE Command (Continued)

```
; ** DUMP COMMAND

CDUMP:  MOV     A,STS           ; LOAD THE FIRST DUMP REG INTO ACC
        MOVX    @DPTR,A         ; WRITE TO THE COMMAND/DATA REGISTER
        CLR     IE0             ; IGNORE PENDING EXT INT0 (IF ANY)
        SETB    PX0             ; INTRERRUPT 0: PRIORITY 1
        SETB    P1.1            ; ENABLE DMA REQUEST
        JB      P3.3,$          ; WAIT FOR DMA ACK
        MOV     A,SMD
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,STAD
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,TBS
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,TBL
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,TCB
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,RBS
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,RBL
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,RCB
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,RFL
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,PSW
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,IP
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,IE
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,TMOD
        MOVX    @DPTR,A
        JB      P3.3,$
        MOV     A,TCON
        MOVX    @DPTR,A
        JB      P3.3,$
        CLR     P1.1            ; DISABLE DRQ
        CLR     PX0             ; EXTERNAL INT0: PRIORITY 0

        SETB    P1.2            ; UPDATE STATUS WITH
        SETB    P1.3            ;DUMP-DONE EVENT
        CLR     P1.4            ; (STATUS=CDH)

        CLR     IE0             ; IGNORE PENDING EXT INT0
        CLR     P1.0
        SETB    P1.0            ; INTERRUPT THE 80186
        JB      P3.2,$          ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
        RETI                    ; RETURN
```

231784-28

231784-29

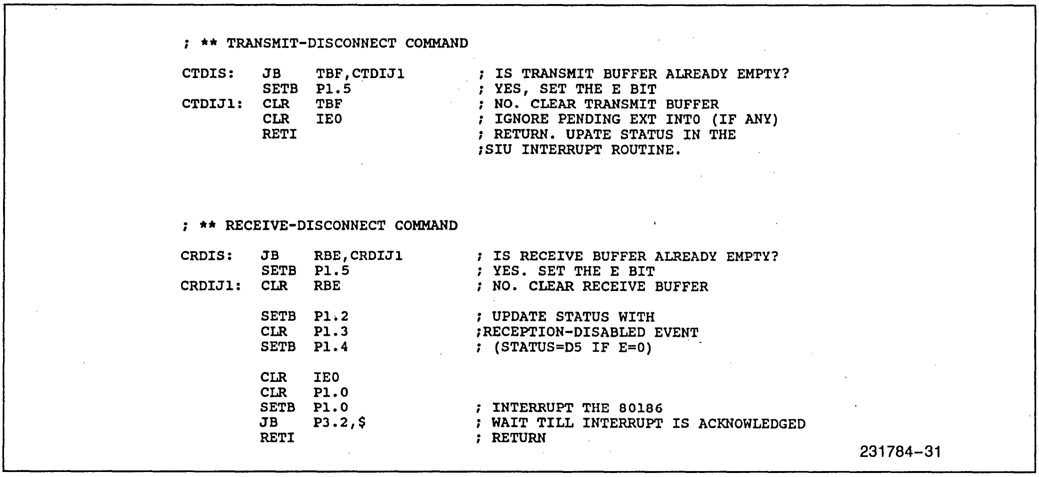
Figure A-11. Execution of DUMP Command



```
; ** RECEIVE COMMAND.
CREC:    JNB   RBE,CRECJ1       ; IS SIU ALREADY IN RECEIVE MODE?
         SETB  P1.5             ; YES. SET THE E BIT
CRECJ1:  SETB  RBE              ; NO. ENABLE RECEPTION
         CLR   RBP              ; CLEAR RECEIVE BUFFER PROTECT BIT
         CLR   IE0              ; IGNORE PENDING EXT INT0 (IF ANY)
         RETI                   ; RETURN. UPDATE STATUS IN THE
                                ;SIU INTERRUPT ROUTINE.

; ** TRANSMIT COMMAND.
CTRA:    MOV   R1,TBS           ; LOAD TRANSMIT BUFFER START
         CLR   IE0              ; IGNORE PENDING EXT INT0 (IF ANY)
         SETB  PX0              ; EXT INT0: PRIROITY 1
         SETB  P1.1             ; ENABLE DMA REQUEST
         JB    P3.3,$           ; WAIT FOR DMA ACK.
         MOVX  A,@DPTR          ; READ FROM COMMAND/DATA REG.
         MOV   R0,A             ; LOAD THE BYTE COUNT
         DEC   A                ; SUBTRACT 2 FROM THE BYTE
         DEC   A                ;COUNT AND LOAD INTO XMIT
         MOV   TBL,A            ; LOAD BUFFER LENGTH
CTRAJ2:  JB    P3.3,CTRAJ2      ; WAIT FOR DMA ACK.
         MOVX  A,@DPTR          ; READ FROM COMMAND/DATA REG.
         MOV   STAD,A           ; LOAD DESTINATION ADDRESS
         DEC   R0               ; DECREMENT THE BYTE COUNT
CTRAJ3:  JB    P3.3,CTRAJ3      ; WAIT FOR DMA ACK.
         MOVX  A,@DPTR          ; READ FROM COMMAND/DATA REG.
         MOV   TCB,A            ; LOAD THE TRANSMIT CONTROL BYTE
         DJNZ  R0,CTRAJ4        ; IS THERE ANY INFO. BYTE?
         SJMP  CTRAJ5           ; NO.
CTRAJ4:  JB    P3.3,CTRAJ4      ; YES. WAIT FOR DMA ACK.
         MOVX  A,@DPTR          ; READ FROM COMMAND/DATA REG.
         MOV   @R1,A            ; MOVE DATA TO THE TRANSMIT BUFFER
         INC   R1               ; INC. POINTER TO BUFFER
         DJNZ  R0,CTRAJ4        ; LAST BYTE FETCHED INTO THE BUFFER?
                                ; NO. FETCH THE NEXT BYTE
CTRAJ5:  CLR   P1.1             ; YES. DISABLE DMA REQUEST
         CLR   PX0              ; EXT INT0: PRIORITY 0
         SETB  TBF              ; SET TRANSMIT BUFFER FULL
         SETB  RTS              ; ENABLE TRANSMISSION
         CLR   IE0              ; IGNORE PENDING EXT INT0 (IF ANY)
         RETI                   ; RETURN. UPDATE STATUS IN THE
                                ;SIU INTERRUPT ROUTINE
```

231784-30

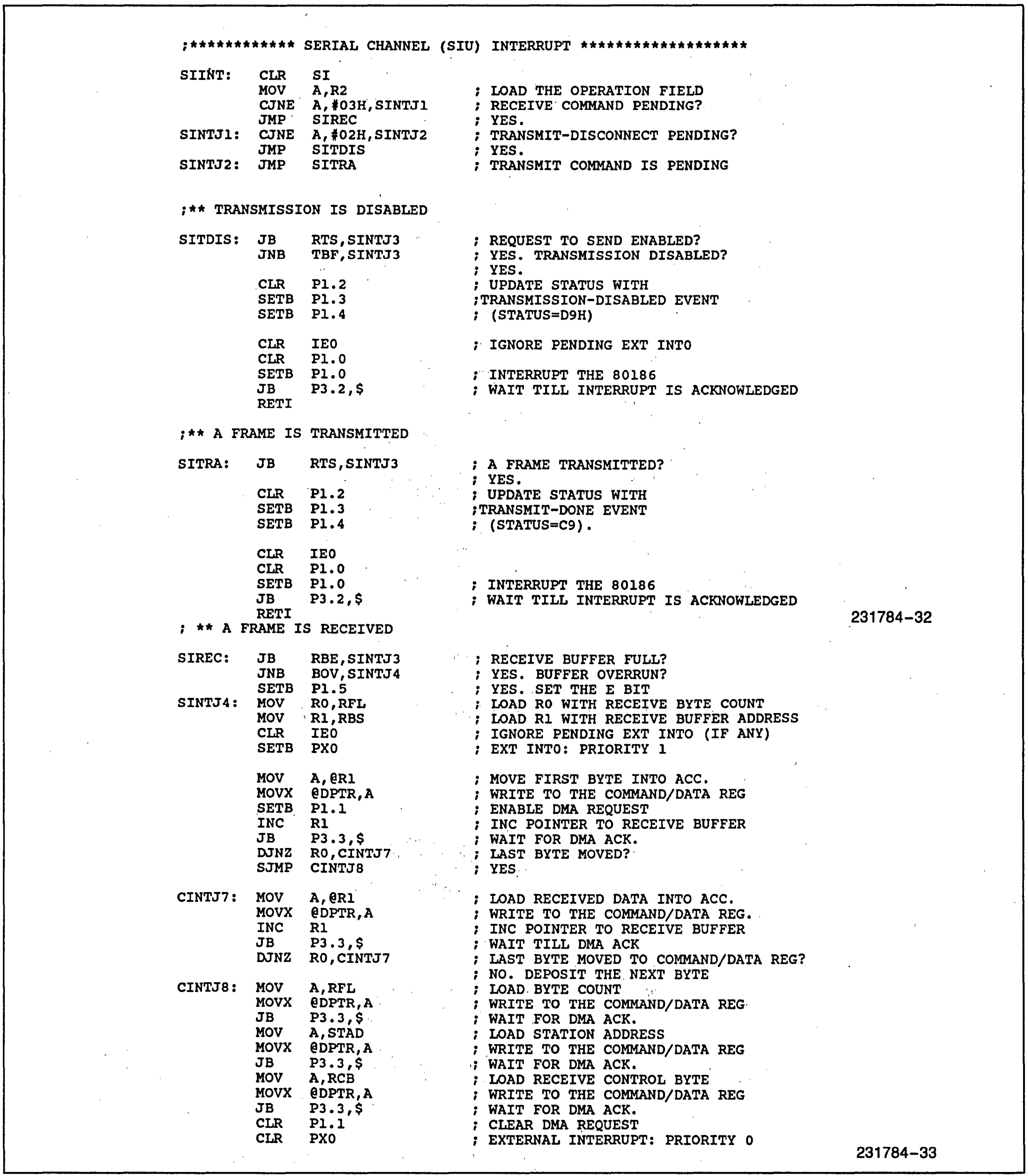
Figure A-12. Execution of RECEIVE and TRANSMIT Commands



```
; ** TRANSMIT-DISCONNECT COMMAND

CTDIS:   JB    TBF,CTDIJ1       ; IS TRANSMIT BUFFER ALREADY EMPTY?
         SETB  P1.5             ; YES, SET THE E BIT
CTDIJ1:  CLR   TBF              ; NO. CLEAR TRANSMIT BUFFER
         CLR   IE0              ; IGNORE PENDING EXT INT0 (IF ANY)
         RETI                   ; RETURN. UPATE STATUS IN THE
                                ;SIU INTERRUPT ROUTINE.


; ** RECEIVE-DISCONNECT COMMAND

CRDIS:   JB    RBE,CRDIJ1       ; IS RECEIVE BUFFER ALREADY EMPTY?
         SETB  P1.5             ; YES. SET THE E BIT
CRDIJ1:  CLR   RBE              ; NO. CLEAR RECEIVE BUFFER

         SETB  P1.2             ; UPDATE STATUS WITH
         CLR   P1.3             ;RECEPTION-DISABLED EVENT
         SETB  P1.4             ; (STATUS=D5 IF E=0)

         CLR   IE0
         CLR   P1.0
         SETB  P1.0             ; INTERRUPT THE 80186
         JB    P3.2,$           ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
         RETI                   ; RETURN
```

231784-31

Figure A-13. Execution of RECEIVE-DISCONNECT and TRANSMIT-DISCONNECT Commands



```
;************* SERIAL CHANNEL (SIU) INTERRUPT ********************

SIINT:   CLR   SI
         MOV   A,R2                ; LOAD THE OPERATION FIELD
         CJNE  A,#03H,SINTJ1       ; RECEIVE COMMAND PENDING?
         JMP   SIREC               ; YES.
SINTJ1:  CJNE  A,#02H,SINTJ2       ; TRANSMIT-DISCONNECT PENDING?
         JMP   SITDIS              ; YES.
SINTJ2:  JMP   SITRA               ; TRANSMIT COMMAND IS PENDING


;** TRANSMISSION IS DISABLED

SITDIS:  JB    RTS,SINTJ3          ; REQUEST TO SEND ENABLED?
         JNB   TBF,SINTJ3          ; YES. TRANSMISSION DISABLED?
                                   ; YES.
         CLR   P1.2                ; UPDATE STATUS WITH
         SETB  P1.3                ;TRANSMISSION-DISABLED EVENT
         SETB  P1.4                ; (STATUS=D9H)

         CLR   IE0                 ; IGNORE PENDING EXT INTO
         CLR   P1.0
         SETB  P1.0                ; INTERRUPT THE 80186
         JB    P3.2,$              ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
         RETI

;** A FRAME IS TRANSMITTED

SITRA:   JB    RTS,SINTJ3          ; A FRAME TRANSMITTED?
                                   ; YES.
         CLR   P1.2                ; UPDATE STATUS WITH
         SETB  P1.3                ;TRANSMIT-DONE EVENT
         SETB  P1.4                ; (STATUS=C9).

         CLR   IE0
         CLR   P1.0
         SETB  P1.0                ; INTERRUPT THE 80186
         JB    P3.2,$              ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
         RETI
```

231784-32

```
; ** A FRAME IS RECEIVED

SIREC:   JB    RBE,SINTJ3          ; RECEIVE BUFFER FULL?
         JNB   BOV,SINTJ4          ; YES. BUFFER OVERRUN?
         SETB  P1.5                ; YES. SET THE E BIT
SINTJ4:  MOV   R0,RFL              ; LOAD R0 WITH RECEIVE BYTE COUNT
         MOV   R1,RBS              ; LOAD R1 WITH RECEIVE BUFFER ADDRESS
         CLR   IE0                 ; IGNORE PENDING EXT INTO (IF ANY)
         SETB  PX0                 ; EXT INTO: PRIORITY 1

         MOV   A,@R1               ; MOVE FIRST BYTE INTO ACC.
         MOVX  @DPTR,A             ; WRITE TO THE COMMAND/DATA REG
         SETB  P1.1                ; ENABLE DMA REQUEST
         INC   R1                  ; INC POINTER TO RECEIVE BUFFER
         JB    P3.3,$              ; WAIT FOR DMA ACK.
         DJNZ  R0,CINTJ7           ; LAST BYTE MOVED?
         SJMP  CINTJ8              ; YES

CINTJ7:  MOV   A,@R1               ; LOAD RECEIVED DATA INTO ACC.
         MOVX  @DPTR,A             ; WRITE TO THE COMMAND/DATA REG.
         INC   R1                  ; INC POINTER TO RECEIVE BUFFER
         JB    P3.3,$              ; WAIT TILL DMA ACK
         DJNZ  R0,CINTJ7           ; LAST BYTE MOVED TO COMMAND/DATA REG?
                                   ; NO. DEPOSIT THE NEXT BYTE
CINTJ8:  MOV   A,RFL               ; LOAD BYTE COUNT
         MOVX  @DPTR,A             ; WRITE TO THE COMMAND/DATA REG
         JB    P3.3,$              ; WAIT FOR DMA ACK.
         MOV   A,STAD              ; LOAD STATION ADDRESS
         MOVX  @DPTR,A             ; WRITE TO THE COMMAND/DATA REG
         JB    P3.3,$              ; WAIT FOR DMA ACK.
         MOV   A,RCB               ; LOAD RECEIVE CONTROL BYTE
         MOVX  @DPTR,A             ; WRITE TO THE COMMAND/DATA REG
         JB    P3.3,$              ; WAIT FOR DMA ACK.
         CLR   P1.1                ; CLEAR DMA REQUEST
         CLR   PX0                 ; EXTERNAL INTERRUPT: PRIORITY 0
```

231784-33

Figure A-14. Serial Channel Interrupt Routine

```
         CLR    P1.2            ; UPDATE STATUS WITH
         CLR    P1.3            ;RECEIVE-DONE EVENT
         SETB   P1.4            ; (STATUS=D1H IF E=0)
         CLR    IE0             ; IGNORE PENDING EXT INT0
         CLR    P1.0
         SETB   P1.0            ; INTERRUPT THE 80186
         JB     P3.2,$          ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
         RETI

SINTJ3:  NOP
         RETI
END
```

231784-34

Figure A-14. Serial Channel Interrupt Routine (Continued)

8086/80186 SOFTWARE PACKAGES

8086/80186 Software Development Package

- Macro Assembler with Complete System Development Capability for 8086/80186 Designs
- Complete Set of Utilities for Object Module Management and Program Linkage

FORTRAN 8086/80186 Software Package

- Features High-Level Language Support for Floating-Point Calculation, Transcendentals, Interrupt Procedures, and Run-Time Exception Handling
- Meets ANSI FORTRAN 77 Subset Language Specifications
- Supports Complex Data Types

PASCAL 8086/80186 Software Package

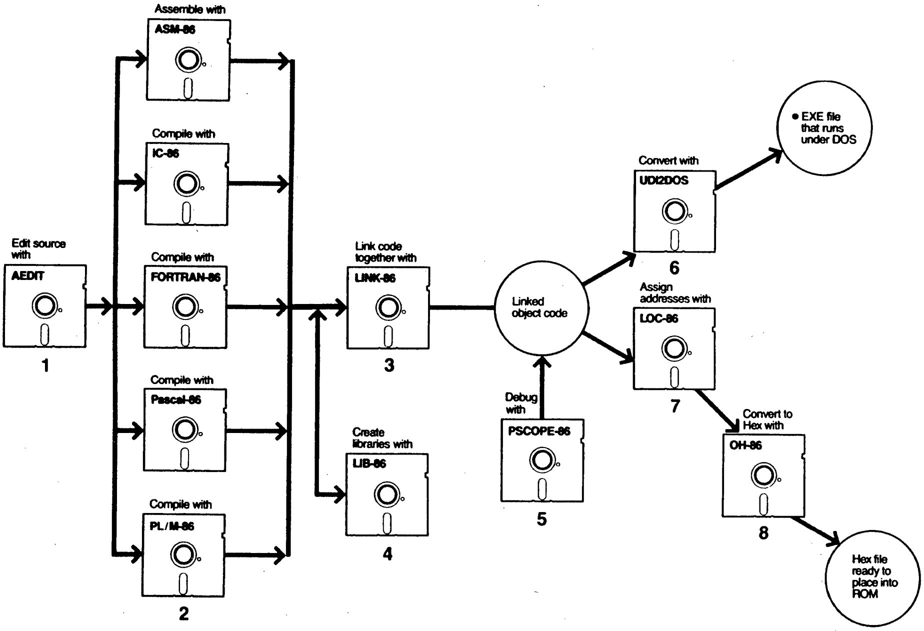
- Object Compatible and Linkable with PL/M 8086, ASM 8086 and FORTRAN 86
- Supports Large Array Operation

PL/M 8086/80186 Software Package

- Advanced Structured System Implementation Language for Algorithm Development
- Easy-to-Learn Block-Structured Language Encourages Program Modularity

iC-86 Compiler for the 8086/80186

- Implements Full ANSI Standard C Language
- Produces High Density Code Rivaling Assembler



210689-6

Figure 1. Program modules compiled with any of the 8086 languages may be linked together. Each language is compatible with Intel's debug tools. This is an example of development under DOS.

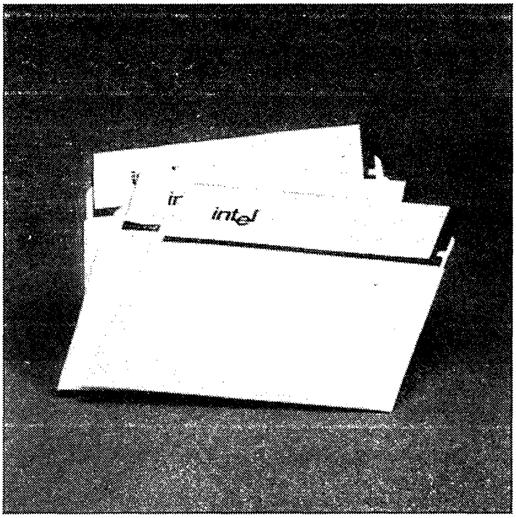
October 1987
Order Number: 210689-008

8086/80186 SOFTWARE DEVELOPMENT PACKAGE

- **Complete System Development Capability for High-Performance 8086 Applications**
- **Macro Assembler for Machine-Level Programming**
- **System Utilities for Program Linkage and Relocation**
- **Package Supports Program Development with PLM-86, Pascal-86, FORTAN 86, & iC 86**
- **Available on a Choice of Hosts**

The 8086 Software Development package contains a macro assembler, a program linker (for linking separately compiled modules together, a system locator, library manager, an object to hex code converter, and a conversion utility to create DOS executable files.

All the utilities in the Software Development Package run on the Intel Microcomputer Development Systems (Series III/Series IV) as weil as the IBM PC XT/AT DEC VAX† Minicomputer under the VMS† Operating System, and Intel systems 86/3XX under iRMX™86, and Intel System 286/3XX under iRMX™286.



210689-7

†VAX, VMS are trademarks of Digital Equipment Corporation.

8086/80186 MACRO ASSEMBLER

- **Produces Relocatable Object Code Which is Linkable to All Other Intel 86/186 Object Modules, Generated by Intel 8086 Compilers**
- **Powerful and Flexible Text Macro Facility with Three Macro Listings Options to Aid Debugging**
- **Highly Mnemonic and Compact Language, Most Mnemonics Represent Several Distinct Machine Instructions**
- **"Strongly Typed' Assembler Helps Detect Errors at Assembly Time**
- **High-Level Data Structuring Facilities Such as "STRUCTURES" and "RECORDS"**
- **Over 120 Detailed and Fully Documented Error Messages**

ASM-86 is the "high-level" macro assembler for the 86/186 assembly language. ASM-86 translates symbolic 86/186 assembly language mnemonics into 86/186 relocatable object code.

ASM-86 should be used where maximum code efficiency and hardware control is needed. The 86/186 assembly language includes approximately 100 instruction mnemonics. From these few mnemonics the assembler can generate over 3,800 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 100 mnemonics to generate all possible 86/186 machine instructions. ASM-86 will generate the shortest machine instruction possible given no forward referencing or given explicit information as to the characteristics of forward referenced symbols.

ASM-86 offers many features normally found only in high-level languages. The 86/186 assembly language is strongly typed. The assembler performs extensive checks on the usage of variables and labels. The assembler uses the attributes which are derived explicitly when a variable or label is first defined, then makes sure that each use of the symbol in later instructions conforms to the usage defined for that symbol. This means that many programming errors will be detected when the program is assembled, long before it is being debugged on hardware.

LINK-86

- **Automatic Combination of 8086 Programs Separately Translated Using Intel Compilers or Assemblers into Relocatable Object Module**
- **Automatic Selection of Required Modules from Specified Libraries to Satisfy Symbolic References**
- **Extensive Debug Symbol Manipulation, allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**
- **Automatic Generation of a Summary Map Giving Results of the LINK-86 Process**
- **Abbreviated Control Syntax**
- **Relocatable Modules May Be Merged into a Single Module Suitable for Inclusion in a Library**
- **Supports "Incremental" Linking**
- **Supports Type Checking of Public and External Symbols**

LINK-86 combines object modules specified in the LINK-86 input list into a single output module. LINK-86 combines segments from the input modules according to the order in which the modules are listed.

LINK-86 will accept libraries and object modules built from any Intel translator generating 8086 Relocatable Object Modules.

Support for incremental linking is provided since an output module produced by LINK-86 can be an input to another link. At each stage in the incremental linking process, unneeded public symbols may be purged.

LINK-86 supports type checking of PUBLIC and EXTERNAL symbols reporting a warning if their types are not consistant.

LINK-86 will link any valid set of input modules without any controls. However, controls are available to control the output of diagnostic information in the LINK-86 process and to control the content of the output module.

LINK-86 allows the user to create a large program as the combination of several smaller, separately compiled modules. After development and debugging of these component modules the user can link them together, locate them using LOC-86 and enter final testing with much of the work accomplished.

LOC-86

- **Automatic Generation of a Summary Map Giving Starting Address, Segment Addresses and Length, and Debug Symbols and Their Addresses**
- **Abbreviated Control Syntax**
- **Segments May be Relocated to Best Match Users Memory Configuration**
- **Extensive Debug Symbol Manipulation Allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**

Relocatability allows the programmer to code programs or sections of programs without having to know the final arrangement of the object code in memory.

LOC-86 converts relative addresses in an input module in 86/186 object module format to absolute addresses. LOC-86 orders the segments in the input module and assigns absolute addresses to the segments. The sequence in which the segments in the input module are assigned absolute addresses is determined by their order in the input module and the controls supplied with the command.

LOC-86 will relocate any valid input module without any controls. However, controls are available to control the output of diagnostic information in the LOC-86 process, to control the content of the output module, or both.

The program you are developing will almost certainly use some mix of random access memory (RAM), read-only memory (ROM), and/or programmable read-only memory (PROM). Therefore, the location of your program affects both cost and performance in your application. The relocation feature allows you to develop your program and then simply relocate the object code to suit your application.

LIB-86

- **LIB-86 is a Library Manager Program which Allows You to:**
  - **—Create Specifically Formatted Files to Contain Libraries of Object Modules**
  - **—Maintain These Libraries by Adding or Deleting Modules**
  - **—Print a Listing of the Modules and Public Symbols in a Library File**
- **Libraries Can be Used as Input to LINK-86 which Will Automatically Link Modules from the Library that Satisfy External References in the Modules Being Linked**
- **Abbreviated Control Syntax**

Libraries aid in the job of building programs. The library manager program LIB-86 creates and maintains files containing object modules. The operation of LIB-86 is controlled by commands to indicate which operation LIB-86 is to perform. The commands are:

CREATE:	creates an empty library file
ADD:	adds object modules to a library file
DELETE:	deletes modules from a library file
LIST:	lists the module directory of library files
EXIT:	terminates the LIB-86 program and returns control to VMS

When using object libraries, the linker will call only those object modules that are required to satisfy external references, thus saving memory space.

OH-86

- **Converts an 86/186 Absolute Object Module to Symbolic Hexadecimal Format**
- **Facilitates Preparing a File for Loading by Symbolic Hexadecimal Loader (e.g. iSBC™ Monitor SDK-86 Loader), or Universal PROM Mapper**
- **Converts an Absolute Module to a More Readable Format that can be Displayed on a CRT or Printed for Debugging**

The OH-86 utility converts an 86/186 absolute object module to the hexadecimal format. This conversion may be necessary for later loading by a hexadecimal loader such as the iSBC 86/12 monitor or the Universal PROM Mapper. The conversion may also be made to put the module in a more readable format that can be displayed or printed.

The module to be converted must be in absolute form; the output from LOC-86 is in absolute format.

SPECIFICATIONS

Documentation Package

ASM-86 Assembly Language Reference Manual

8086/87/88 Macro Assembler Operating Instructions

iAPX 86 Family Utilities User's Guide

Support Available

Software Updates, Subscription Service, Hotline Support

ORDERING INFORMATION

Order Code	Operating Environment
D86ASM86	IBM PC XT/AT running PC DOS Version 3.0 or later
VVSASM86	VAX†/VMS†
MVVSASM86	MICROVAX†/VMS†
R86ASM86	Intel 86/3XX Systems running: iRMX™ 86
R286ASM286	Intel 286/3XX Systems running: iRMX™ 286

†MICROVAX, VAX, VMS are trademarks of Digital Equipment Corporation.

*IBM, AT are registered trademarks of International Business Machines Corporation.

FORTRAN 8086/80186 SOFTWARE PACKAGE

- **Features High-Level Language Support for Floating-Point Calculations, Transcendentals, Interrupt Procedures, and Run-Time Exception Handling**
- **Meets ANSI FORTRAN 77 Subset Language Specifications**
- **Supports 8086/20, 8088/20 Numeric Data Processor for Fast and Efficient Execution of Numeric Instructions**
- **Uses REALMATH Floating-Point Standard for Consistent and Reliable Results**
- **Supports Arrays Larger Than 64K**
- **Unlimited User Program Symbols**
- **Offers Upward Compatibility with FORTRAN 80**
- **Provides FORTRAN Run-Time Support for 86/186 Based Design**
- **Provides Users Ability to do Formatted and Unformatted I/O with Sequential or Direct Access Methods**
- **I^{2}ICE™ Symbolic Debugging Fully Supported**
- **PSCOPE Source Level Debugging Fully Supported**
- **Supports Complex Data Types**
- **Choice of Industry Standard Hosts**

FORTRAN 86/186 meets the ANSI FORTRAN 77 Language Subset Specification and includes many features of the full standard. Therefore, the user is assured of portability of most existing ANS FORTRAN programs and of full portability from other computer systems with an ANS FORTRAN 77 Compiler.

FORTRAN 86/186 is available to run on the Intel Microcomputer Development Systems (Series III/Series IV) as well as the IBM PC XT/AT running PC DOS Version 3.0 or later, Digital Equipment VAX†/VMS† and Intel System 86/3XX running iRMX™ 86 operating system.

FORTRAN 86/186 is one of a complete family of compatible programming languages for 8086, 8088, 80186, 80188 development: PL/M, Pascal, FORTRAN, C, and Assembler. Therefore, users may choose the language best suited for a specific problem solution.

†VAX, VMS are trademarks of Digital Equipment Corporation.

*IBM, AT are registered trademarks of International Business Machines Corporation.

FEATURES

Extensive High-Level Language Numeric Processing Support

Single (32-bit), double (64-bit), and double extended precision (80-bit) complex (two 32-bit), and double complex (two 64-bit) floating-point data types

REALMATH Proposed IEEE Floating-Point Standard) for consistent and reliable results

Full support for all other data types: integer, logical, character

Ability to use hardware (8086/20, 8088/20 Numeric Data Processor) or software (simulator) floating-point support chosen at link time

ANSI FORTRAN 77 Standard

Intel® Microprocessor Support

FORTRAN 86/186 language features support of 8086/20, 8088/20 Numeric Data Processor

Compiler generates in-line iAPX 8086/20, 8088/20 Numeric Data Processor object code for floating-point arithmetic (See Figure 2)

Intrinsics allow user to control iAPX 8086/20, 8088/20 Numeric Data processor

8086, 8088, 80186, 80188 architectural advantages used for indexing and character-string handling

Symbolic debugging of application using ICE emulators

Source level debugging using PSCOPE

FLOATING-POINT-STATEMENT

```
      TEMPER = (PRESS - VOLUM / QUEK) - 3.45 / (PRESS - VOLUM / QUEK
     & - (PRESS - VOLUM / QUEK) * (PRESS - VOLUM / QUEK)
```

OBJECT CODE GENERATED

```
Intel FORTRAN 8086 Compiler
      8086/20, 8088/20
      MACHINE-CODE           ASSEMBLER MNEMONICS
                                                        ; STATEMENT # 2
0013  9BD9060C00             FLD         VOLUM
0018  9BD8360000             FDIV        QUEK
001D  9BD82E0800             FSUBR       PRESS
0022  9BDDD1                 FST         TOS+1H
0025  9B2ED83E0000           FDIVR       CS:@CONST
002B  9BD9C9                 FXCHG       TOS+1H
002E  9BDDD2                 FST         TOS+2H
0031  9BDEE9                 FSUBRP
0034  9BD9C1                 FLD         TOS+1H
0037  9BD8C8                 FMUL        TOS
003A  9BDDC2                 FFREE       TOS+2H
003D  9BDEE1                 FSUBP
0040  9BD91E0400             FSTP        TEMPER
0045  9B                     WAIT
```

Figure 2. Object code generated by FORTRAN 86/186 for a floating-point calculation using 8086/20, 8088/20 Numeric Processor.

Microprocessor Application Support

— Direct byte- or word-oriented port I/O

— Reentrant procedures

— Interrupt procedures

BENEFITS

FORTRAN 86/186 provides a means of developing application software for the Intel 86/186 products lines in a familiar, widely accepted, and industry-standard programming language. FORTRAN 86/186 will greatly enhance the user's ability to provide cost-effective software development for Intel microprocessors as illustrated by the following:

Early Project Completion

FORTRAN is an industry-standard, high-level numerics processing language. FORTRAN programmers can use FORTRAN 86/186 on microprocessor projects with little retraining. Existing FORTRAN software can be compiled with FORTRAN 86/186 and programs developed in FORTRAN 86/186 can run on other computers with ANSI FORTRAN 77 with little or no change. Libraries of mathematical programs using ANSI 77 standards may be compiled with FORTRAN 86/186.

Application Object Code Portability for a Processor Family

FORTRAN 86/186 modules "talk" to the resident Intellec development operating system using Intel's standard interface for all development-system software. This allows an application developed under the ISIS-II operating system to execute on iRMX/86, or a user-supplied operating system by linking in the iRMX/86 or other appropriate interface library. A standard logical-record interface enables communication with non-standard I/O devices.

Comprehensive, Reliable and Efficient Numeric Processing

The unique combination of FORTRAN 8086/8088, 8086/20, 8088/20 Numeric Data processor, and REALMATH (Proposed IEEE Floating-Point Standard) provide universal consistency in results of numeric computations and efficient object code generation.

SPECIFICATIONS

Documentation Package

FORTRAN 86/88/186/188 User's Guide

ORDERING INFORMATION

Order Code	Operating Environment
D86FOR86	IBM PC XT/AT running PC DOS Version 3.0 or later
R86FOR86	Intel System 86/3XX running iRMX 86
VVSFOR86	For 86 VAX/VMS 4.3 and later

SUPPORT AVAILABLE

Software updates, Subscription Service, Hotline Support.

intel®

PASCAL 86/186 SOFTWARE PACKAGE

- **Choice of Industry Standard Hosts**
- **Object Compatible and Linkable with PL/M 86/186, ASM 86/186, iC86/186 and FORTRAN 86/186**
- **I²ICE™ Symbolic Debugging Fully Supported**
- **PSCOPE Source Level Dubugging Fully Supported**
- **Implements REALMATH for Consistent and Reliable Results**
- **Supports Large Array Operation**
- **Unlimited User Program Symbols**
- **Supports 8086/20, 8088/20 Numeric Data Processors**
- **Strict Implementation of ISO Standard Pascal**
- **Useful Extensions Essential for Microcomputer Applications**
- **Separate Compilation with Type-Checking Enforced Between Pascal Modules**
- **Compiler Option to Support Full Run-Time Range-Checking**

PASCAL 86/186 conforms to and implements the ISO PASCAL standard. The language is enhanced to support microcomputer applications with special features, such as separate compilation, interrupt handling and direct port I/O. To assist the development of portable software, the compiler can be directed to flag all non-standard features.

The PASCAL 86/186 compiler runs on Series III and Series IV Microcomputer Development Systems, as well as the IBM* XT/AT* running PC DOS Version 3.0 or later, Digital Equipment VAX/VMS†, and Intel System 8086/3XX running iRMX™ 86.

A well-defined I/O interface is provided for run-time support. This allows a user-written operating system to support application programs as an alternate to the development system environment. Program modules compiled under PASCAL 86/186 are compatible and linkable with modules written in PL/M 86/186, ASM 86/186, C86/186 or FORTRAN 86/186. With a complete family of compatible programming languages for the 86/186 one can implement each module in the language most appropriate to the task at hand.

PASCAL 86/186 object modules contain symbol and type information for program debugging using ICE emulators and PSCOPE source language debugger. For final production version, the compiler can remove this extra information and code.

†VAX, VMS are trademarks of Digital Equipment Corporation.

FEATURES

Includes all the language features of Jensen & Wirth Pascal as defined in the ISO Pascal Standard.

Supports required extensions for microcomputer applications.

- Interrupt handling
- Direct port I/O

Separate compilation extensions allow:

- Modular decomposition of large programs
- Linkage with other Pascal modules as well as PL/M 86/186, ASM 86/186, C86/186 and FORTRAN 86/186
- Enforcement of type-checking at LINK-time

Supports numerous compiler options to control the compilation process, to INCLUDE files, flag non-standard Pascal statements and others to control program listing and object modules.

Utilizes the IEEE standard for Floating-Point Arithmetic (the Intel REALMATH standard) for arithmetic operations.

Well-defined and documented run-time operating system interfaces allow the user to execute the applications under user-designed operations systems.

Predefined type extensions allow:

- Create precision in read, integer, and unsigned calculations.
- Means to check 8087 errors
- Circumvention of rigid type checking on calls to non-Pascal routines

BENEFITS

Provides a standard Pascal for 86/186 based applications.

- Pascal has gained wide acceptance as a portable application language for microcomputer applications
- It is being taught in many colleges and universities around the world
- It is easy to learn, originally intended as a vehicle for teaching computer programming
- Improves maintainability: Type mechanism is both strictly enforced and user extendable
- Few machine specific language constructs

Strict implementation of the proposed ISO standard for Pascal aids portability of application programs. A compile time option checks conformance to the standard making it easy to write conforming programs.

PASCAL 86/186 extensions via predefined procedures for interrupt handling and direct port I/O make it possible to code an entire application in Pascal without compromising portability.

Standard Intel REALMATH is easy to use and provides reliable results, consistent with other Intel languages and other implementations of the IEEE proposed Floating-Point standard.

Provides run-time support for co-processors. All real-type arithmetic is performed on the 86/20 numeric data processor unit or software emulator. Runtime library routines, common between Pascal and other Intel languages (such as FORTRAN), permit efficient and consistently accurate results.

Extended relocation and linkage support allows the user to link Pascal program modules with routines written in other languages for certain parts of the program. For example, real-time or hardware dependent routines written in ASM 86/186 or PL/M 86/186 can be linked to Pascal routines, further extending the user's ability to write structured and modular programs.

PASCAL 86/186 programs "talk" to the resident operating system using Intel's standard interface for translated programs. This allows users to replace the development operating system by their own operating systems in the final application.

PASCAL 8086/8088 takes full advantage of 86/186 high level language architecture to generate efficient machine code.

Compiler options can be used to control the program listings and object modules. While debugging, the user may generate additional information such as the symbol record information required and useful for debugging using PSCOPE or ICE emulation. After debugging, the production version may be streamlined by removing this additional information.

SPECIFICATIONS

ORDERING INFORMATION

Ordering Code	Operating Environment
D86PAS86	IBM PC XT/AT running PC DOS Version 3.0 or later
R86PAS86	Intel System 86/3XX running iRMX™ 86
VVSPAS86	VAX/VMS
MVVPAS86	MICROVAX/VMS

Documentation Package

PASCAL 86 User's Guide

SUPPORT

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

PL/M 86/186 Software Package

- **Systems Programming Language for the 86/186 Processors**
- **Language is Upward Compatible from PL/M 80, Assuring MCS®-80/85 Design Portability**
- **Advanced Structured System Implementation Language for Algorithm Development**
- **Supports 16-Bit Signed Integer and 32-Bit Floating Point Arithmetic in Accordance with IEEE Proposed Standard**
- **Easy-to-Learn Block-Structured Language Encourages Program Modularity**
- **Improved Compiler Performance Now Supports More User Symbols and Faster Compilation Speeds**
- **Produces Relocatable Object Code Which Is Linkable to All Other 8086 Object Modules**
- **Code Optimization Assures Efficient Code Generation and Minimum Application Memory Utilization**
- **Built-In Syntax Checker Doubles Performance for Compiling Programs Containing Errors**
- **Resident on Choice of Hosts**
- **I^2ICE Symbolic Debugging Fully Supported**
- **PSCOPE Source Level Debugging Fully Supported**

PL/M 86/186 is an advanced, structured, high-level systems programming language. The PL/M 86/186 compiler was created specifically for performing software development for the Intel 86/186 Microprocessors. PL/M was designed so that program statements naturally express the program algorithm. This frees the programmer to concentrate on the logic of the program without concern for burdensome details of machine or assembly language programming (such as register allocation, meanings of assembler mnemonics, etc.).

The PL/M 86/186 compiler efficiently converts free-form PL/M language statements into machine instructions. Substantially fewer PL/M statements are necessary for a given application than if it were programmed at the assembly language or machine code level.

The use of PL/M high-level language for system programming, instead of assembly language, results in a high degree of engineering productivity during project development. This translates into significant reductions in initial software development and follow-up maintenance costs for the user.

PL/M 8086 is available to run on the Intellec® Microcomputer Development Systems (Series III/Series IV) as well as the IBM PC XT/AT, DEC VAX†/VMS†, and Intel System 8086/3XX running iRMX™ 86.

†VAX, VMS are trademarks of Digital Equipment Corporation.

FEATURES

Major features of the Intel PL/M 8086 compiler and programming language include:

Block Structure

PL/M source code is developed in a series of modules, procedures, and blocks. Encouraging program modularity in this manner makes programs more readable, and easier to maintain and debug. The language becomes more flexible, by clearly defining the scope of user variables (local to a private procedure).

The use of procedures to break down a large problem is paramount to productive software development. The PL/M 8086 implementation of a block structure allows the use of REENTRANT (recursive) procedures, which are especially useful in system design.

Language Compatibility

PL/M 8086 object modules are compatible with object modules generated by all other 8086 translators. This means that PL/M programs may be linked to programs written in any other 8086 language.

Object modules are compatible with In-Circuit Emulators; DEBUG compiler control provides the In-Circuit Emulators with symbolic debugging capabilities.

PL/M 8086 Language is upward compatible with PL/M 80, so that application programs may be easily ported to run on the 8086.

Supports Seven Data Types

PL/M makes use of seven data types for various applications. These data types range from one to four bytes, and facilitate various arithmetic, logic, and addressing functions:

- Byte: 8-bit unsigned number
- Word: 16-bit unsigned number
- DWORD: 32-bit unsigned number
- Integer: 16-bit signed number
- Read: 32-bit floating point number
- Pointer: 16-bit or 32-bit memory address indicator
- Selector: 16-bit base portion of a pointer

Another powerful facility allows the use of BASED variables that map more than one variable to the same memory location. This is especially useful for passing parameters, relative and absolute addressing, and memory allocation.

Two Data Structuring Facilities

In addition to the five data types and based variables, PL/M supports two data structuring facilities. These help the user to organize data into logical groups.

- Array: Indexed list of same type of data elements
- Structure: Named collection of same or different type data elements
- Combinations of Each: Arrays of structures or structures of arrays

8087 Numerics Support

PL/M programs that use 32-bit REAL data may be executed using the Numeric Data Processor for improved performance. All floating-point operations supported by PL/M may be executed on the 8086/20 or 8088/20 NDP, or the 8087 Emulator (a software module) provided with the package. Determination of use of the chip or Emulator takes place at linktime, allowing compilations to be run-time independent.

Built-In String Handling Facilities

The PL/M 8086 language contains built-in functions for string manipulation. These byte and word functions perform the following operations on character strings: MOVE, COMPARE, TRANSLATE, SEARCH, SKIP, and SET.

Interrupt Handling

PL/M has the facility for handling interrupts. A procedure may be defined with the INTERRUPT attribute, and the compiler willl automatically initialize an interrupt vector at the appropriate memory location. The compiler will also generate code to save and restore the processor status, for execution of the user-defined interrupt handler routine. The procedure SET$INTERRUPT, the function retuning an INTERRUPT$PTR, and the PL/M statement CAUSE$INTERRUPT all add flexibility to user programs involving interrupt and handling.

Compiler Controls

Including several that have been mentioned, the PL/M 8086 compiler offers more than 25 controls that facilitate such features as:

- Conditional compilation
- Including additional PL/M source files from disk
- Corresponding assembly language code in the listing file
- Setting overflow conditions for run-time handling

Segmentation Control

The PL/M 8086 compiler takes full advantage of program addressing with the SMALL, COMPACT, MEDIUM, and LARGE segmentation controls. Programs with less than 64 KB total code space can exploit the most efficient memory addressing schemes, which lowers total memory requirements. Larger programs can exploit the flexibility of extended one-megabyte addressing.

Code Optimization

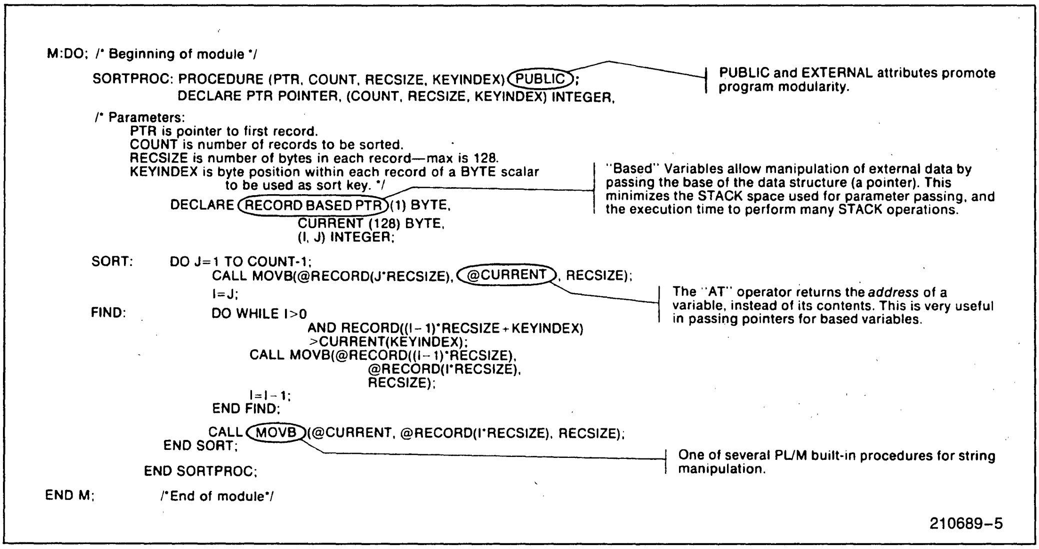
The PL/M 8086 compiler offers four levels of optimization for significantly reducing overall program size.

- Combination or "folding" of constant expressions; and short-circuit evaluation of Boolean expressions
- "Strength reductions" (such as a shift left rather than multiply by 2); and elimination of common sub-expressions within the same block
- Machine code optimizations; elimination of superfluous branches; re-use of duplicate code; removal of unreachable code
- Byte comparisons (rather than 20-bit address calculations) for pointer variables; optimization of based-variable operations

Error Checking

The PL/M 8086 compiler has a very powerful feature to speed up compilations. If a syntax or program error is detected, the compiler will skip the code generation and optimization passes. This usually yields a 2X performance increase for compilation of programs with errors.

A fully detailed set of programming and compilation errors is provided by the compiler.



```
M:DO; /* Beginning of module */
      SORTPROC: PROCEDURE (PTR, COUNT, RECSIZE, KEYINDEX) PUBLIC;
                DECLARE PTR POINTER, (COUNT, RECSIZE, KEYINDEX) INTEGER,
      /* Parameters:
           PTR is pointer to first record.
           COUNT is number of records to be sorted.
           RECSIZE is number of bytes in each record—max is 128.
           KEYINDEX is byte position within each record of a BYTE scalar
                     to be used as sort key. */
              DECLARE RECORD BASED PTR (1) BYTE,
                               CURRENT (128) BYTE,
                               (I, J) INTEGER;

      SORT:   DO J=1 TO COUNT-1;
                  CALL MOVB(@RECORD(J*RECSIZE), @CURRENT, RECSIZE);
                  I=J;
      FIND:       DO WHILE I>0
                            AND RECORD((I-1)*RECSIZE+KEYINDEX)
                            >CURRENT(KEYINDEX);
                      CALL MOVB(@RECORD((I-1)*RECSIZE),
                                      @RECORD(I*RECSIZE),
                                      RECSIZE);
                      I=I-1;
                  END FIND;
                 CALL MOVB(@CURRENT, @RECORD(I*RECSIZE), RECSIZE);
             END SORT;
           END SORTPROC;
END M;       /*End of module*/
```

PUBLIC and EXTERNAL attributes promote program modularity.

"Based" Variables allow manipulation of external data by passing the base of the data structure (a pointer). This minimizes the STACK space used for parameter passing, and the execution time to perform many STACK operations.

The "AT" operator returns the *address* of a variable, instead of its contents. This is very useful in passing pointers for based variables.

One of several PL/M built-in procedures for string manipulation.

210689-5

Figure 3. Sample PL/M 8086 Program

BENEFITS

PL/M 8086 is designed to be an efficient, cost-effective solution to the special requirements of 8086 Microsystem Software Development, as illustrated by the following benefits of PL/M use:

Cost-Effective Alternative to Assembly Language

PL/M 8086 programs are code efficient. PL/M 8086 combines all of the benefits of a high-level language (ease of use, high productivity) with the ability to access the 8086 architecture. Consequently, for the development of systems software, PL/M 8086 is the cost-effective alternative to assembly language programming.

Low Learning Effort

PL/M is easy to learn and to use, even for the novice programmer.

Earlier Project Completion

Critical projects are completed much earlier than otherwise possible because PL/M 8086, a structured high-level language, increases programmer productivity.

Lower Development Cost

Increases in programmer productivity translate immediately into lower software development costs because fewer programming resources are required for a given programmed function.

Increased Reliability

PL/M 8086 is designed to aid in the development of reliable software (PL/M 8086 programs are simple statements of the program algorithm). This substantially reduces the risk of costly correction of errors in systems that have already reached full production status, as the more simply stated the program is, the more likely it is to perform its intended function.

Easier Enhancements and Maintenance

Programs written in PL/M tend to be self-documenting, thus easier to read and understand. This means it is easier to enhance and maintain PL/M programs as the system capabilities expand and future products are developed.

SPECIFICATIONS

Documentation Package

PL/M-8086 User's Guide for 8086-based Development Systems

SUPPORT:

Hotline Telephone Support, Software Performance Reporting (SPR), Software Updates, Technical Reports, Monthly Newsletter available.

ORDERING INFORMATION

Order Code	Operating Environment
D86PLM86	IBM PC XT/AT running PCDOS Version 3.0 or later
R86PLM86	Intel System 8086/3XX running iRMX™ 86
WSPLM86	VAX/VMS
MVVSPLM86	MICROVAX/VMS

iC-86/186
C COMPILER FOR THE 8086 AND THE 80186

- **Implements Full C Language as Defined by the Draft ANSI Standard**
- **Produces High Density Code Rivaling Assembler**
- **Supports Both Standard Intel (PL/M-like) and Standard C Calling Conventions**
- **Allows Mixed Memory Mode/ Programming via Near and Far Pointers**
- **Available for DOS and VAX/VMS* Operating System**
- **Designed to Work with Intel Debuggers such as I²ICE and PSCOPE**
- **Supports Small, Medium, Compact, and Large Models of Computation**
- **Supports IEEE Floating Point Math with 8087 Coprocessor**
- **Supports I/O and Hardware Interrupts Directly in C**
- **Supports Full Standard I/O Library (STDIO)**
- **Written in C**
- **All Code and Libraries Are Fully Compatible**

The C Programming Language was originally designed in 1972 and has become increasingly popular as a systems development language. C combines the flexibility and programming speed of a higher level language with the efficiency and control of assembly language.

Intel iC-86 brings the full power of the C programming language to 8086, 8088, 80186, and 80188 based microprocessor systems. iC-86 has been developed specifically for embedded microprocessor-based applications.

Intel iC-86 supports the full C language as described in the Kernighan and Ritchie book, "The C Programming Language", (Prentice-Hall, 1978). iC-86 implements the complete C language specification as defined in the ANSI X3J11 standard.

iC-86 is an outstanding microprocessor system implementation language because it provides:

1. the ability to manipulate the fundamental objects of the machine (including machine addresses) as easily as assembly language.

2. the power and speed of a structured language supporting a large number of data types, storage classes, expressions and statements,

3. processor independence (most programs developed for other processors can be easily transported to the 8086), and

4. code that rivals assembly language in efficiency

INTEL iC-86 COMPILER DESCRIPTION

The iC-86 compiler operates in four phases: preprocessor, parser, code generator, and optimizer. The preprocessor phase interprets directives in C source code, including conditional compilations (# define). The parser phase converts the C program into an intermediate free form and does all syntactic and semantic error checking. The code generator phase converts the parser's output into an efficient intermediate binary code, performs constant folding, and features an extremely efficient register allocator, ensuring high quality code. The optimizer phase converts the output of the code generator into relocatable Intel Object Module Format (OMF) code, without creating an intermediate assembly file. Optionally, the iC-86 compiler can produce a symbolic pseudo-assembly file. The iC-86 optimizer eliminates common code, eliminates redundant loads and stores, and resolves span dependencies (shortens branches) within a program.

The iC-86 runtime library consists of a number of functions which the C programmer can call. The runtime system includes the standard I/O library (STDIO), conversion routines, routines for manipulating strings, special routines to perform functions not available on the 8086 (32-bit arithmetic and emulated floating point), and (where appropriate) routines for interfacing with the operating system.

iC-86 uses Intel's linker and locator and generates debug records for symbols and lines on request, permitting access to Intel's PSCOPE AND I^2ICE^{TM} to aid in program testing. Intel's DOS LINK86 can also be used to create DOS executable .EXE files for prototyping.

FEATURES

Memory Model Support

iC-86 supports the SMALL, MEDIUM, COMPACT, and LARGE segmentation models. A SMALL Model Program can have up to 64K bytes of code space and 64K bytes of total data, memory, and stack space for all combined modules. SMALL model will generate the most efficient code and is the compiler default. A MEDIUM Model Program can have a seperate 64K segment for each module of code, while total data, memory, and stack must be less than 64K. In the COMPACT model code, data, stack, and memory can each reside in a separate 64K segment. The LARGE model is intended for programs needing up to 64K of code space and 64K of data space for each module. LARGE model also provides up to 64K of stack space and up to 64K of space for memory. Mixed model programming is supported with "near" and "far" calls.

Preprocessor Directives

#define—defines a macro

#include—includes code outside of the program source file

#if—conditionally includes or excludes code

Other preprocessor directives include #undef, #ifdef, #ifndef, #else, #endif, and #line.

Statements

The C language supports a variety of statements:

Conditionals: IF, IF-ELSE

Loops: WHILE, DO-WHILE, FOR

Selection of cases: SWITCH, CASE, DEFAULT

Exit from a function: RETURN

Loop control: CONTINUE, BREAK

Branching: GOTO

Expressions and Operators

The C language includes a rich set of expressions and operators.

Primary expression: invoke functions, select elements from arrays, and extract fields from structures or unions

Arithmetic operators: add, subtract, multiply, divide, modulus

Relational operators: greater than, greater than or equal, less than, less than or equal, not equal

Unary operators: indirect through a pointer, compute an address, logical negation, ones complement, provide the size in bytes of an operand.

Logical operators: AND, OR

Bitwise operators: AND, exclusive OR, inclusive OR, bitwise complement

Calling Conventions

iC-86 provides two distinct calling conventions for handling the way parameters are passed on the stack. The *variable parameter list* (VPL) is the default, and is consistent with most other C compilers. VPL pushes the last (rightmost) parameter first, and the first parameter is pushed last. The *fixed parameter list* (FPL) is the calling convention for most other Intel compilers, including PL/M. FPL pushes the first parameter first, and the last parameter last. By using the keyword "alien", the user can make direct PL/M calls.

Data Types and Storage Classes

Data in C is described by its type and storage class. The type determines its representation and use, and the storage class determines its lifetime, scope, and storage allocation. The following data types are fully supported by iC-86.

char

an 8-bit signed integer

int

a 16-bit signed integer

short

same as int (on the 8086)

long

a 32-bit signed integer

unsigned

a modifier for integer data types (char, int, short, and long) which doubles the positive range of values

float

a 32-bit floating point number which utilizes the 8087 or a software floating point library

double

a 64-bit floating point number

bit-field

maximum size is that of an int

void

a special type that cannot be used as an operator; normally used for functions called only for effect (to prevent their use in contexts where a value is required).

enum

an enumerated data type

These fundamental data types may be used to create other data types including: arrays, functions, structures, pointers, and unions.

The storage classes available in iC-86 include:

register

suggests that a variable be kept in a machine register, often enhancing code density and speed

extern

a variable defined outside of the function where it is declared; retaining its value throughout the entire program and accessible to other modules

auto

a local variable, created when a block of code is entered and discarded when the block is exited

static

a local variable that retains its value until the termination of the entire program

typedef

defines a new data type name from existing data types

BENEFITS

Faster Compilation

Intel iC-86 compiles C programs substantially faster than standard C compilers because it produces Intel OMF code directly, eliminating the traditional intermediate process of generating an assembly file.

Portability of Code

Because Intel iC-86 supports the STDIO and produces Intel OMF code, programs developed on a variety of machines can easily be transported to the 8086.

Rapid Program Development

Intel iC-86 provides the programmer with detailed error messages and access to PSCOPE-86 and I^{2}ICE to speed program development. A complete listing file can also be produced.

Full Manipulation of the 8086 and 80186

Intel iC-86 enables the programmer to utilize features of the C language to control bit fields, pointers, addresses and register allocation, taking full advantage of the fundamental concepts of the 8086. A MOD186 control is also available to provide full support for the additional instructions in the 80186.

SPECIFICATIONS

Operating Environment

The iC-86 compiler runs host resident on DOS 3.0 or greater. iC-86 can also run as a cross compiler on a VAX 11/780 computer under the VMS operating system. 640K bytes of User Memory is required on all versions. The PC DOS Operating Environment is also supported. Specify desired version when ordering.

Required Hardware

VAX version:

— Digital Equipment Corporation VAX 11/780 or compatible computer running VMS 4.5 or greater

PC DOS version:

— PC XT or AT using PC DOS 3.0 or later

— Hard disk recommended

Required Software

MicroVAX or VAX version:

— VMS Operating System 4.5 or greater

PC DOS version:

— PC DOS Release 3.0 or later Operating System

Documentation Package

iC-86 User Manual

C: A Reference Manual by Harbison and Steele (1987 Prentice-Hall)

Shipping Media

VAX version:

— 1600 bpi, 9 track Magnetic tape

DOS version:

— 5¼″ DOS format diskette

— 3.5″ DOS format diskette

ORDERING INFORMATION

Order Code	Description
MVVSC86	iC-86 Cross Compiler for MicroVAX/VMS
VVSC86	iC-86 Cross Compiler for VAX/VMS
D86C86	iC-86 Compiler for PC DOS

Intel Software License required for VAX and MicroVAX versions

SUPPORT

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

*MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.

VAX, VMS are registered trademarks of Digital Equipment Corporation.

VAX*/VMS* RESIDENT 8086/88/186 SOFTWARE DEVELOPMENT PACKAGES

- **Executes on DEC VAX*/MicroVAX Minicomputer under VMS* Operating System to translate PL/M-86, Pascal-86 and ASM-86 Programs for 8086, 88 and 186 Microprocessors.**
- **Packages include C-86; FORTRAN-86; Pascal-86; PL/M-86; ASM-86; Link and Relocation Utilities; OH-86 Absolute Object Module to Hexadecimal Format Converter; and Library Manager Program.**
- **Output linkable with Code Generated on Intellec® Development Systems.**

The VAX/VMS Resident Software Development Packages contain software development tools for the 8086, 88, and 186 microprocessors. The package lets the user develop, compile, maintain libraries, and link and locate programs on a VAX running the VMS operating system. The translator output is object module compatible with programs translated by the corresponding version of the translator on an Intellec Development System.

Four packages are available:

1. An ASM-86 Assembler Package which includes the Assembler, the Link Utility, the Locate Utility, the absolute object to hexadecimal format conversion utility and the Library Manager Program.
2. A PL/M-86 Compiler Package which contains the PL/M-86 Compiler and Runtime Support Libraries.
3. A Pascal-86 Compiler Package which contains the Pascal-86 Compiler and Runtime Support Libraries.
4. A C-86 Compiler Package which contains the C-86 Compiler and Run-Time Libraries.
5. A FORTRAN-86 Compiler Package which contains the FORTRAN-86 Compiler and Run-Time Libraries.

The VAX/VMS resident development packages and the Intellec Development System development packages are built from the same technology base. Therefore, the VAX/VMS resident development packages and the Intellec Development System development packages are very similar.

Version numbers can be used to identify features correspondence. The VAX/VMS resident development packages will have the same features as the Intellec Development System product with the same version number.

The object modules produced by the translators contain symbol and type information for programming debugging using ICE™ translators and/or the PSCOPE debugger. For final production version, the compiler can remove this extra information and code.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

September 1987
Order Number: 210643-004

VAX*-PL/M-86/88/186 SOFTWARE PACKAGE

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Supports 16-Bit Signed Integer and 32-Bit Floating Point Arithmetic in Accordance with IEEE Proposed Standard**
- **Easy-To-Learn Block-Structured Language Encourages Program Modularity**
- **Produces Relocatable Object Code Which is Linkable to All Other Intel 8086 Object Modules, Generated on Either a VAX*, a PC XT/AT running PC-DOS Version 3.0 or Intellec® Development Systems**
- **Code Optimization Assures Efficient Code Generation and Minimum Application Memory Utilization**
- **Built-In Syntax Checker Doubles Performance for Compiling Programs Containing Errors**
- **Source Input/Object Output Compatible with PL/M-86 Hosted on an Intellec® Development System**
- **ICE™, PSCOPE Symbolic Debugging Fully Supported**

Like its counterpart for MCS®-80/85 program development, and Intellec® hosted 8086 program development, VAX-PL/M-86 is an advanced, structured high-level programming language. The VAX-PL/M-86 compiler was created specifically for performing software development for the Intel 8086, 88 and 186 Microprocessors.

PL/M is a powerful, structured, high-level system implementation language in which program statements can naturally express the program algorithm. This frees the programmer to concentrate on the logic of the program without concern for burdensome details of machine or assembly language programming (such as register allocation, meanings of assembler mnemonics, etc.).

The VAX-PL/M-86 compiler efficiently converts free-form PL/M language statements into equivalent 8086/88/186 machine instructions. Substantially fewer PL/M statements are necessary for a given application than if it were programmed at the assembly language or machine code level.

The use of PL/M high-level language for system programming, instead of assembly language, results in a high degree of engineering productivity during project development. This translates into significant reductions in initial software development and follow-on maintenance costs for the user.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX*-PASCAL-86/88 SOFTWARE PACKAGE

- **Executes VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Produces Relocatable Object Code Which is Linkable to All Other Intel 8086 Object Modules, Generated on Either a VAX*, a PC XT/AT running PC-DOS Version 3.0 or Intellec® Development Systems**
- **ICE™, PSCOPE Symbolic Debugging Fully Supported**
- **Implements REALMATH for Consistent and Reliable Results**
- **Supports 8086/20, 88/20 Numeric Data Processors**
- **Strict Implementation of ISO Standard Pascal**
- **Useful Extensions Essential for Microcomputer Applications**
- **Separate Compilation with Type-Checking Enforced between Pascal Modules**
- **Compiler Option to Support Full Run-Time Range-Checking**
- **Source Input/Object Output Compatible with Pascal-86 Hosted on a Intellec® Development System**

VAX-PASCAL-86 conforms to and implements the ISO Pascal standard. The language is enhanced to support microcomputer applications with special features, such as separate compilation, interrupt handling and direct port I/O. Other extensions include additional data types not required by the standard and miscellaneous enhancements such as an allowed underscore in names, an OTHERWISE clause in CASE construction and so forth. To assist the development of portable software, the compiler can be directed to flag all non-standard features.

The VAX-PASCAL-86 compiler runs on the Digital Equipment Corporation VAX under the VMS Operating System. A well-defined I/O interface is provided for run-time support. This allows a user-written operating system to support application programs on the target system as an alternate to the development system environment. Program modules compiled under PASCAL-86 are compatible and linkable with modules written in PL/M-86, and ASM-86. With a complete family of compatible programming languages for the 8086, 88, and 186 one can implement each module in the language most appropriate to the task at hand.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX* 8086/88/186 MACRO ASSEMBLER

- **Executes on VAX*/MicroVAX Minicomputers under The VMS* Operating System**
- **Produces Relocatable Object Code Which is Linkable to All Other Intel 8086/88/186 Object Modules, Generated on Either a VAX*, a PC XT/AT running PC-DOS Version 3.0 or Intellec® Development Systems**
- **Powerful and Flexible Text Macro Facility with Three Macro Listing Options to Aid Debugging**
- **Highly Mnemonic and Compact Language, Most Mnemonics Represent Several Distinct Machine Instructions**
- **"Strongly Typed" Assembler Helps Detect Errors at Assembly Time**
- **High-Level Data Structuring Facilities Such as "STRUCTURES" and "RECORDS"**
- **Over 120 Detailed and Fully Documented Error Messages**
- **Produces Relocatable and Linkable Object Code**
- **Source Input/Object Output Compatible with ASM-86 hosted on an Intellec® Development System**

VAX-ASM-86 is the "high-level" macro assembler for the 8086/88/186 assembly language. VAX-ASM-86 translates symbolic 8086/88/186 assembly language mnemonics into 8086/88/186 relocatable object code.

VAX-ASM-86 should be used where maximum code efficiency and hardware control is needed. The 8086/88/186 assembly language includes approximately 100 instruction mnemonics. From these few mnemonics the assembler can generate over 3,800 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 100 mnemonics to generate all possible 8086/88/186 machine instructions. VAX-ASM-86 will generate the shortest machine instruction possible given no forward referencing or given explicit information as to the characteristics of forward referenced symbols.

VAX-ASM-86 offers many features normally found only in high-level languages. The 8086/88/186 assembly language is strongly typed. The assembler performs extensive checks on the usage of variable and labels. The assembler uses the attributes which are derived explicity when a variable or label is first defined, then makes sure that each use of the symbol in later instructions conforms to the usage defined for that symbol. This means that many programming errors will be deteced when the program is assembled, long before it is being debugged on hardware.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX*-LIB-86

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **VAX-LIB-86 is a Library Manager Program which Allows You to:**
  Create Specifically Formatted Files to Contain Libraries of Object Modules
  Maintain These Libraries by Adding or Deleting Modules
  Print a Listing of the Modules and Public Symbols in a Library File
- **Libraries Can be Used as Input to VAX-LINK-86 Which Will Automatically Link Modules from the Library that Satisfy External References in the Modules Being Linked**
- **Abbreviated Control Syntax**

Libraries aid in the job of building programs. The library manager program VAX-LIB-86 creates and maintains files containing object modules. The operation of VAX-LIB-86 is controlled by commands to indicate which operation VAX-LIB-86 is to perform. The commands are:

CREATE:	creates an empty library file
ADD:	adds object modules to a library file
DELETE:	deletes modules from a library file
LIST:	lists the module directory of library files
EXIT:	terminates the LIB-86 program and returns control to VMS

When using object libraries, the linker will call only those object modules that are required to satisfy external references, thus saving memory space.

VAX-OH-86

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Converts an 8086/88/186 Absolute Object Module to Symbolic Hexademical Format**
- **Facilitates Preparing a file for Loading by Symbolic Hexadecimal Loader (e.g. iSBC® Monitor SDK-86 Loader), or Universal PROM Mapper**
- **Converts an Absolute Module to a More Readable Format that can be Displayed on a CRT or Printed for Debugging**

The VAX-OH-86 utility converts an 86/88 absolute object module to the hexadecimal format. This conversion may be necessary for later loading by a hexadecimal loader such as the iSBC 86/12 monitor or the Universal PROM Mapper. The conversion may also be made to put the module in a more readable format that can be displayed or printed.

The module to be converted must be in absolute form; the output from VAX-LOC-86 is in absolute format.

*VAX, VMS are trademarks of Digital Equipment Corporation.

VAX*-LINK-86

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Automatic Combination of Separately Compiled or Assembled 86/88/186 Programs into a Relocatable Module, Generated on Either a VAX, a PC XT/AT running PC-DOS Version 3.0 or an Intellec® Development System**
- **Automatic Selection of Required Modules from Specified Libraries to Satisfy Symbolic References**
- **Extensive Debug Symbol Manipulation, allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**
- **Automatic Generation of a Summary Map Giving Results of the LINK-86 Process**
- **Abbreviated Control Syntax**
- **Relocatable modules may be Merged into a Single Module Suitable for Inclusion in a Library**
- **Supports "Incremental" Linking**
- **Supports Type Checking of Public and External Symbols**

VAX-LINK-86 combines object modules specified in the VAX-LINK-86 input list into a single output module. VAX-LINK-86 combines segments from the input modules according to the order in which the modules are listed.

VAX-LINK-86 will accept libraries and object modules built from VAX-PL/M-86, VAX-PASCAL-86, VAX-ASM-86, or any other Intel translator generating 8086 Relocatable Object Modules, such as the Series III resident translators.

Support for incremental linking is provided since an output module produced by VAX-LINK-86 can be an input to another link. At each stage in the incremental linking process, unneeded public symbols may be purged.

VAX-LINK-86 supports type checking of PUBLIC and EXTERNAL symbols reporting a warning if their types are not consistent.

VAX-LINK-86 will link any valid set of input modules without any controls. However, controls are available to control the output of diagnostic information in the VAX-LINK-86 process and to control the content of the output module.

VAX-LINK-86 allows the user to create a large program as the combination of several smaller, separately compiled modules. After development and debugging of these component modules the user can link them together, locate them using VAX-LOC-86 and enter final testing with much of the work accomplished.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX*-LOC-86

- **Executes on the VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Automatic Generation of a Summary Map Giving Starting Address, Segment Addresses and Length, and Debug Symbols and their Addresses**
- **Extensive Capability to Manipulate the Order and Placement of Segments in 8086/8088 Memory**
- **Abbreviated Control Syntax**
- **Automatic and Independent Relocation of Independent Relocation of Segments. Segments May be Relocated to Best Match Users Memory Configuration**
- **Extensive Debug Symbol Manipulation, Allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**

Relocatability allows the programmer to code programs or sections of programs without having to know the final arrangement of the object code in memory.

VAX-LOC-86 converts relative addresses in an input module in iAPX-86/88/186 object module format to absolute addresses. VAX-LOC-86 orders the segments in the input module and assigns absolute addresses to the segments. The sequence in which the segments in the input module are assigned absolute addresses is determined by their order in the input module and the controls supplied with the command.

VAX-LOC-86 will relocate any valid input module without any controls. However, controls are available to control the output of diagnostic information in the VAX-LOC-86 process, to control the content of the output module, or both.

The program you are developing will almost certainly use some mix of random access memory (RAM), read-only memory (ROM), and/or programmable read-only memory (PROM). Therefore, the location of your program affects both cost and performance in your application. The relocation feature allows you to develop your program and then simply relocate the object code to suit your application.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

SPECIFICATIONS

Operating Environment

Required Hardware

VAX* 11/780, 11/782, 11/750, or 11/730 9 Track Magnetic Tape Drive, 1600 BPI

MicroVAX II with TK-50 tape drive.

Required Software

VMS Operating System V3.0 or Later. All of the development packages are delivered as unlinked VAX object code which can be linked to VMS as designed for the system where the development package is to be used. VMS command files to perform the link are provided.

MicroVMS (V4.4 or later).

Documentation Package

iAPX-86, 88 Development Software Installation Manual and User's Guide for VAX/VMS, Order number 121950-001

Shipping Media

9 Track Magnetic Tape 1600 bpi (VAX)

TK-50 Cartridge Tape (MicroVAX)

ORDERING INFORMATION

Part Number	Description
VVSASM86	VAX-ASM-86, VAX-LINK-86, VAX-LOC-86, VAX-LIB-86, VAX-OH-86, Package
VVSPLM86	VAX-PLM-86 Package
iMDX-344VX	VAX-PASCAL-86 Package
VVSC86	VAX-C-86 Package
MVVSASM86	MICROVAX ASM86 Package
MVVSPLM86	MICROVAX PLM86 Package
MVVSC86	MICROVAX C86 Package
MVVSFORT86	MICROVAX FORTRAN 86 Package

REQUIRES SOFTWARE LICENSE

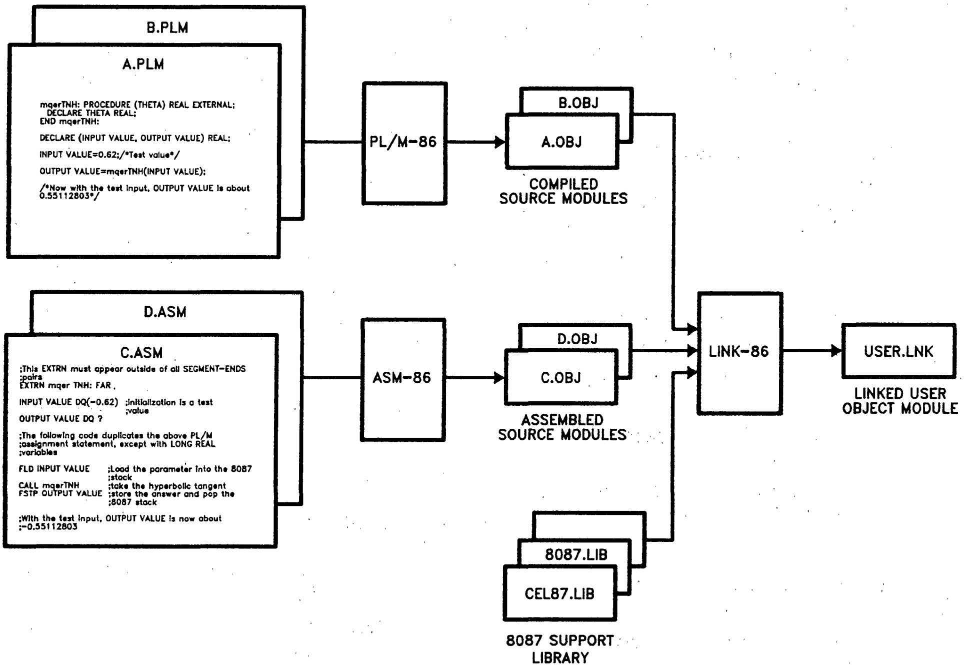
intel®

8087 SUPPORT LIBRARY

- **Library to Support Floating Point Arithmetic in Pascal-86, PL/M-86, FTN-86 and ASM-86**
- **Decimal Conversion Library Supports Binary-Decimal Conversions**
- **Supports Proposed IEEE Floating Point Standard for High Accuracy and Software Portability**
- **Common Elementary Function Library Provides Trigonometric, Logarithmic and Other Useful Functions**
- **Error-Handler Module Simplifies Floating Point Error Recovery**

The 8087 Support Library provides Pascal-86, FORTRAN-86, PL/M-86 and ASM-86 users with numeric data processing capability. With the Library, it is easy for programs to do floating point arithmetic. Programs can bind in library modules to do trigonometric, logarithmic and other numeric functions, and the user is guaranteed accurate, reliable results for all appropriate inputs. Figure 1 below illustrates how the 8087 Support Library can be bound with PL/M-86 and ASM-86 user code to do this. The 8087 Support Library supports the proposed IEEE Floating Point Standard. Consequently, by using this Library, the user not only saves software development time, but is guaranteed that the numeric software meets industry standards and is portable–the software investment is maintained.

The 8087 Support Library consists of the common elementary function library (CEL87.LIB), the decimal conversion library (DC87.LIB), the emulator interface library E8087.LIB, the error handler module (EH87.LIB) and interface libraries (8087.LIB, NUL87.LIB).



```
mqerTNH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerTNH:

DECLARE (INPUT VALUE, OUTPUT VALUE) REAL;

INPUT VALUE=0.62;/*Test value*/

OUTPUT VALUE=mqerTNH(INPUT VALUE);

/*Now with the test input, OUTPUT VALUE is about
0.55112803*/
```

```
;This EXTRN must appear outside of all SEGMENT-ENDS
;pairs
EXTRN mqer TNH: FAR .

INPUT VALUE DQ(-0.62)  ;initialization is a test
                       ;value
OUTPUT VALUE DQ ?

;The following code duplicates the above PL/M
;assignment statement, except with LONG REAL
;variables

FLD INPUT VALUE       ;Load the parameter into the 8087
                      ;stack
CALL mqerTNH          ;take the hyperbolic tangent
FSTP OUTPUT VALUE     ;store the answer and pop the
                      ;8087 stack

;With the test input, OUTPUT VALUE is now about
;-0.55112803
```

231613-1

Figure 1. Use of 8087 Support Library with PL/M-86 and ASM-86

August 1985
Order Number: 231613-001

CEL87.LIB
THE COMMON ELEMENTARY FUNCTION LIBRARY

FUNCTIONS

CEL87.LIB contains commonly used floating point functions. It is used along with the 8087 numeric coprocessor. It provides a complete package of elementary functions, giving valid results for all appropriate inputs. Following is a summary of CEL87 functions, grouped by functionality.

Rounding and Truncation Functions:

mqerIEX, mqerIE2, and mqerIE4. Round a real number to the nearest integer; to the even integer if there is a tie. The answer returned is real, a 16-bit integer or a 32-bit integer respectively.

mqerIAX, mqerIA2, mqerIA4. Round a real number to the nearest integer, to the integer away from zero if there is a tie; the answer returned is real, a 16-bit integer or a 32-bit integer, respectively.

mqerICX, mqerIC2, mqerIC4. Truncate the fractional part of a real input; the answer is real, a 16-bit integer or 32-bit integer, repectively.

Logarithmic and Exponential Functions:

mqerLGD computes decimal (base 10) logarithms.

mqerLGE computes natural base (base e) logarithms.

mqerEXP computes exponentials to the base e.

mqerY2X computes exponentials to any base.

mqerY12 raises an input real to a 16-bit integer power.

mqerY14 is as mqerY12, except to a 32-bit integer power.

mqerYIS is as mqerY12, but it accommodates PL/M-286 users.

Trigonometric and Hyperbolic Functions:

mqerSIN, mqerCOS, mqerTAN compute sine, cosine, and tangent.

mqerASN, mqerACS, mqerATN compute the corresponding inverse functions.

mqerSNH, mqerCSH, mqerTNH compute the corresponding hyperbolic functions.

mqerAT2 is a special version of the arc tangent function that accepts rectangular coordinate inputs.

Other Functions (of real variables):

mqerDIM is FORTRAN's positive difference function.

mqerMAX returns the maximum of two real inputs.

mqerMIN returns the minimum of two real inputs.

mqerSGH combines the sign of one input with the magnitude of the other input.

mqerMOD computes a modulus, retaining the sign of the dividend.

mqerRMD computes a modulus, giving the value closest to zero.

Complex Number Functions:

mqercCMUL, and mqercCDIV perform complex multiplication and division of complex numbers.

mqercCPOL converts complex numbers from rectangular to polar form. mqercCREC converts complex numbers from polar to rectangular form.

mqercCSQR, and mqercCABS compute the complex square root and real absolute value (magnitude) of a complex number.

mqercCEXP, and mqercCLGE compute the complex value of e raised to a complex power and the complex natural logarithm (base e) of a complex number.

mqercCSIN, mqercCCOS, and mqercCTAN compute the complex sine, cosine, and tangent of a complex number.

mqercCASN, mqercCACS, and mqercCATN compute the complex inverse sine, cosine, and tangent of a complex number.

mqercCSNH, mqercCCSH, and mqercCTNH compute the complex hyperbolic sine, cosine, and tangent of a complex number.

mqercCACH, mqercCASH, and mqercCATH compute the comples inverse hyperbolic sine, cosine, and tangent of a complex number.

mqercCC2C, mqercCR2C,mqercCC2R,mqercCCI2, mqercCCI4, and mqercCCIS return complex values of complex (or real) values raised to complex (real, short integer, or long integer) values.

DC87.LIB THE DECIMAL CONVERSION LIBRARY

DC87.LIB is a library of procedures which convert binary representations of floating point numbers and ASCII-encoded string of digits.

The binary-to-decimal procedure mqcBIN_DECLOW accepts a binary number in any of the formats used for the representation of floating point numbers in the 8087. Because there are so many output formats for floating point numbers, mqcBIN_DECLOW does not attempt to provide a finished, formatted text string. Instead, it provides the "building blocks" for you to use to construct the output string which meets your exact format specification.

The decimal-to-binary procedure mqcDEC_BIN accepts a text string which consists of a decimal number with optional sign, decimal point, and/or power-of-ten exponent. It translates the string into the caller's choice of binary formats.

Decimal-to-binary procedure mqcDECLOW_BIN is provided for callers who have already broken the decimal number into its constituent parts.

The procedures mqcLONG_TEMP, mqcSHORT_TEMP, mqcTEMP_LONG, and mqcTEMP_SHORT convert floating point numbers between the longest binary format, TEMP_REAL, and the shorter formats.

EH87.LIB THE ERROR HANDLER LIBRARY

EH87.LIB is a library of five utility procedures for writing trap handlers. Trap handlers are called when an unmasked 8087 error occurs.

The 8087 error reporting mechanism can be used not only to report error conditions, but also to let software implement IEEE standard options not directly supported by the chip. The three such extensions to the 8087 are: normalizing mode, non-trapping not-a-number (NaN), and non-ordered comparison. The utility procedures support these extra features.

DECODE is called near the beginning of the trap handler. It preserves the complete state of the 8087, and also identifies what function called the trap handler, and returns available arguments and/or results. DECODE eliminates much of the effort needed to determine what error caused the trap handler to be called.

NORMAL provides the "normalizing mode" capability for handling the "D" exception. By calling NORMAL in your trap handler, you eliminate the need to write code in your application program which tests for non-normal inputs.

SIEVE provides two capabilities for handling the "I" exception. It implements non-trapping NaN's and non-ordered comparisons. These two IEEE standard features are useful for diagnostic work.

ENCODE is called near the end of the trap handler. It restores the state of the 8087 saved by DECODE, and performs a choice of concluding actions, by either retrying the offending function or returning a specified result.

FILTER calls each of the above four procedures. If your error handler does nothing more than detect fatal errors and implement the features supported by SIEVE and NORMAL, then your interface to EH87.LIB can be accomplished with a single call to FILTER.

8087.LIB, NUL87.LIB, E8087.LIB INTERFACE LIBRARIES

E8087.LIB, 8087.LIB and NUL87.LIB libraries configure a user's application program for his run-time environment; running with the 8087 component or without floating point arithmetic, respectively.

FULL 8087 EMULATOR

The Full 8087 Emulator is a 16-kilobyte object module that is linked to the application program for floating-point operations. Its functionality is identical to the 8087 chip, and is ideal for prototyping and debugging floating-point applications. The Emulator is an alternative to the use of the 8087 chip, although the latter executes floating-point applications up to 100 times faster than an 8086 with the 8087 Emulator. Furthermore, since the 8087 is a "coprocessor," use of the chip will allow many operations to be performed in parallel with the 8086.

SPECIFICATIONS

Operating Environment

Intel Microcomputer Development Systems (Series III, Series IV)

Documentation Package

8087 Support Library Reference Manual

ORDERING INFORMATION

Part Number	Description
iMDS 319	8087 Support Library

Requires Software License

SUPPORT

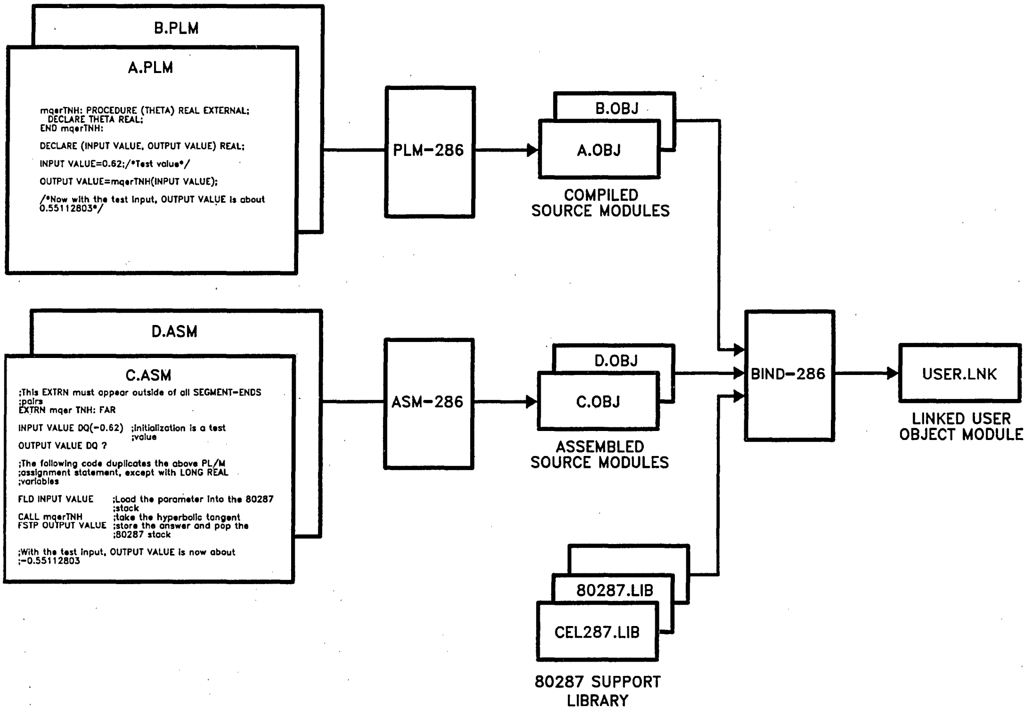
Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

80287 SUPPORT LIBRARY

- **Library to support floating point arithmetic in Pascal-286, PL/M-286 and ASM-286**
- **Decimal conversion library supports binary-decimal conversions**
- **Supports proposed IEEE Floating Point Standard for high accuracy and software portability**
- **Common elementary function library provides trigonometric, logarithmic and other useful functions**
- **Error-handler module simplifies floating point error recovery**

The 80287 Support Library provides Pascal-286, PL/M-286 and ASM-286 users with numeric data processing capability. With the Library, it is easy for programs to do floating point arithmetic. Programs can bind in library modules to do trigonometric, logarithmic and other numeric functions, and the user is guaranteed accurate, reliable results for all appropriate inputs. Figure 1 below illustrates how the 80287 Support Library can be bound with PL/M-286 and ASM-286 user code to do this. The 80287 Support Library supports the proposed IEEE Floating Point Standard. Consequently, by using this Library, the user not only saves software development time, but is guaranteed that the numeric software meets industry standards and is portable–the software investment is maintained.

The 80287 Support Library consists of the common elementary function library (CEL287.LIB), the decimal conversion library (DC287.LIB), the error handler module (EH287.LIB) and interface libraries (80287.LIB, NUL287.LIB).



A.PLM

```
mqerTNH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerTNH:

DECLARE (INPUT VALUE, OUTPUT VALUE) REAL;

INPUT VALUE=0.62;/*Test value*/

OUTPUT VALUE=mqerTNH(INPUT VALUE);

/*Now with the test input, OUTPUT VALUE is about
0.55112803*/
```

C.ASM

```
;This EXTRN must appear outside of all SEGMENT-ENDS
;pairs
EXTRN mqer TNH: FAR

INPUT VALUE DQ(-0.62)  ;Initialization is a test
                       ;value
OUTPUT VALUE DQ ?

;The following code duplicates the above PL/M
;assignment statement, except with LONG REAL
;variables

FLD INPUT VALUE      ;Load the parameter into the 80287
                     ;stack
CALL mqerTNH         ;take the hyperbolic tangent
FSTP OUTPUT VALUE    ;store the answer and pop the
                     ;80287 stack

;With the test input, OUTPUT VALUE is now about
;-0.55112803
```

231041-1

Figure 1. Use of 80287 Support Library with PL/M-286 and ASM-286

CEL287.LIB
THE COMMON ELEMENTARY FUNCTION LIBRARY

FUNCTIONS

CEL287.LIB contains commonly used floating point functions. It is used along with the 80287 numeric coprocessor. It provides a complete package of elementary functions, giving valid results for all appropriate inputs. Following is a summary of CEL287 functions, grouped by functionality.

Rounding and Truncation Functions:

mqerIEX, mqerIE2, and mqerIE4. Round a real number to the nearest integer; to the even integer if there is a tie. The answer returned is real, a 16-bit integer or a 32-bit integer respectively.

mqerIAX, mqerIA2, mqerIA4. Round a real number to the nearest integer, to the integer away from zero if there is a tie; the answer returned is real, a 16-bit integer or a 32-bit integer, respectively.

mqerICX, mqerIC2, mqerIC4. Truncate the fractional part of a real input; the answer is real, a 16-bit integer or 32-bit integer, repectively.

Logarithmic and Exponential Functions:

mqerLGD computes decimal (base 10) logarithms.

mqerLGE computes natural base (base e) logarithms.

mqerEXP computes exponentials to the base e.

mqerY2X computes exponentials to any base.

mqerY12 raises an input real to a 16-bit integer power.

mqerY14 is as mqerY12, except to a 32-bit integer power.

mqerYIS is as mqerY12, but it accommodates PL/M-286 users.

Trigonometric and Hyperbolic Functions:

mqerSIN, mqerCOS, mqerTAN compute sine, cosine, and tangent.

mqerASN, mqerACS, mqerATN compute the corresponding inverse functions.

mqerSNH, mqerCSH, mqerTNH compute the corresponding hyperbolic functions.

mqerAT2 is a special version of the arc tangent function that accepts rectangular coordinate inputs.

Other Functions (of real variables):

mqerDIM is FORTRAN's positive difference function.

mqerMAX returns the maximum of two real inputs.

mqerMIN returns the minimum of two real inputs.

mqerSGH combines the sign of one input with the magnitude of the other input.

mqerMOD computes a modulus, retaining the sign of the dividend.

mqerRMD computes a modulus, giving the value closest to zero.

Complex Number Functions:

mqercCMUL, and mqercCDIV perform complex multiplication and division of complex numbers.

mqercCPOL converts complex numbers from rectangular to polar form. mqercCREC converts complex numbers from polar to rectangular form.

mqercCSQR, and mqercCABS compute the complex square root and real absolute value (magnitude) of a complex number.

mqercCEXP, and mqercCLGE compute the complex value of e raised to a complex power and the complex natural logarithm (base e) of a complex number.

mqercCSIN, mqercCCOS, and mqercCTAN compute the complex sine, cosine, and tangent of a complex number.

mqercCASN, mqercCACS, and mqercCATN compute the complex inverse sine, cosine, and tangent of a complex number.

mqercCSNH, mqercCCSH, and mqercCTNH compute the complex hyperbolic sine, cosine, and tangent of a complex number.

Complex Number Functions: (Continued)

mqercCACH, mqercCASH, and mqercCATH compute the comples inverse hyperbolic sine, cosine, and tangent of a complex number.

mqercCC2C, mqercCR2C, mqercCC2R, mqercCCI2, mqercCCI4, and mqercCCIS return complex values of complex (or real) values raised to complex (real, short integer, or long integer) values.

DC287.LIB THE DECIMAL CONVERSION LIBRARY

DC287.LIB is a library of procedures which convert binary representations of floating point numbers and ASCII-encoded string of digits.

The binary-to-decimal procedure mqcBIN_DECLOW accepts a binary number in any of the formats used for the representation of floating point numbers in the 80287. Because there are so many output formats for floating point numbers, mqcBIN_DECLOW does not attempt to provide a finished, formatted text string. Instead, it provides the "building blocks" for you to use to construct the output string which meets your exact format specification.

The decimal-to-binary procedure mqcDEC_BIN accepts a text string which consists of a decimal number with optional sign, decimal point, and/or power-of-ten exponent. It translates the string into the caller's choice of binary formats.

Decimal-to-binary procedure mqcDECLOW_BIN is provided for callers who have already broken the decimal number into its constituent parts.

The procedures mqcLONG_TEMP, mqcSHORT_TEMP, mqcTEMP_LONG, and mqcTEMP_SHORT convert floating point numbers between the longest binary format, TEMP_REAL, and the shorter formats.

EH287.LIB THE ERROR HANDLER LIBRARY

EH287.LIB is a library of five utility procedures for writing trap handlers. Trap handlers are called when an unmasked 80287 error occurs.

The 80287 error reporting mechanism can be used not only to report error conditions, but also to let software implement IEEE standard options not directly supported by the chip. The three such extensions to the 80287 are: normalizing mode, non-trapping not-a-number (NaN), and non-ordered comparison. The utility procedures support these extra features.

DECODE is called near the beginning of the trap handler. It preserves the complete state of the 80287, and also identifies what function called the trap handler, and returns available arguments and/or results. DECODE eliminates much of the effort needed to determine what error caused the trap handler to be called.

NORMAL provides the "normalizing mode" capability for handling the "D" exception. By calling NORMAL in your trap handler, you eliminate the need to write code in your application program which tests for non-normal inputs.

SIEVE provides two capabilities for handling the "I" exception. It implements non-trapping NaN's and non-ordered comparisons. These two IEEE standard features are useful for diagnostic work.

ENCODE is called near the end of the trap handler. It restores the state of the 80287 saved by DECODE, and performs a choice of concluding actions, by either retrying the offending function or returning a specified result.

FILTER calls each of the above four procedures. If your error handler does nothing more than detect fatal errors and implement the features supported by SIEVE and NORMAL, then your interface to EH287.LIB can be accomplished with a single call to FILTER.

80287.LIB, NUL287.LIB INTERFACE LIBRARIES

80287.LIB and NUL287.LIB libraries configure a user's application program for his run-time environment; running with the 80287 component or without floating point arithmetic, respectively.

SPECIFICATIONS

Operating Environment

Intel Microcomputer Development Systems (Series III, Series IV)

Documentation Package

80287 Support Library Reference Manual

Related Software

A 80287 software emulator is available as part of the 8086 software toolbox (iMDX364)

ORDERING INFORMATION

Part Number	Description
iMDX329	80287 Support Library

Requires Software License

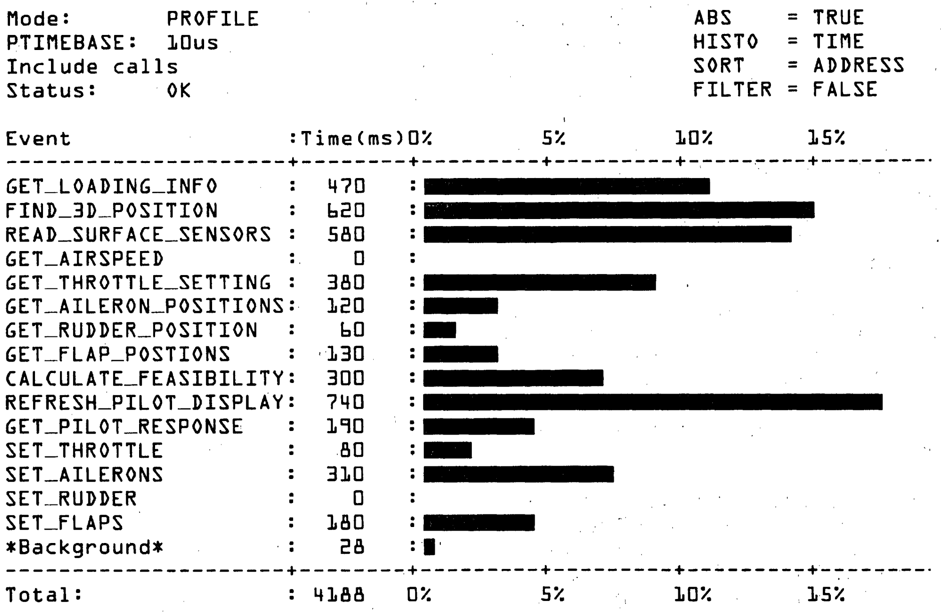
SUPPORT

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

iPAT™ PERFORMANCE ANALYSIS TOOL

- **Provides Real-Time Performance Analysis and Real-Time Test Coverage of Code Written for 8086/8088, 80186/80188, and 80286 Processors**
- **Displays Performance-Analysis Histograms to Isolate Slow Code**
- **Displays Test Coverage Tables to Isolate Untested Code; Permits Saving and Updating Test Results**
- **Measures Interrupt Latency**
- **Does not Intrude Into Program Being Analyzed**
- **Collects 100% of Execution Data**
- **Complements Emulator by Allowing Simultaneous Debugging and Performance Analysis**
- **Permits Activation of Analysis using Emulator Procedures**
- **Handles Up to 24-Bit Execution Address Space**
- **Permits Specification of Analysis Address Ranges Symbolically or with Absolute Addresses**
- **Provides Flexible Isolation of Code Ranges, Windowed Events, and Interrupt Activity**

The Intel Performance Analysis Tool (iPAT™) helps software engineers optimize code and improve software reliability. Software object code generated by Intel assemblers and Intel compilers (e.g., for C, PL/M, Pascal, Ada, and FORTRAN) can be analyzed symbolically to improve software execution efficiency and to validate test coverage. Any object code that lacks Intel compiler information—but that can be run by Intel emulators and for which an absolute program map is available—can also be analyzed (nonsymbolically) by the iPAT analyst. iPAT operation is currently supported via a target interface to the I^{2}ICE™ Integrated Instrumentation and In-Circuit Emulation System.



```
Mode:        PROFILE                                      ABS    = TRUE
PTIMEBASE:   10us                                         HISTO  = TIME
Include calls                                             SORT   = ADDRESS
Status:       OK                                          FILTER = FALSE

Event                  :Time(ms)0%          5%         10%        15%
-----------------------+---------+----------+----------+----------+---------
GET_LOADING_INFO       :  470   :
FIND_3D_POSITION       :  620   :
READ_SURFACE_SENSORS   :  580   :
GET_AIRSPEED           :    0   :
GET_THROTTLE_SETTING   :  380   :
GET_AILERON_POSITIONS  :  120   :
GET_RUDDER_POSITION    :   60   :
GET_FLAP_POSTIONS      :  130   :
CALCULATE_FEASIBILITY  :  300   :
REFRESH_PILOT_DISPLAY  :  740   :
GET_PILOT_RESPONSE     :  190   :
SET_THROTTLE           :   80   :
SET_AILERONS           :  310   :
SET_RUDDER             :    0   :
SET_FLAPS              :  180   :
*Background*           :   28   :
-----------------------+---------+----------+----------+----------+---------
Total:                 : 4188    0%         5%         10%        15%
```

280165–1

PERFORMANCE ANALYSIS INTRODUCTION

The size and complexity of software has increased with each new generation of microprocessors. As a result, it has become increasingly important to optimize software and to ensure its reliability. The iPAT analyst answers these needs.

Optimizing Software

Optimizing software means maximizing software speed without sacrificing functionality or reliability. To increase speed, execution bottlenecks need careful attention. But, how can the crucial slow code be located?

Without the iPAT analyst, you might analyze the various paths in the source code and make educated guesses where the bottlenecks will occur. Or you might place count statements in the code to learn how often the various paths are entered. Neither of these methods can ensure that you really will isolate the bottlenecks. Furthermore, the second method is intrusive—with the extra statements, real-time operation of your original code cannot occur.

The iPAT analyst provides the solution to the software engineering problem of locating crucial code. With the iPAT analyst, you can quickly and easily show (with histograms or tables) timing and count information for specified program modules, procedures, lines, or absolute address ranges. Because it fully supports symbolic information from Intel high-level languages, the iPAT analyst enables you to use the names of procedures and modules to specify ranges that you want to analyze. (For object code that lacks symbolic information, consult your code's absolute program map and then specify absolute address ranges of interest.)

Furthermore, the iPAT analyst is nonintrusive and operates in real-time. It does not sample program operation on a statistical basis; rather, it has available to it each address that is executed so that no potentially troublesome code will be overlooked. (The iPAT analyst can also monitor when interrupts occur.)

Software teams currently doing their coding in assembly language (to ensure speed of program execution) can now consider writing future code in high-level languages. Since much code does not have a significant effect on overall program speed, after the code is written in high-level language, the bottlenecks can be located by the iPAT analyst. Then, if need be, the code causing the bottlenecks can be redone in assembly language. This method of software development means faster product development, since coding can progress much faster using a high-level language.

Measuring Hardware-Interrupt-to-Software-Response Time (Latency)

The iPAT analyst not only allows you to acquire timing and count information on software events; it also allows you to examine hardware-interrupt-to-software interactions. For example, you can measure how long it is before the appropriate service routine is executed in response to a hardware interrupt. If the measured hardware-interrupt-to-software latency period is not acceptable, the iPAT analyst can help you isolate the causes.

Coordinating Performance Analysis with Emulator Controls

Using the emulator with the iPAT analyst also enables you to analyze program execution as a function of differing target-system conditions. You can set up the conditions in the target system with the emulator, set up iPAT data collection for a section of code, then run the program with the iPAT analyst activated. Change the target conditions and repeat program execution and performance analysis.

You can also create emulator procedures (PROCs) containing emulator commands that trigger performance analysis as a function of selected software or hardware events.

Ensuring Software Reliability

As code is developed, there is a need to ensure that it has no defective code. Typically for this purpose, test suites are developed by software engineers. The engineers use their theoretical understanding of the software to devise test suites that will exercise the code paths. Then, the program under test is run with the test suites, and the program's output is examined. If the desired values are present in the output, it is assumed that the paths were tested. But this is an inference; the test results do not themselves show whether the paths were all exercised.

Thus, without the help of the iPAT analyst, testers cannot be confident that their tests exercised all the code. As a result, there may be a tendency to restrict designs to familiar algorithms and techniques, so that previously successful test suites can be reused.

By contrast, the coverage mode in the iPAT analyst enables you to identify easily and quickly which lines or procedures in your software are not being exercised by the test suites. Thus, you need not restrict your test suites or your coding techniques and options. Furthermore, when the iPAT analyst reveals untested code, you can modify your test suites until the iPAT analyst shows that all code is tested.

How the iPAT™ Analyst Affects Development

As your code is being developed, preliminary analyses can be made with the iPAT analyst. Then, when your system hardware is developed to the point that code can be loaded into it and run, the iPAT analyst can make real-time measurements. Refinements of software and test suites can occur up until product release, with each new modification being checked by the iPAT analyst for execution efficiency and reliability.

But, the iPAT analyst's usefulness to the product is not at an end, because most products are enhanced after the first release. As new releases are being prepared (to add new features), the iPAT analyst will be available to analyze the new code and the newest test suites.

The iPAT analyst can also be used to enhance existing products—products that were developed before performance analysis was available. you can examine existing code with the iPAT analyst to identify slow code; recode; re-examine; then, when performance (and reliability) have been improved, release the enhanced products.

The iPAT analyst provides a way for software engineers to check whether the software meets performance specifications. In addition, in the future you will be able to write more meaningful specifications that cite desired iPAT measurements.

If portions of code are likely to be reused, the iPAT analyst can provide measurements of the reusable code's performance characteristics. Then, future users of the code will know in advance what to expect from the code.

Another use of performance analysis is encouraging engineers to engage in "what-if" thinking. They can ask, "What if this portion of the code was designed this way?" Then, after they complete several ways of coding, the various versions can be analyzed by the iPAT analyst to reveal which has the greatest efficiency.

PHYSICAL DESCRIPTION

The iPAT system consists of hardware and software.

Figure 1 shows the iPAT hardware connected to the I^{2}ICE emulation system and hosted by an IBM PC AT. The iPAT hardware includes the following:

- Power supply (with AC and DC power cables)
- Core module
- Emulator-specific target interface (which enables the core module to function with a specific emulator)
- Cable for connecting the core module to the target interface
- RS-232 serial cable for connecting the core module to the host system

iPAT software is integrated with the emulator software. Thus, with the iPAT/I^{2}ICE system target interface you receive I^{2}ICE system host software. (You do not receive I^{2}ICE system probe software; continue to use the probe software—version 1.7 or later—supplied with the I^{2}ICE system.) In addition, you receive iPAT diagnostic and tutorial software.

FUNCTIONAL DESCRIPTION

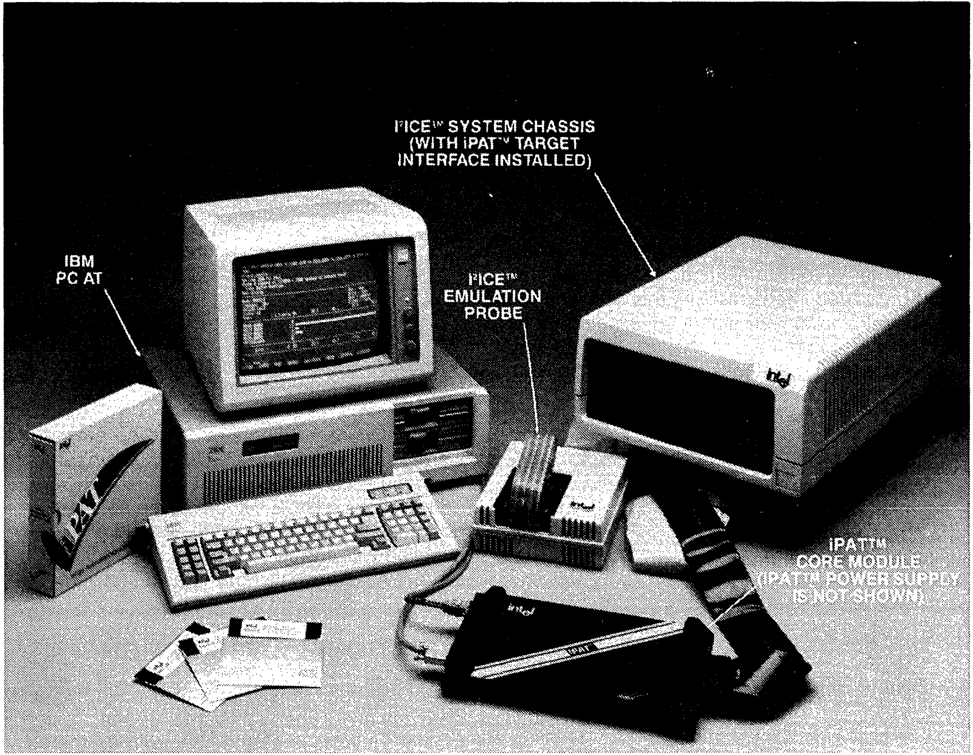
Users will begin analysis of their code by obtaining an overview of their software's operation, and then restrict their focus as they home in on the problem areas in their code. Five analysis modes are available:

- profile
- coverage
- windowed event count
- duration
- linkage

Of these, the profile and coverage modes can be used to acquire both overviews and more localized inspection of your software behavior. The iPAT windowed-event-count, duration, and linkage modes each provide specific perspectives on localized software behavior.

GAINING AN OVERVIEW OF SOFTWARE OPERATION

Gaining an overview of your software operation is simple with the iPAT analyst. If you want an overview of program activity, you load your program, select



280165-2

Figure 1. The iPAT™ Analyst Used with an IBM PC AT

the profile analysis mode, and then run the program. To do so, you need only enter the following commands:

```
LOAD new_program
PAT INIT PROFILE
GO
```

To display the results (during or after program execution), enter:

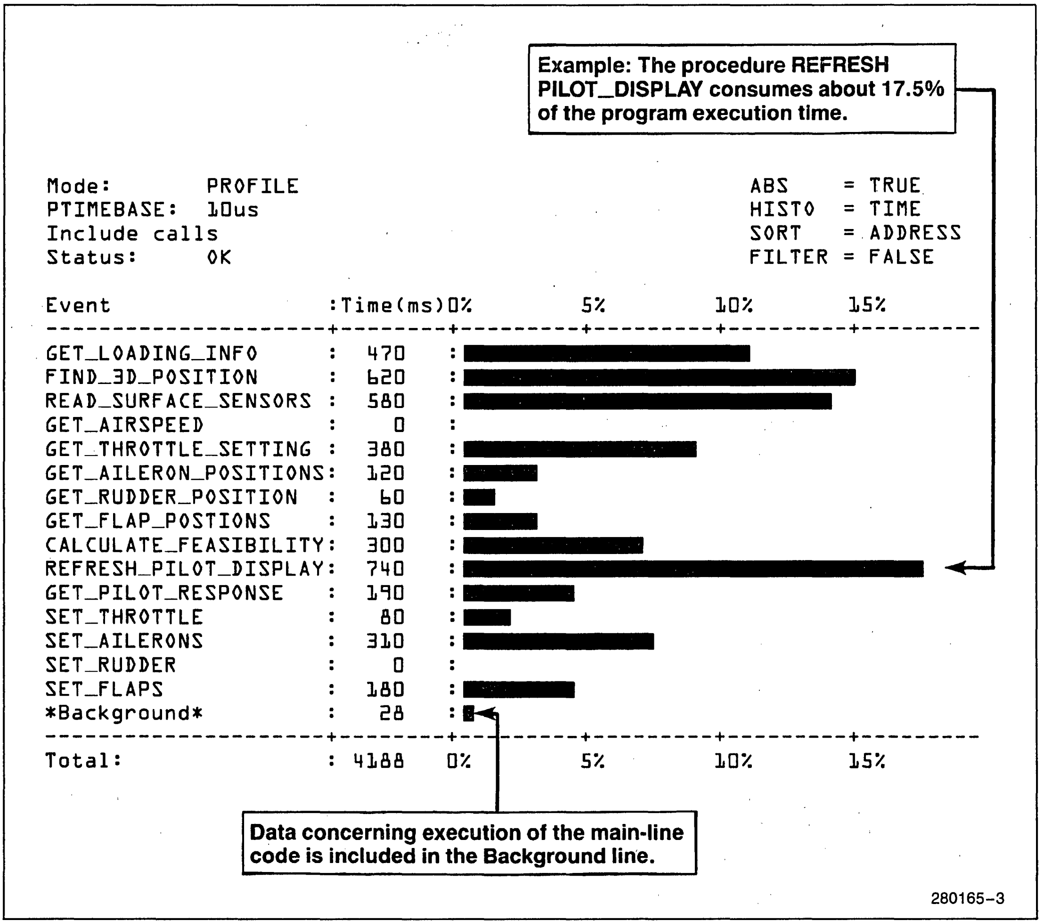
```
PAT DISPLAY
```

iPAT options and controls provide considerable flexibility in monitoring and displaying information about your code. Yet the default settings have been designed with a view to typical applications and ease-of-learning. Default operation in the profile mode monitors all procedures in the user program and measures their real-time characteristics.

The default display for profile mode is a histogram that shows the time spent in each of your program's procedures. See Figure 2 for a sample default profile display.

Acquiring an overview of test coverage is also simple. First set up the coverage mode.

```
PAT INIT COVERAGE
```



Example: The procedure REFRESH PILOT_DISPLAY consumes about 17.5% of the program execution time.

```
Mode:         PROFILE                                   ABS      = TRUE
PTIMEBASE:    10us                                      HISTO    = TIME
Include calls                                           SORT     = ADDRESS
Status:       OK                                        FILTER   = FALSE

Event                    :Time(ms)0%         5%         10%        15%
-------------------------+---------+---------+----------+----------+---------
GET_LOADING_INFO         :  470    :
FIND_3D_POSITION         :  620    :
READ_SURFACE_SENSORS     :  580    :
GET_AIRSPEED             :    0    :
GET_THROTTLE_SETTING     :  380    :
GET_AILERON_POSITIONS    :  120    :
GET_RUDDER_POSITION      :   60    :
GET_FLAP_POSTIONS        :  130    :
CALCULATE_FEASIBILITY    :  300    :
REFRESH_PILOT_DISPLAY    :  740    :
GET_PILOT_RESPONSE       :  190    :
SET_THROTTLE             :   80    :
SET_AILERONS             :  310    :
SET_RUDDER               :    0    :
SET_FLAPS                :  180    :
*Background*             :   28    :
-------------------------+---------+---------+----------+----------+---------
Total:                   : 4188    0%        5%         10%        15%
```

Data concerning execution of the main-line code is included in the Background line.

280165-3

Figure 2. Profile Mode: Time Histogram Display

Then, run your program with the data inputs from your tests suites, and request a display of results using the following commands:

```
GO FROM top
PAT DISPLAY
```

By default, the coverage display lists all procedures and indicates whether each was executed. Figure 3 shows a sample coverage display. It indicates that no code in the procedures GET_AIR_SPEED and SET_RUDDER was executed by the test suites.

GETTING OTHER VIEWS OF SOFTWARE OPERATION

To obtain more refined information about program operation and test coverage, you can use all five analysis modes. For all modes, the basic display command is the same:

```
PAT DISPLAY
```

You can select whether the display should be renewed periodically during real-time program execution. If you select periodic renewal, you can also select how frequently (in seconds) it is renewed.

Data collection occurs with one of five selectable time bases: 100 μs, 10 μs, 1 μs, and 200 ns. The default value is 10 μs.

The following sections describe how each of the five analysis modes and their associated displays can be used to obtain other kinds of overviews and how to localize the collection of data.

Coverage Mode

The default features of the coverage mode have already been described. Once you have a coverage overview, you may want to restrict the data displayed.

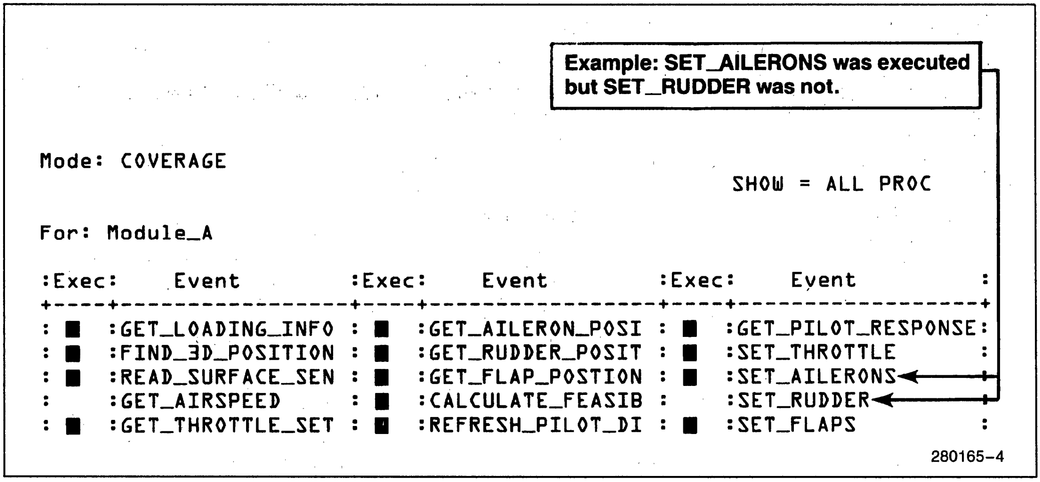
For example, if the default coverage information shows that all procedures were executed by test suites, you may next wish to determine whether all lines in certain procedures were executed. You would then request a display (for the address range desired) of the lines not executed. Using this method, you can obtain very refined test-coverage information and thus help ensure software reliability.

Profile Mode

For profile mode there are a number of ways you can control analysis and the display of data.

Profile-Mode Analysis: For profile mode, data, by default, is collected on program procedures. If you want to acquire an even wider overview, you can change the focus to program modules. Or, for a very close view, you can request that data be collected on the lines executed.

After you have examined your program's profile display, you may notice that several procedures are using excessive time. You will next want to use the iPAT analyst to determine whether the time spent is really attributable to those procedures or rather to calls by those procedures to other procedures. In the default case, when a procedure calls another, the time spent in the called procedure is accumulat-



Example: SET_AILERONS was executed but SET_RUDDER was not.

```
Mode: COVERAGE
                                                    SHOW = ALL PROC

For: Module_A

:Exec:    Event           :Exec:    Event          :Exec:    Event          :
+----+-------------------+----+------------------+----+-------------------+
: ■  :GET_LOADING_INFO  : ■  :GET_AILERON_POSI  : ■  :GET_PILOT_RESPONSE:
: ■  :FIND_3D_POSITION  : ■  :GET_RUDDER_POSIT  : ■  :SET_THROTTLE       :
: ■  :READ_SURFACE_SEN  : ■  :GET_FLAP_POSTION  : ■  :SET_AILERONS
:    :GET_AIRSPEED      : ■  :CALCULATE_FEASIB  :    :SET_RUDDER
: ■  :GET_THROTTLE_SET  : ■  :REFRESH_PILOT_DI  : ■  :SET_FLAPS          :
```

280165-4

Figure 3. Coverage Mode: Display Showing Procedures Executed and Not Executed

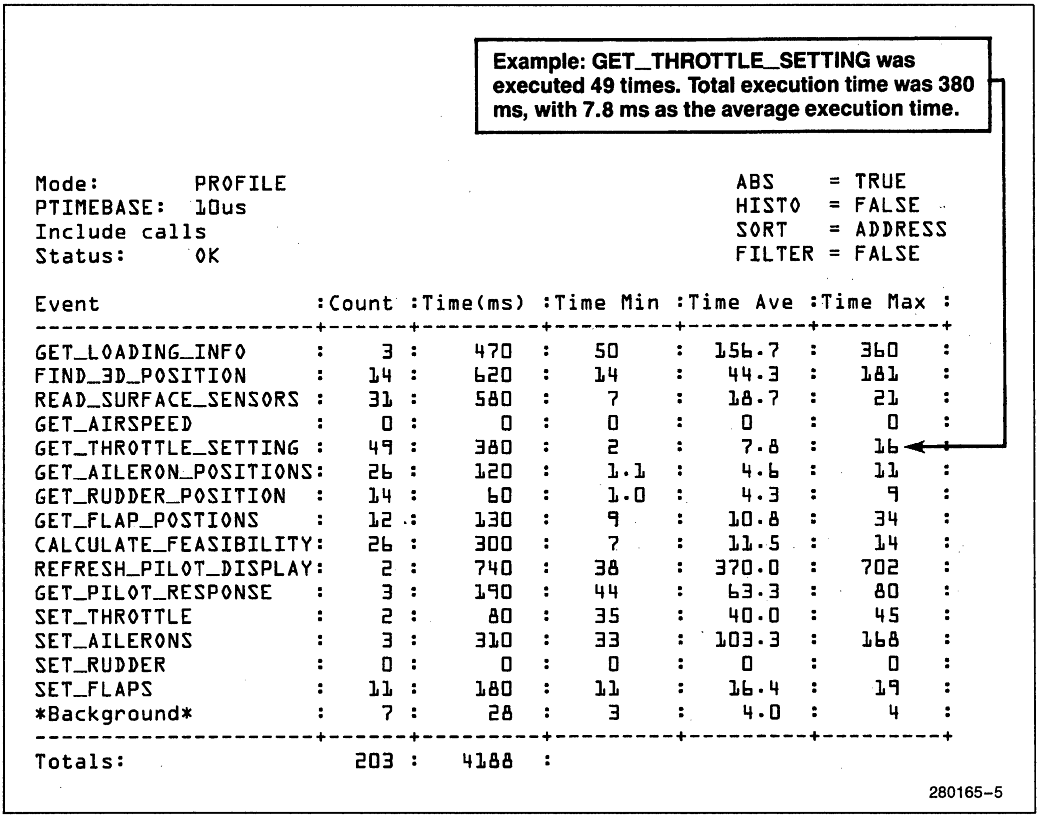
ed by the iPAT analyst as part of the calling procedure's time. If you do not want time charged to the caller, change the control so that the time accumulated by calling procedures excludes time used by called procedures. Then rerun the program and collect new data. Now, by comparing the time charged to the calling procedure in the two cases, you can determine to what extent calls by the procedure use excessive time.

When you use profile mode, you need not collect data on the whole program. You can restrict the range of modules, procedures, or lines that are profiled. In addition, you can restrict the profile to specified absolute-address ranges or to an interrupt-address pair.

Profile-Mode Displays: The default profile display (shown in Figure 2) provides a histogram of the time used by program procedures. Once you notice that some procedures are taking too long, you will want to determine how often those procedures are called. Is the excessive time a result of their being called frequently or the result of slow code? To find out, you need only select a display of count information. A histogram appears immediately (derived from already-acquired data). In the histogram, the lines for the procedures that are taking too long will show whether their counts are small (implying slow code) or large.

You can also display count and time information simultaneously by selecting the table display option. To do so, simply change the HISTO control to false and request a new display. Figure 4 shows a sample profile table display.

Another display control allows you to specify in what order data is presented. By default, data is presented in address order. But you can also direct the iPAT analyst to arrange results in time order or count order, with highest values first.



Example: GET_THROTTLE_SETTING was executed 49 times. Total execution time was 380 ms, with 7.8 ms as the average execution time.

```
Mode:        PROFILE                                  ABS    = TRUE
PTIMEBASE:   10us                                     HISTO  = FALSE
Include calls                                         SORT   = ADDRESS
Status:      OK                                       FILTER = FALSE

Event                    :Count :Time(ms) :Time Min :Time Ave :Time Max :
-------------------------+------+---------+---------+---------+---------+
GET_LOADING_INFO         :    3 :     470 :   50    :  156.7  :   360   :
FIND_3D_POSITION         :   14 :     620 :   14    :   44.3  :   181   :
READ_SURFACE_SENSORS     :   31 :     580 :    7    :   18.7  :    21   :
GET_AIRSPEED             :    0 :       0 :    0    :    0    :     0   :
GET_THROTTLE_SETTING     :   49 :     380 :    2    :    7.8  :    16   :
GET_AILERON_POSITIONS:        26 :     120 :    1.1  :    4.6  :    11   :
GET_RUDDER_POSITION      :   14 :      60 :    1.0  :    4.3  :     9   :
GET_FLAP_POSTIONS        :   12 :     130 :    9    :   10.8  :    34   :
CALCULATE_FEASIBILITY:        26 :     300 :    7    :   11.5  :    14   :
REFRESH_PILOT_DISPLAY:         2 :     740 :   38    :  370.0  :   702   :
GET_PILOT_RESPONSE       :    3 :     190 :   44    :   63.3  :    80   :
SET_THROTTLE             :    2 :      80 :   35    :   40.0  :    45   :
SET_AILERONS             :    3 :     310 :   33    :  103.3  :   168   :
SET_RUDDER               :    0 :       0 :    0    :    0    :     0   :
SET_FLAPS                :   11 :     180 :   11    :   16.4  :    19   :
*Background*             :    7 :      28 :    3    :    4.0  :     4   :
-------------------------+------+---------+---------+---------+---------+
Totals:                     203 :    4188 :
```

280165-5

Figure 4. Profile Mode: Table Display

Duration Mode

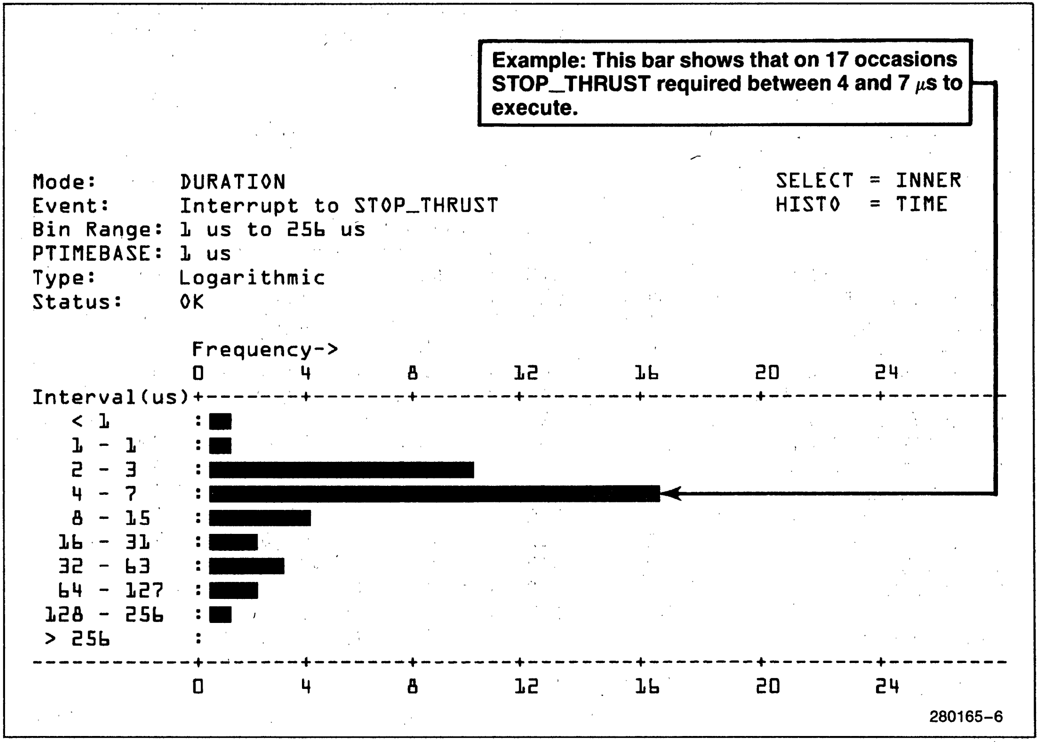
Duration-Mode Analysis: With the duration mode you can focus on timing information for one block of code or one interrupt-address pair. If you wish to determine how regularly a procedure meets performance specifications for timing, duration mode will provide the answer. This mode also is useful when you want information on how widely response time varies between the arrival of an interrupt and the execution of a particular service routine.

Duration mode collects data from repeated executions of a specified block of code or interrupt-address pair. The data is then placed in a number of bins (selectable as 8, 16, or 32 bins). You can select whether the bins have equal intervals or bin size increases logarithmically (use the latter when you expect a wide variation in time values).

Figure 5 shows a sample duration-mode default display. It assumes that a user wishes to find out the variation in response time for a specific interrupt-address sequence. In this case, the user is interested in the elapsed time between an interrupt caused by ground contact of an airplane's landing gear and the execution of the first statement in the procedure that controls thrust shutdown. The display shows, for example, that the bin for the elapsed time interval 4 μs to 7 μs recorded 17 instances of the interrupt-procedure execution pair. Note that in this case the performance specification indicated that elapsed time should never exceed 64 μs, the duration display shows that the current design does not meet the specification.

Duration-Mode Displays: The default duration display (as shown in Figure 5) provides a time histogram. A table display can also be selected.

In duration mode, you are not restricted to learning only about timing that occurs between two events. You can also learn about timing that occurs outside the event pair—the demand for the event pair. Suppose, for instance, RAM memory in your operating system is currently filled, and you want to determine whether one of the processes stored there is used too infrequently to justify its placement in RAM. Collect data on this process using the duration mode. Then use the duration-mode OUTER display option.



Example: This bar shows that on 17 occasions STOP_THRUST required between 4 and 7 μs to execute.

```
Mode:       DURATION                                   SELECT = INNER
Event:      Interrupt to STOP_THRUST                   HISTO  = TIME
Bin Range:  1 us to 256 us
PTIMEBASE:  1 us
Type:       Logarithmic
Status:     OK

             Frequency->
             0       4       8       12       16       20       24
Interval(us)+-------+-------+-------+--------+--------+--------+--------
   < 1      :
   1 - 1    :
   2 - 3    :
   4 - 7    :
   8 - 15   :
  16 - 31   :
  32 - 63   :
  64 - 127  :
 128 - 256  :
 > 256      :
------------+-------+-------+-------+--------+--------+--------+---------
             0       4       8       12       16       20       24
```

280165-6

Figure 5. Duration Mode: Histogram Display

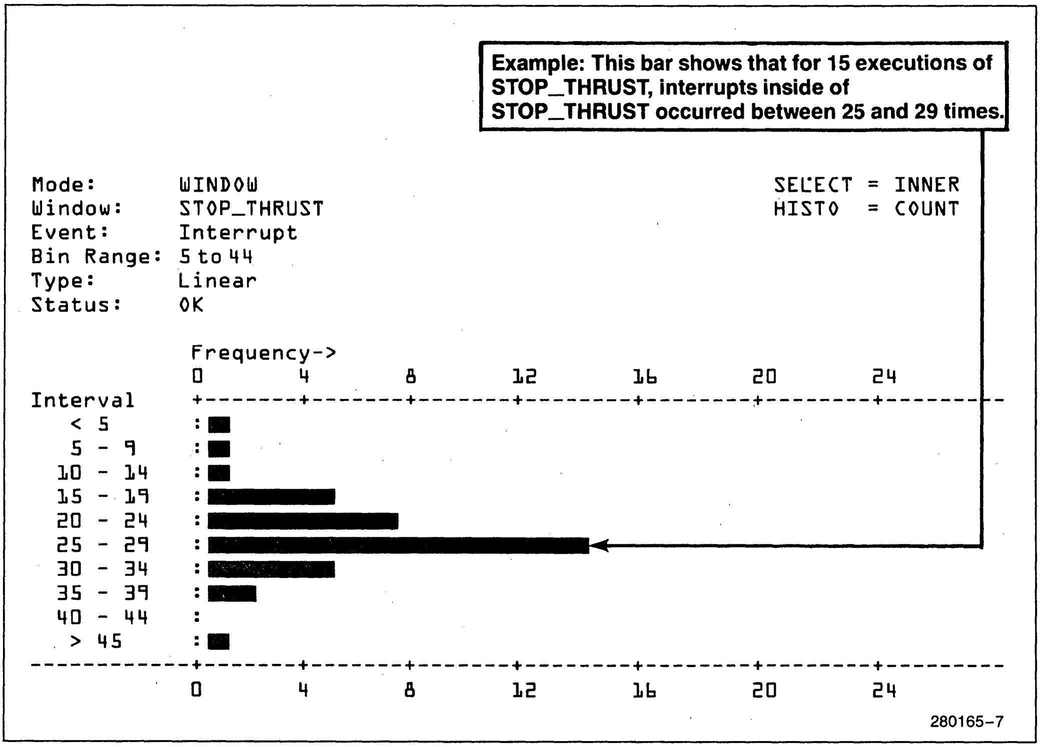
By doing so, you select a display of binned timing data that shows the distribution of the specified process's demand. If the process is infrequently used (contrary to original expectations), it could be moved to disk and RAM space made available for other, more frequently used, routines.

Windowed-Event-Count Mode

The windowed-event-count mode counts how often a specified begin-end pair (window) is entered—and how often, once the window is entered, an interrupt occurs or a specified address is executed. (A count is also kept of how often the selected event occurs outside the window.) As with the duration mode, data is binned. The begin-end pair can be two addresses (specified absolutely or symbolically) or an address and the occurrence of an interrupt.

This mode is useful for obtaining refined count data. For example, if profile mode indicates that procedure A is using excessive time and that much of the time is attributable to procedure calls, you can use this mode to get a better understanding of the situation. Use procedure A as the window and the name of a procedure it calls (B) as the event of interest. Data will then be gathered and placed in bins. The resulting display will show the distribution of how often procedure B is called each time procedure A is executed. Thus, you can see whether procedure B is the procedure causing procedure A to use so much time.

Because the event is counted both inside and outside the window, you can use this mode to determine whether an undesired event occurs excessively within a given block of code. If, for example, one procedure consumes too much time and you suspect that interrupts are occuring excessively during the procedure, use this mode to corroborate your suspicions. Specify the procedure as the window and interrupts as the event. Then display the results both for interrupts within the procedure and those outside the procedure. By comparing the two displays, you can determine whether interrupt frequency within the procedure is skewed. Figure 6 shows a sample display for interrupts that occur inside the window.



Example: This bar shows that for 15 executions of STOP_THRUST, interrupts inside of STOP_THRUST occurred between 25 and 29 times.

```
Mode:        WINDOW                                        SELECT = INNER
Window:      STOP_THRUST                                   HISTO  = COUNT
Event:       Interrupt
Bin Range:   5 to 44
Type:        Linear
Status:      OK

             Frequency->
             0       4       8       12       16       20       24
Interval     +-------+-------+-------+--------+--------+--------+--------
   < 5       :
   5 - 9     :
  10 - 14    :
  15 - 19    :
  20 - 24    :
  25 - 29    :
  30 - 34    :
  35 - 39    :
  40 - 44    :
   > 45      :
-------------+-------+-------+-------+--------+--------+--------+--------
             0       4       8       12       16       20       24
```

280165-7

Figure 6. Windowed-Event-Count Mode: Interrupt Latency Histogram

As with the duration mode, you can select the granularity of data collection for the windowed-event-count mode (8, 16, or 32 bins), and you can specify linear or logarithmic binning.

Linkage Mode

Linkage mode has two options, the many-to-one option and the many-to-many option. Both options allow you to focus on inter-procedure activity.

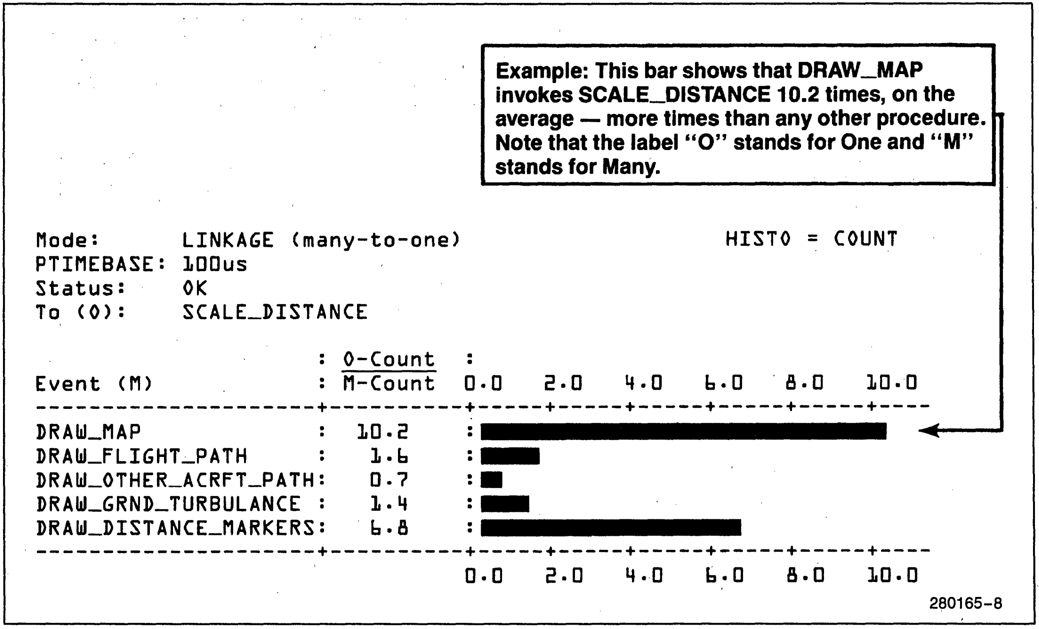
Many-to-One Option: With this linkage option, you can focus on one procedure or block of code (the one) and determine its linkage to other procedures or blocks of code (the many) that call it.

For example, suppose that profile mode has revealed procedure SCALE_DISTANCE to be using excessive time and is called often (see Figure 7). If many of the calls to it are from one or two procedures, to improve execution speed SCALE_DISTANCE could be optimized and moved in line into the procedures that call it often. The many-to-one option can help in this case. You simply enter the PAT LINKAGE analysis command and specify the names of the procedure that call SCALE_DISTANCE (the many) and specify SCALE_DISTANCE as the one. Then, when the program is executed, appropriate count and time data is collected. Figure 7 shows a sample count histogram display for the many-to-one option. For each of the calling procedures, Figure 7 shows the average number of invocations of SCALE_DISTANCE. We see that procedure DRAW_MAP, on the average, invokes SCALE_DISTANCE 10.2 times.

The many-to-one display can also be changed to a time histogram (showing, for each of the many procedures, the average time the one procedure uses) or to a table.

Many-to-Many Option: This linkage option allows you to collect information on the linkage between many event pairs.

In the other modes, you cannot use an interrupt or the same address to specify both members of an event pair. For the many-to-many option, there is no such restriction. Thus, with this option you can collect timing and count information on recursive procedures and interrupt-to-interrupt activity.



Example: This bar shows that DRAW_MAP invokes SCALE_DISTANCE 10.2 times, on the average — more times than any other procedure. Note that the label "O" stands for One and "M" stands for Many.

```
Mode:      LINKAGE (many-to-one)                   HISTO = COUNT
PTIMEBASE: 100us
Status:    OK
To (O):    SCALE_DISTANCE

                     : O-Count  :
Event (M)            : M-Count  0.0   2.0   4.0   6.0   8.0   10.0
---------------------+----------+-----+-----+-----+-----+-----+----
DRAW_MAP             :   10.2   :██████████████████████████████
DRAW_FLIGHT_PATH     :    1.6   :████
DRAW_OTHER_ACRFT_PATH:    0.7   :██
DRAW_GRND_TURBULANCE :    1.4   :████
DRAW_DISTANCE_MARKERS:    6.8   :███████████████████
---------------------+----------+-----+-----+-----+-----+-----+----
                                0.0   2.0   4.0   6.0   8.0   10.0
```

280165-8

Figure 7. Linkage Mode (Many-to-One): Count Ratio Histogram

Data for the many-to-many option is displayed in a table. See Figure 8 for a sample display.

USER INTERFACE

The iPAT software is integrated with the emulator software. For example, iPAT command options are integrated in the emulator syntax menu at the bottom of the screen.

In addition, the emulator LITERALLY command can be used to abbreviate frequently used commands. The history buffer is also available to retrieve previous commands.

As already noted, the iPAT analyst requires only one command line to set up an analysis-mode (PAT INIT) and one to request a data display (PAT DISPLAY). There are also six display pseudo-variables used to set display options: SHOW, ABS, SELECT, FILTER, SORT, and HISTO.

Users can save test-coverage data collected for subsequent reviewing. The command PAT SAVE saves coverage data to a user-specified file; the command RECALL enables you to restore the file and then update the test information with additional test runs.

Displays for all modes can be saved to a file using the emulator LIST command.

To speed command entry, you can create registers that save frequently used commands. Then use the names of the desired registers with your analysis and display commands.

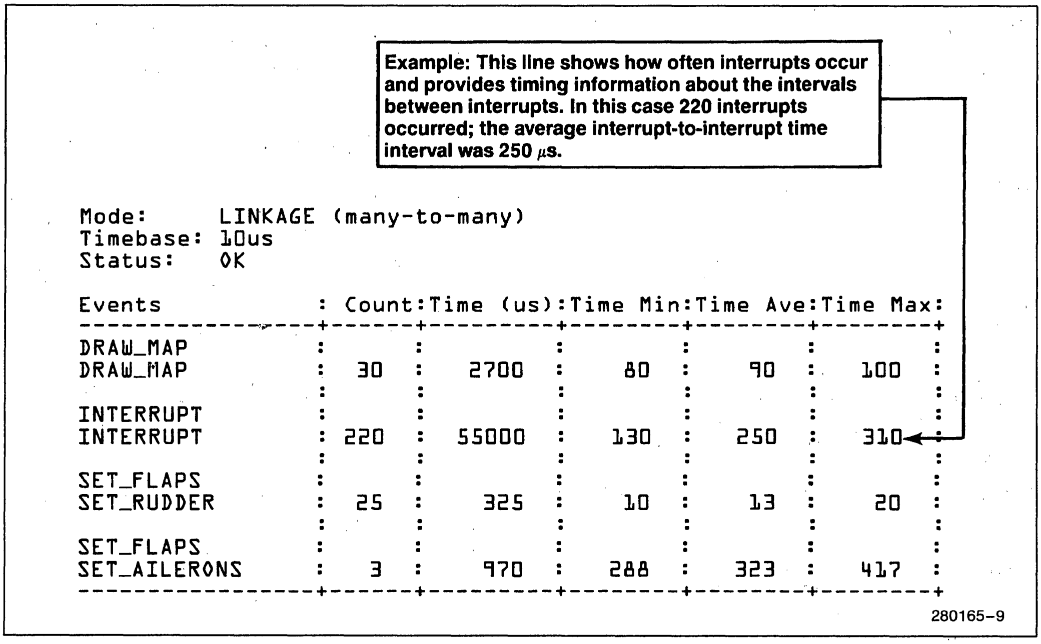
The emulator's screen editor can be used to examine and modify source code that the iPAT analyst has pinpointed as needing improvement.

SPECIFICATIONS FOR iPAT™ AN I²ICE™ SYSTEM

Host Requirements

Intel Series III or Series IV development system ; or an IBM PC XT or PC AT system

At least 512K bytes of RAM (of which 384K bytes must be available for the iPAT/I²ICE system software)



Example: This line shows how often interrupts occur and provides timing information about the intervals between interrupts. In this case 220 interrupts occurred; the average interrupt-to-interrupt time interval was 250 μs.

```
Mode:      LINKAGE (many-to-many)
Timebase:  10us
Status:    OK

Events              : Count:Time (us):Time Min:Time Ave:Time Max:
--------------------+------+---------+--------+--------+--------+
DRAW_MAP            :      :         :        :        :        :
DRAW_MAP            :  30  :   2700  :   80   :   90   :  100   :
                    :      :         :        :        :        :
INTERRUPT           :      :         :        :        :        :
INTERRUPT           : 220  :  55000  :  130   :  250   :  310 <-:
                    :      :         :        :        :        :
SET_FLAPS           :      :         :        :        :        :
SET_RUDDER          :  25  :    325  :   10   :   13   :   20   :
                    :      :         :        :        :        :
SET_FLAPS           :      :         :        :        :        :
SET_AILERONS        :   3  :    970  :  288   :  323   :  417   :
--------------------+------+---------+--------+--------+--------+
```

280165-9

Figure 8. Linkage Mode (Many-to-Many) Display

Available serial channel that operates at 300, 1200, 9600, or 19200 baud. (For a Series IV host, the available channel must be the IEU channel and, to use the iPAT analyst at baud rates greater than 300, an SPU board must be installed.)

Two double-density diskette drives or a hard disk

I²ICE™ System Requirements

Version 1.7 (or greater) probe software

iPAT software does not support I²ICE system operation with the Intel Logic Timing Analyzer (iLTA) and iLTA software does not support iPAT operation.

After the iPAT analyst interface board is installed, space is available in the I²ICE system instrumentation chassis for only one optional board. (Thus the user can install only one optional high-speed (OHS) memory board.)

Only one iPAT analyst will function in a multiple-probe I²ICE system.

iPAT™ Analyst Software

I²ICE host software that includes iPAT software

iPAT confidence tests

iPAT tutorial software

System Performance

Address Range Specification: Address ranges can be specified symbolically (for code compiled by Intel compilers) or with absolute addresses. Addresses anywhere within processor address space can be used.

Speed: The iPAT analyst captures instruction addresses at full processor speeds (however, when users specify many short intervals that are frequently executed, iPAT processing overflow may occur).

Timebase: Data collection timebase selectable as 200 ns, 1 μs, 10 μs, or 100 μs.

Display Updates: Users can specify how frequently (in seconds) displays are updated.

Status: If time-count, bin-count, or FIFO overflow occurs, the display indicates the overflow.

Profile Mode: Collects time and count information on specified entry-exit pairs. Permits specification of 125 entry-exit pairs when calls to other procedures are included in data collection and a minimum of 63 pairs when calls are excluded. Data collection can focus on modules, procedures, lines, absolute address pairs, or interrupt-address pairs. Displays are selectable as histograms or tables; data displayed can be sorted by address, count, or time.

Coverage Mode: Provides up to 252K bytes of coverage, mappable anywhere within the processor address space. Results are displayed in a table; users can select whether the table shows modules, procedures, or lines executed (and/or not executed).

Linkage Mode: The linkage mode has two options:

Many-to-One Option: Collects count and time data about interaction of one specified entry-exit pair with respect to other specified entry-exit pairs. Permits specification of 63 entry-exit address pairs for the many and one entry-exit address pair for the one. Displays are selectable as histograms or tables; data displayed can be sorted by address, count, or time.

Many-to-Many Option: Collects count and time data on one or more pairs of events. Permits specification of 63 event pairs; each member of a pair can be an address or interrupt. Measurements of recursion and interrupt to interrupt are supported. Display is a table.

Modes that Organize Data into Bins: The following two iPAT modes organize collected data into bins. Users can select bin granularity (8, 16, or 32 bins) and the highest and lowest values for the outer bins. Users can also select whether bin intervals are equal or increase logarithmically.

Windowed-Event Count Mode: Collects count data concerning an event that occurs within a specified window. Permits selection of the window entry-exit pair as an address pair, interrupt-address pair, or address-interrupt pair. The event selected can be an address or an interrupt. Resulting binned count data can be displayed as a histogram or table.

Duration Mode: Collects time information for a selected entry-exit pair. Permits selection of an entry-exit pair as an address pair, interrupt-address pair, or address-interrupt pair. Resulting binned timing information can be displayed as a histogram or table.

PHYSICAL CHARACTERISTICS

Target-Interface Board (to be installed in I²ICE system instrumentation chassis):	
Length Width	30 cm (12 in) 30 cm (12 in)
iPAT Core Module:	
Length Width Height	35 cm (13¾ in) 21 cm (8¼ in) 4 cm (1¾ in)
iPAT Power supply:	
Length Width Height	28 cm (11 in) 11 cm (4¼ in) 19 cm (7¾ in)

AC power cord for the power supply: 3.0 m (10 ft)

Power-supply-to-core DC power cable: 1.8 m (6 ft), 10 conductor

Emulation-clips jumper cable: 20 cm (8 in), 40 conductor

Execution-trace jumper cable: 10 cm (4 in), 60 conductor

iPAT-to-emulator cable: 0.9 m (36 in), 60 conductor

RS232 serial cable (for connecting the iPAT core to the host system): 3.7 m (12 ft). This cable is shipped with the iPAT software.

Electrical Characteristics

Selectable AC power source: 100V, 120V, 220V, 240V

47–63 Hz

2 amps (AC) at 100V or 120V, 1 amp at 220V or 240V

Environmental Requirements

Operating Temperature: 10°C to 40°C (50° to 104°F)

Operating Humidity: Maximum of 85%' relative humidity, non-condensing

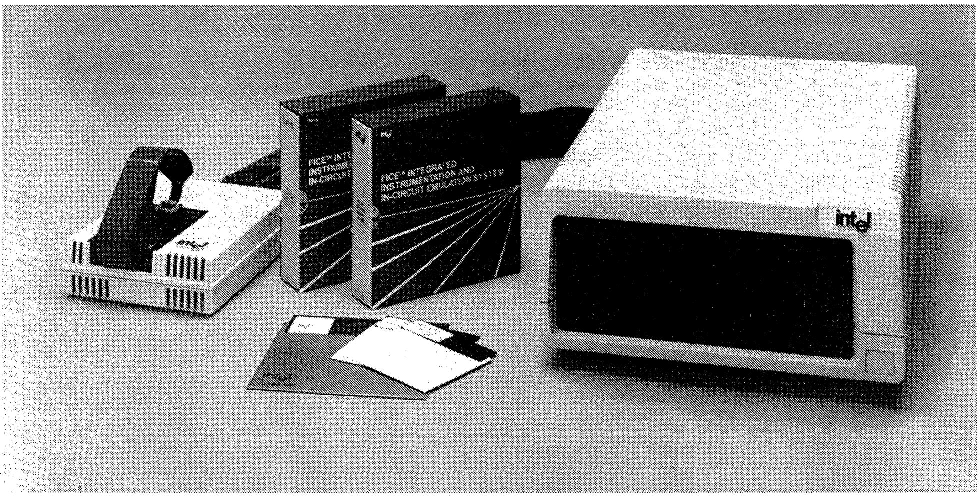
ORDERING INFORMATION

Order Code	Description
iPATCORE	iPAT core unit that supports Intel 8- and 16-bit microprocessors. It must be used with the appropriate emulator target interface, cables, and software.
iPAT86PC	iPAT-I²ICE system target interface, cables, and DOS software for IBM PC AT and PC XT host
iPAT86S3	iPAT-I²ICE system target interface, cables, and ISIS software for Series III host
iPAT86S4	iPAT-I²ICE system target interface, cables, and iNDX software for Series IV host
iPAT86DOS	iPAT DOS software (for use with IBM PC AT and PC XT hosts) and serial cables
iPAT86NDX	iPAT Series IV (iNDX) software and serial cable
iPAT86ISS	iPAT Series III (ISIS) software and serial cable

I²ICE™ Integrated Instrumentation and In-Circuit Emulation System

- Provides Real-Time In-Circuit Emulation
- Offers Symbolic Debugging Capabilities
  - Accesses Memory Locations and Program Variables (Including Dynamic Variables) Using Program-Defined Names
  - Maintains a Virtual Symbol Table
  - Source Code Display at Breakpoints
- Offers Multi-Condition, Multi-Level, Multi-Probe Break and Trace Capability
- Provides Built-In AEDIT Editor to Allow Editing of Development System Files without Exiting from I²ICE Operation
- Provides Low Cost Conversions Among 8086, 8088, 80186, 80188 and 80286 Microprocessors
- Simultaneously Controls up to Four Microprocessors for Debugging Multiprocessor Systems for a Single Work Station
- Supports Common Memory between Processors without Any User System Hardware
- Offers a Performance Analysis Tool (iPAT™ Analyst)
- Maps User Program Memory into a Maximum of 288K Zero-Wait-State RAM (Zero Wait-States to 10 MHz)
- Maps User I/O to Console or to Debugging Procedures
- Provides Disassembly and Single-Line Assembly to Help with On-Line Code Patching
- Common Human Interface Provided by the PSCOPE-86 Debugging Language and the I²ICE Command Language
- Uses Integrated Command Directory, ICD™, for Command Syntax Direction/Correction to Ease Debug Operations

The Intel Integrated Instrumentation and In-Circuit Emulation (I²ICE™) system aids the design of systems that use the 8086, 8088, 80186, 80188 and 80286 microprocessors. The I²ICE system combines symbolic software debugging, in-circuit emulation, and the optional Intel Performance Analysis Tool (iPAT analyst). Support features for the 8087 and 80287 coprocessors are also included. For the 8086/8088, 80186/80188, and 80286 processors, the I²ICE system supports programs written in C, PL/M, FORTRAN, Pascal, and assembly language. Up to four I²ICE instrumentation chassis can be hosted by one of Intel's Intellec® microcomputer development systems or by an IBM* PC AT or PC XT.



210469–1

*IBM is a trademark of the International Business Machines Corporation.

October 1987
Order Number: 210469-008

PHYSICAL DESCRIPTION

The I2ICE system hardware consists of the host interface board, the I2ICE instrumentation chassis, the emulation base module, the emulation personality module, a host/chassis cable, inter-chassis cables (for multiple chassis systems), a user cable, optional high-speed memory boards, and an optional performance analyzer. The I2ICE system software consists of I2ICE host software, I2ICE probe software, confidence tests, PSCOPE-86, and optional iPAT analyst software. Table 1 shows elements of the I2ICE system.

The host interface board resides in the host development system. A cable connects the host interface board to the I2ICE instrumentation chassis. Another cable connects the I2ICE instrumentation chassis to the buffer box.

The instrumentation chassis contains high-speed zero-wait-state emulation memory, break-and-trace logic, memory and I/O maps, and the emulation clips assembly.

The chassis may also contain the optional performance analyzer and optional high-speed memory. High-speed memory is expandable from 32K bytes to 288K bytes in 128K increments.

The buffer box contains the emulation personality module. This module configures the I2ICE system for a particular iAPX microprocessor. The user cable connects the buffer box to user prototype hardware.

The host development system may host up to four I2ICE instrumentation chassis. Each chassis may have its own buffer box, user cable, emulation clips, optional high-speed memory boards, and performance analyzer.

TARGET SYSTEM CONSIDERATIONS

To ensure proper emulation of a host target system, consider the following:

- Each I2ICE system probe has specific timing parameters that differ from the chip which the probe emulates. Hence, a customer design that follows the chip's timing specifications may not work with the I2ICE system probe. The target system may have to be modified slightly to account for the differences in timing between the probe and the chip. See the probe waveform section in this data sheet for timing differences.
- Target system noise and signal margins and timings are a critical consideration for emulation at speeds beyond 6 MHz. Typical solutions used to reduce target system noise such as RC networks and series resistor terminations could cause unacceptable timing degradation. Consequently, Intel recommends that wirewrap target boards be carefully designed for emulation with the I2ICE system. Printed circuit boards should be used because of the superior signal transmission characteristics. All target systems must have power and ground planes, decoupling capacitors, and signal lines layed out according to correct design techniques. For an introduction to proper design, see Application Note 125, Designing Microcontroller Systems for Electrically Noisy Environments, Order Number 210313.
- The I2ICE system depends on a target system clock signal to run the internal probe circuitry. To run the internal probe circuitry, the clock signal must satisfy two criteria. The target system clock must meet the voltage levels defined in this data sheet and it must also exceed the TTL logic family minimal noise and ringing specifications. This is necessary since the signal must travel up the user cable and through data buffers to reach the probe circuitry. The I2ICE system is designed to minimize the capacitive, noise, and chip delay associated with this path, but these effects worsen timings and amplify target system noise that may exist.

FUNCTIONAL DESCRIPTION

Resource Borrowing

The I2ICE system memory map allows the prototype system to borrow memory resources from the I2ICE system.

If prototype memory is not yet available, the user program may reside in I2ICE memory. Because this memory is RAM, changes can be made quickly and easily. For example, if the prototype contains EPROM, it does not need to be erased and reprogrammed during development.

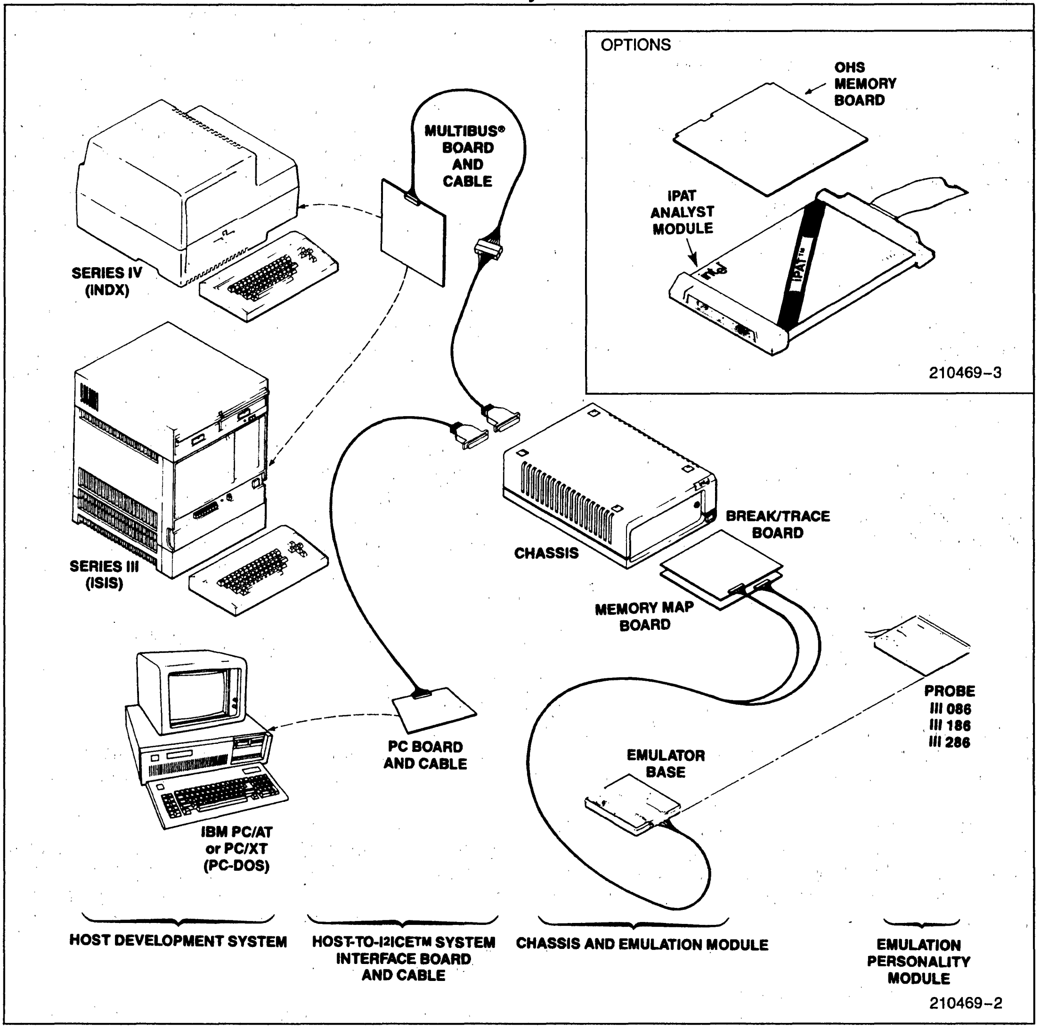
Later, as prototype memory becomes available, the verified user program can be reassigned, memory block by memory block, to prototype memory.

The I2ICE™ System Memory Map

The I2ICE system can direct (map) an emulated microprocessor's memory space (the user program memory) to any combination of the following:

- High-speed I2ICE system memory—this consists of 32K bytes of programamble wait-state memory (programmable from 0 to 15). This memory resides in the I2ICE system chassis on the map-I/O board.

Table 1. I²ICE™ System Overview



Name	Description
Host Development System	Required for all applications. Use one of the following: • Intellec Series III development system • Intellec Series IV development system • IBM PC AT or PC XT (with 512K bytes of available memory and version 3.0 of PC DOS) • IBM 50 system (available in Japan; features *kanji*)

Table 1. I²ICE™ System Overview (Continued)

Name	Description
Host-to-I²ICE System Interface Board, Cable, and Host Software	Required for communication between the host and the I²ICE system. • MULTIBUS® bus interface board for Series III and Series IV (product code III520) • Host-to-I²ICE system cable for the Series III and Series IV (product code III530 or III531) • I²ICE system host software for the Series III and Series IV (product code III951A, B, or C) • Package with PC host interface board, cable and PC DOS version of I²ICE host software (product code III520AT954D)
Instrumentation Chassis and Emulation Module	Required for real-time microprocessor emulation, break and trace capability, and memory and I/O capability. • Instrumentation chassis (product code III514B) has four board slots: 1 slot for break/trace board 1 slot for map-I/O board 2 slots for 1 (or 2) optional high-speed memory board(s) and/or 1 optional logic timing analyzer board • Maximum of four chassis for multi-probe applications • Emulation module (product code III620) includes break/trace board, map-I/O board, and buffer base box
Emulation Personality Module (Probe) and Probe Software	Required for emulation of specific microprocessors: 8086/8088, 80186/80188, or 80286. • Module includes personality board, buffer box cover, and user cable • Series III or IV: Order probe and probe software separately • PC host: Probe and probe software packaged together
Intel Performance Analysis Tool (iPAT Analyst)	Used to optimize code execution speed and control and to improve software reliability. • Complete with sytem software, power supply, core module, iPAT-to-I²ICE interface board, and cables
Optional High-Speed Memory Board (OHS)	Required for memory expansion. • 128K bytes of programmable (0 to 15) wait-state memory • One or two boards mount in the instrumentation chassis

- Optional high-speed I²ICE memory—this consists of up to 256K bytes of programmable wait-state memory (0 wait-states up to 10 MHz). This memory resides in the I²ICE system chassis on one or two optional high-speed memory boards (128K bytes each).
- MULTIBUS® bus memory (host system memory)—this resides in the host development system itself. (Any amount of unused host memory can be used in 1K increments.) Note that this feature is not available for a PC host.
- User memory—this resides in the user prototype hardware.

When a user program runs in I²ICE memory or user memory, the I²ICE system emulates in real time. A memory access to MULTIBUS bus memory, however, inserts approximately 25 wait-states into the memory cycle.

Access Restrictions

In addition to directing memory accesses, the following access restrictions can be specified.

- Read-only—the I²ICE system displays an error message if a user program attempts to write to an area of memory designated as read-only. The user can, however, write to a read-only area with I²ICE system commands.
- Read/write, no verify—normally, the I²ICE system performs a read-after-write verification after program loads and after writing to memory with an I²ICE system command. The I²ICE system can suppress this verification. For example, if a prototype has memory-mapped I/O, a verifying read may change the state of the I/O device.
- Guarded—initially, the I²ICE system puts all memory in a guarded state. Neither the user program nor the I²ICE system user can access guarded memory.

The I²ICE™ System I/O Map

The I²ICE system can direct (map) an emulated microprocessor's I/O space to the host development system's console, to the prototype system, to debugging procedures, or to a combination of these.

SIMULATING I/O WITH THE HOST DEVELOPMENT CONSOLE

Suppose a user program requires input from an I/O device not yet part of the prototype. Map the input port range assigned to that device to the host development system's console. Then, when the user program requires input, it halts and the I2ICE system console displays a message requesting the data. When you enter the required data at the keyboard, the user program continues.

SIMULATING I/O WITH I2ICE™ SYSTEM DEBUGGING PROCEDURES

Procedures that supply the needed input data can be written in the I2ICE system command language. When setting up the I/O map, the user specifies that the I/O procedure is invoked when certain I/O ports are accessed.

I/O ports are mapped in blocks of 64 byte-wide ports or 32 word-wide ports. A total of 64K byte-wide ports or 32K word-wide ports can be mapped.

Symbolic Debugging

With symbolic debugging, a memory location can be referenced by specifying its symbolic reference. A symbolic reference is a procedure name, line number, or label in the user program that corresponds to a location in the user program's memory space.

TYPICAL SYMBOLIC FUNCTIONS

Symbolic functions include:

- Changing or inspecting the value and type of a program variable by using its program-defined name, rather than the address of the memory location where the variable and a hexadecimal value for the data are stored.
- Defining break and trace events using source-code symbols.

With symbolic debugging, the user can reference static variables, dynamic (stack-resident) variables, based variables, and record structures combining primitive data types. The primitive data types are ADDRESS, BOOLEAN, BYTE, BCD, CHAR, WORD, DWORD, SELECTOR, POINTER, three INTEGER types, and four REAL types.

THE VIRTUAL SYMBOL TABLE

The I2ICE system maintains a virtual symbol table for program symbols; that is, the entire symbol table need not fit into memory at the same time. (The size of the virtual symbol table is constrained only by the capacity of the storage device.)

The I2ICE system divides the symbol table into pages. If a program's symbol table is large, the I2ICE system reads only some of the symbol table pages into memory. When the user references a variable whose symbol is not currently defined in memory, the I2ICE system reads the needed symbol table page from disk into memory.

Breakpoint, Trace, and Arm Specifications

With I2ICE commands, breakpoint, trace and arm specifications can be defined.

Breakpoints allow halting of a user program in order to examine the effect of the program's execution on the prototype. With the I2ICE system, a breakpoint can be set at a particular memory location or at a particular statement in a user program (including high-level language programs). A break can also be set to occur when the user program enters or accesses a specified memory partition or reads or writes a user program variable. When the user program resumes execution, it picks up from where it left off.

Normally, the I2ICE system traces while the user program executes. With a trace specification, however, the user can choose to have tracing occur only when specific conditions are met.

An arm specification describes an event or combination of events that must occur before the I2ICE system can recognize certain breakpoint and trace specifications. Typical events are the execution of an instruction or the modification of a data value.

The I2ICE system command language allows you to specify complex, multilevel events. For example, you can specify that a break occurs when a variable is written, but only if that write occurs within a certain procedure. The execution of the procedure is the arm condition; the variable modification is the break condition. The I2ICE system command language allows users to specify complex events with up to four states with four conditions and to use such events as arm, break, or trace conditions; a specified number of events can be used as a condition.

SOURCE DISPLAY

With the source display commands, a user can correlate a module under debug to a source code file. Then, when breakpoints are encountered, source text is displayed along with the break message and the line number of the breakpoint. The number of source code lines displayed before and after a breakpoint can also be defined.

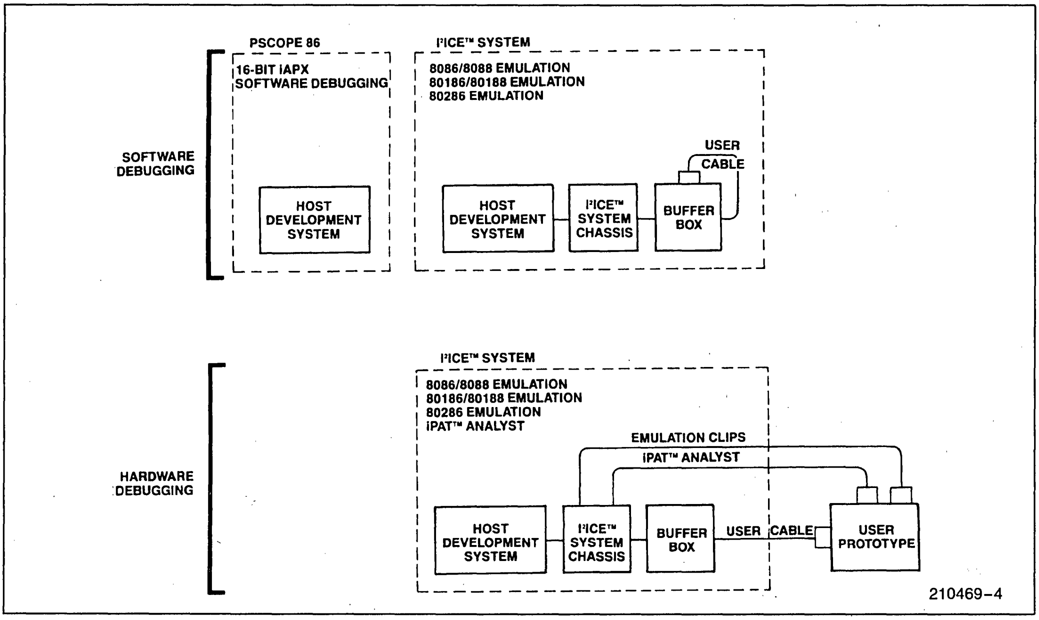


Figure 1. I2ICE™ System Debugging Capabilities

Coprocessor Support

The 8086/8088 emulation personality module provides transparent RQ/GT and MN/MX pin emulation to support real-time prototype systems that use the 8087 as a coprocessor. The 8086/8088 (and the 80186/80188) emulation personality module also provides debugging features specific to the 8087. I2ICE system commands provide access to the 8087's stack, status registers, and flags. The I2ICE system's disassembly and trace features extend to 8087 instructions and data types.

The 80186 and 80286 emulation personality modules also allow the prototype hardware to contain coprocessors. The 80186 probe can qualify break points and collect trace information when the coprocessor drives the status lines ($\overline{S0}-\overline{S2}$) in the prescribed manner. The 80286 personality module allows the hardware to contain the 80287 processor extension and provides special debugging features—the user can enable and disable the 80287 and change and examine its registers.

DEBUGGING WITH THE I2ICE™ SYSTEM

The I2ICE system allows both hardware and software debugging (see Figure 1).

- Software debugging—I2ICE system commands permit symbolic debugging of user programs written in high-level languages as well as assembly language. By looping the user cable back into the buffer box, a user program can be debugged even if no prototype hardware is present. In a multi-probe environment, the I2ICE system can map common memory from the host development system and support semaphore operation even with no user system prototype hardware. This feature makes possible detailed debugging of multi-processor software before the hardware is available.

  Additionally, as code is being developed, preliminary analyses can be made with the optional iPAT analyst. You can also use the I2ICE system and the iPAT analyst to analyze program execution under different target system conditions. This can be accomplished by setting up target system conditions in the I2ICE system and running the program with the iPAT analyst activated.

- Hardware debugging—the I^{2}ICE system is a real-time, in-circuit emulator. Trace data are collected in real time, and I^{2}ICE system software does not intrude into user program space.

The usefulness of an I^{2}ICE system extends throughout the development cycle, beginning with the symbolic debugging of prototype software and ending with the final integration of debugged software and prototype hardware.

PSCOPE-86

PSCOPE-86 is a high-level language, symbolic debugger, designed for use with Pascal-86, PL/M-86, and FORTRAN-86. It is a separate product included with the Series III and Series IV versions of the I^{2}ICE system; it runs in the host development system. PSCOPE-86 is field-proven, familiar to Intel customers, and suited for the debugging of applications software when the hardware capabilities of the I^{2}ICE system are not needed. The PSCOPE-86 and I^{2}ICE command languages are similar. (Note that PSCOPE-86 is available as an option for use with the PC AT or PC XT.)

Designing a product that contains a microcomputer requires close coordination of hardware and software development. A typical design process takes advantage of both the I^{2}ICE system and PSCOPE-86. Use PSCOPE-86 for debugging software before downloading the software into a target environment; use the I^{2}ICE system for debugging and emulation of the target system.

THE I^{2}ICE™ SYSTEM COMMAND LANGUAGE

The syntax of I^{2}ICE system commands resembles that of a high-level language. The I^{2}ICE system command langauge is versatile and powerful while remaining easy to learn and use.

The Integrated Command Directory (ICD™) assists users with command syntax.

- The ICD directory directs the user in choosing commands from a display on the bottom line of the screen. As commands are entered, the bottom line indicates syntax elements available for use in the commands.
- The ICD directory flags syntax errors. Syntax errors are flagged as they occur (rather than after the carriage return is pressed).
- The ICD directory provides on-line help with the HELP command.

Automatic expansion of LITERALLY expressions is available. When the feature is activated, each character string defined by a LITERALLY definition is automatically expanded to its full length.

The I^{2}ICE command language deals with user-created debugging objects. By manipulating debugging objects, the user can streamline complex debugging sessions.

Debugging objects are uniquely named, user-created, software constructs that the I^{2}ICE system uses to manage the debugging environment. The four types of debugging objects are: debugging procedures, LITERALLY definitions, debugging registers, and debugging variables. In the following examples, I^{2}ICE system keywords are shown in all caps.

- Debugging procedures (named groups of I^{2}ICE system commands) can simulate missing software or hardware, collect debugging information, and make troubleshooting decisions. For example, consider a debugging procedure (called **init**) that simulates input from I/O ports 2 and 4.

  The procedure and MAPIO command are given first, followed by an explanation.

```
*DEFINE PROCEDURE init = DO
.*IF %0==2 THEN
..*PORTDATA=100T
..*ELSE IF %0==4 THEN
...*PORTDATA=65T
...*END
..*END
.*END
*MAPIO 0 LENGTH 64K ICE (init)
```

  Whenever the MAPIO command maps I/O ports to an I^{2}ICE system procedure, three parameters are made available to the procedure (even if the procedure does not use them): %0, %1, %2. The parameter %0 passes the port number; %1 passes a Boolean value that indicates whether read or write I/O activity will occur, and %2 passes a Boolean value that indicates whether the I/O is a byte-wide or a word-wide port. PORTDATA is a pseudo-variable that contains the actual port data. This procedure specifies that if port 2 is used, the procedure returns 100 (base ten); if, however, port 4 is used, the procedure returns 65 (base ten).

- LITERALLY definitions are shorthand names for previously defined character strings. LITERALLY definitions can save keystrokes and improve clarity. For example, here is the definition of a LITERALLY that saves keystrokes. This LITERALLY allows the user to type DEF for DEFINE.

```
*DEFINE LITERALLY DEF = "DEFINE"
```

These definitions may be saved to disk and autoreloaded. In addition, an automatic LITERALLY expansion feature can be turned on and off.

- Debugging registers are user-created, software registers that hold arm, breakpoint and trace specifications. The I²ICE system can be ordered to emulate the user program and specify one or more debugging registers. There is no need to reenter the specification for each emulation. For example here is the definition of a debugging register called **pay** that contains a trace specification. This example takes advantage of the previous LITERALLY definition.

```
*DEF TRCREG pay = :cmaker.payment
```

To emulate a user program and trace only during the procedure **payment,** specify the debugging register **pay** as part of the GO command.

```
*GO USING pay
```

- Debugging variables are user-created variables used with I²ICE system commands. For example, here is the definition of a debugging variable called **begin.** Its type is POINTER.

```
*DEFINE POINTER begin = 0020H:0006H
```

During a debugging session, the user can set the execution point to this pointer value by typing:

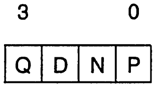
```
*$=begin
```

The I²ICE system pseudo-variable $ represents the current execution point.

Example of a Debugging Session

Figures 2, 3, and 4 illustrate some of the key capabilities of the I²ICE system. The user program is written in Pascal-86. It was compiled, linked, and located on an Intellec Series III development system. The resulting file consists of absolute code and is called CMAKER.86. Figure 2 shows the Pascal listing; Figure 3 shows a sample debugging session; and Figure 4 briefly explains the debugging steps shown in Figure 3.

The CMAKER.86 program controls an automatic changemaker. The program reads the amount tendered (the variable **paid**) and the amount of the purchase (the variable **purchase**). It calculates the coins needed for change and asserts control signals to a change release mechanism by writing an output port. Each of the lower four bits of the output port controls the release of a different coin denomination.

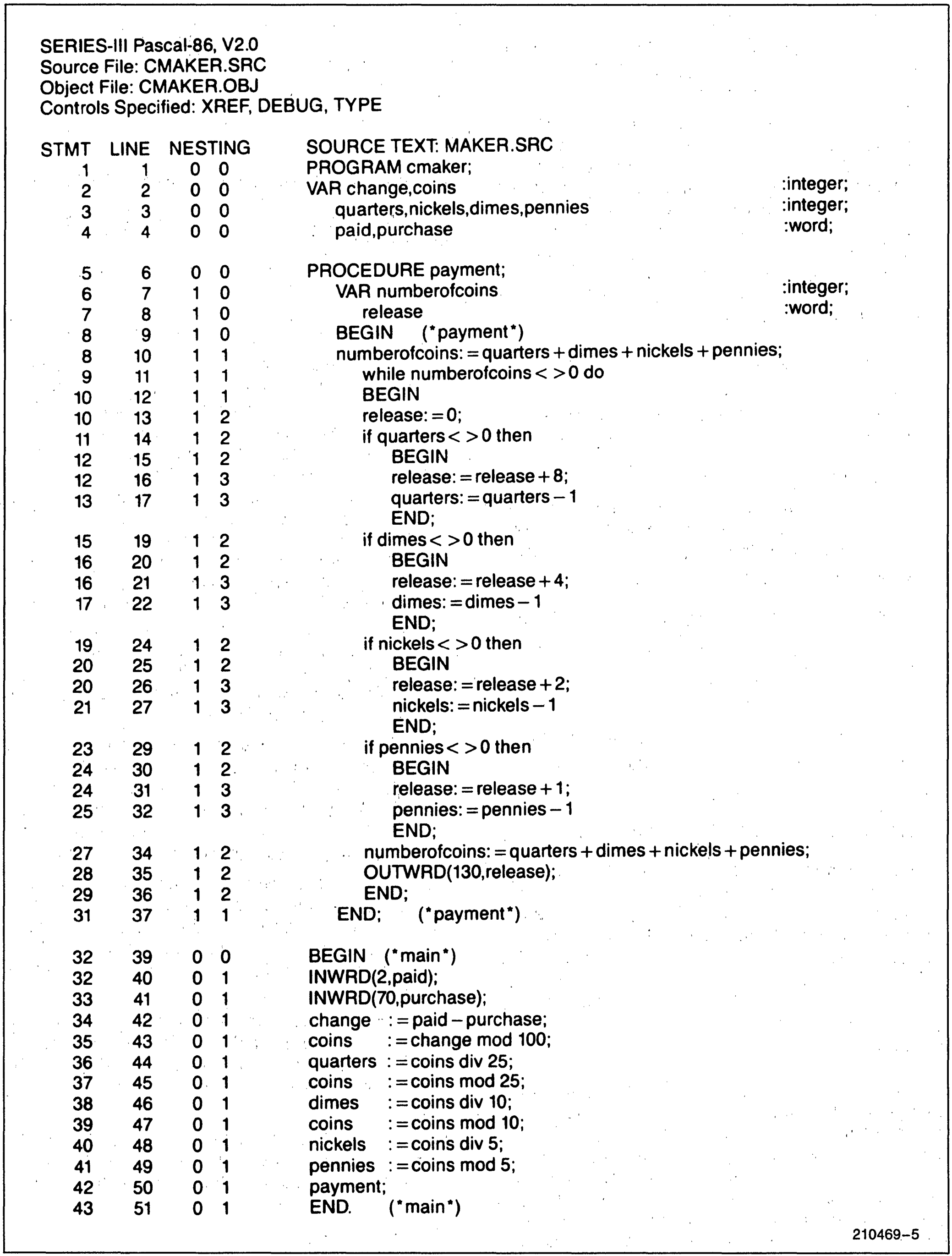


Q = quarters
D = dimes
N = nickels
P = pennies

I²ICE™ System Command Functions

The I²ICE system command language contains a number of functional categories.

- Emulation commands—the GO command instructs the I²ICE system to begin emulation. The user can also command the I²ICE system to break or trace under certain specified conditions.
- Utility commands—these are general purpose commands for use in a debugging environment. For example, one use of the EVAL command is to calculate the nearest source-code line number that corresponds to the address of an assembly language instruction. The PRESRC command can be used to display a specified number of source code lines preceding a breakpoint. The HELP command provides on-line assistance. The EDIT command invokes a menu-driven text editor (AEDIT) that allows updating of debugging object definitions and editing of development system files without exiting the I²ICE system. The shell escape command () enables access to the DOS operating system without exiting the I²ICE system (DOS host specific). A command line editor and history key are also provided.
- Environment commands—these are commands that set up the debugging environment. For example, the MAP command sets up the memory map. Another environment command (WAITSTATE) inserts wait-states into memory accesses, allowing the simulation of slow memories.
- File handling commands—these are commands that access disk files. Debugging object definitions can be saved in a disk file and loaded in later debugging sessions. Debugging sessions can also be recorded in a disk file for later analysis.
- Probe-specific commands—these are commands whose effects are different for different probes. For example, the PINS command displays the state of selected signal lines on the current probe.
- Option-specific commands—these are commands that control an optional test/measurement device, such as the performance analysis tool.



```
SERIES-III Pascal-86, V2.0
Source File: CMAKER.SRC
Object File: CMAKER.OBJ
Controls Specified: XREF, DEBUG, TYPE

STMT  LINE  NESTING   SOURCE TEXT: MAKER.SRC
   1     1    0 0     PROGRAM cmaker;
   2     2    0 0     VAR change,coins                                    :integer;
   3     3    0 0         quarters,nickels,dimes,pennies                  :integer;
   4     4    0 0         paid,purchase                                   :word;

   5     6    0 0     PROCEDURE payment;
   6     7    1 0         VAR numberofcoins                               :integer;
   7     8    1 0             release                                     :word;
   8     9    1 0         BEGIN    (*payment*)
   8    10    1 1         numberofcoins: = quarters + dimes + nickels + pennies;
   9    11    1 1             while numberofcoins < >0 do
  10    12    1 1             BEGIN
  10    13    1 2             release: = 0;
  11    14    1 2             if quarters < >0 then
  12    15    1 2                 BEGIN
  12    16    1 3                 release: = release + 8;
  13    17    1 3                 quarters: = quarters - 1
                                  END;
  15    19    1 2             if dimes < >0 then
  16    20    1 2                 BEGIN
  16    21    1 3                 release: = release + 4;
  17    22    1 3                 dimes: = dimes - 1
                                  END;
  19    24    1 2             if nickels < >0 then
  20    25    1 2                 BEGIN
  20    26    1 3                 release: = release + 2;
  21    27    1 3                 nickels: = nickels - 1
                                  END;
  23    29    1 2             if pennies < >0 then
  24    30    1 2                 BEGIN
  24    31    1 3                 release: = release + 1;
  25    32    1 3                 pennies: = pennies - 1
                                  END;
  27    34    1 2             numberofcoins: = quarters + dimes + nickels + pennies;
  28    35    1 2             OUTWRD(130,release);
  29    36    1 2             END;
  31    37    1 1         END;     (*payment*)

  32    39    0 0     BEGIN  (*main*)
  32    40    0 1     INWRD(2,paid);
  33    41    0 1     INWRD(70,purchase);
  34    42    0 1     change   : = paid - purchase;
  35    43    0 1     coins    : = change mod 100;
  36    44    0 1     quarters : = coins div 25;
  37    45    0 1     coins    : = coins mod 25;
  38    46    0 1     dimes    : = coins div 10;
  39    47    0 1     coins    : = coins mod 10;
  40    48    0 1     nickels  : = coins div 5;
  41    49    0 1     pennies  : = coins mod 5;
  42    50    0 1     payment;
  43    51    0 1     END.     (*main*)
```

210469-5

Figure 2. Listing of CMAKER.86

```
(1)  *BASE
     DECIMAL
(2)  *MAP 0K LENGTH 32K HS
     *MAPIO 0T LENGTH 192T ICE
     *MAP
     MAP          0K LENGTH     32K HS
     MAP         32K LENGTH    992K GUARDED
     *MAPIO
     MAPIO 00000H LENGTH       000C0H ICE
     MAPIO 000C0H LENGTH       0FF40H USER
(3)  *LOAD :F1:CMAKER.86
(4)  *DEFINE POINTER begin = $
     *DEFINE BRKREG pay = :cmaker #9
     *DEFINE PROC display = DO
     .*WRITE USING (‘ “quarters = ”,T,0,>’)quarters
     .*WRITE USING (‘ “dimes = ”,T,0’)dimes
     .*WRITE USING (‘ “nickels = ”,T,0,>’)nickels
     .*WRITE USING (‘ “pennies = ”,T,0’)pennies
     .*RETURN TRUE
     .*END
(5)  *GO USING pay
     ?UNIT 0 PORT 2H REQUESTS WORD INPUT (ENTER VALUE)*100
     ?UNIT 0 PORT 46H REQUESTS WORD INPUT (ENTER VALUE)*65
     *Probe 0 stopped at :CMAKER#9 + 4 because of execute break
       Break register is PAY Trace Buffer Overflow
(6)  *quarters;dimes;numberofcoins
     +1
     +1
     +2
(7)  *DEFINE SYSREG wr_number = WRITE AT .:cmaker.payment.numberofcoins &
     **CALL display
     *GO USING wr_number
       quarters =+1     dimes = +1
       nickels = +0     pennies = +0
     Probe 0 stopped at :CMAKER #28 + 3 because of bus break
       Break register is WR_NUMBER
(8)  *numberofcoins
     +0
     *EVAL release
     1100Y 12T CH'..'
(9)  *CLIPSOUT = 11Y
(10) *GO FOREVER
     ?UNIT 0 PORT 82H OUTPUT WORD 0C
     ?Probe 0 stopped at location 0033:00AEH because of bus not active
       Bus address = 0203DE
     *$=begin
     *
```

Figure 3. Sample Debugging Session (Explanations in Figure 4)

1. Checking to see that the default radix is decimal.
2. Mapping user program memory to I²ICE high-speed memory and user I/O ports to the I²ICE system console.
3. Loading the user program.
4. Defining debugging objects.

   The debugging variable **begin** is set to $, an I²ICE pseudo-variable representing the current execution point. At this point in the debugging session, $ is the beginning of the user program.

   The break register **pay** specifies a breakpoint at statement 9 in the user program.

   The debuggning procedure **display** displays the value of some user program variables on the console.
5. Beginning emulation with the debugging register **pay.** The console requests the two input values, **paid** and **purchase.** Then, the break occurs.
6. Displaying three user program variables.
7. Defining another debugging register. The specified event is the writing of the user program variable **numberofcoins.** When that event occurs, the I²ICE system calls the debugging procedure **display.** In addition to displaying some user program variables, this debugging procedure returns a Boolean value. Because this value is TRUE, the break occurs; if the value were FALSE, emulation would continue.
8. Displaying the two user program variables, **numberofcoins** and **release.** The EVAL command displays **release** in binary, decimal, hexadecimal, and ASCII. Unprintable ASCII characters appear as periods (.).
9. Asserting both output lines on the emulation clips. These lines are input to the prototype hardware and control a change release mechanism.
10. Resuming emulation. The console displays the write of **release** to the output port. The user program finishes executing, and the probe stops emulating because of bus inactivity. The $ is set back to the beginning of the user program in preparation for another emulation.

Figure 4. Explanation of Sample Debugging Session in Figure 3

I²ICE™ SYSTEM INSTRUMENTATION SUPPORT

I²ICE™ System Emulation Clips

Eight external input lines are sampled during each processor bus cycle. The I²ICE system records the values of these lines in the trace buffer during each execution cycle. The I²ICE system can use these values when defining events.

Four additional output lines synchronize I²ICE system events with external hardware. Two lines are active and programmable with I²ICE system commands. Two other lines, break and trace, allow an I²ICE system chassis to be linked to other I²ICE system chassis.

iPAT™ PERFORMANCE ANALYSIS TOOL

The Intel Performance Analysis Tool (iPAT analyst) helps software engineers optimize code and improve software reliability. Software object code generated by Intel assemblers and Intel compilers (e.g., for C, PL/M, Pascal, and FORTRAN) can be analyzed symbolically to improve software execution efficiency and to validate test coverage. Any object code that lacks Intel compiler information—but that can be run by Intel emulators and for which an absolute program map is available—can also be analyzed (non-symbolically) by the iPAT analyst. The iPAT analyst operation is currently supported via a target interface to the I²ICE system. For more information, see the iPAT analyst data sheet, Order Number 280165.

I²ICE™ SYSTEM SPECIFICATIONS

Host Requirements

Series III, Series IV, Model 800, or IBM PC AT or PC XT.

512K bytes in host development system memory space.

Two double-density diskette drives or a hard disk.

I²ICE™ System Software

I²ICE system host software
I²ICE system probe software
I²ICE system confidence tests
PSCOPE-86 (available as an option for the IBM PC AT or PC XT)

System Performance

Mappable zero wait-state memory (zero wait-states up to 10 MHz for 8086; 8 MHz for 8088; up to 10 MHz for 80186/80188, and up to 10 MHz for 80286):	Minimum 32K bytes Maximum 288K bytes
Trace buffer:	1023 x 48 bits
Virtual symbol table:	The number of user program symbols is limited only by available disk space.

Physical Characteristics

INSTRUMENTATION CHASSIS

Width:	17.0 in (43.2 cm)
Height:	8.25 in (21.0 cm)
Depth:	24.13 in (61.3 cm)
Weight:	48 lbs (21.9 kg)

HOST/CHASSIS CABLE

10 ft (3.0m) and 42 ft (12.8m) options for Series III/Series IV host
15 ft (4.6m) for PC host

INTER-CHASSIS CABLE SET

2 ft (61 cm) and 10 ft (3.0m) options

BUFFER BOX

Width:	8.5 in (21.6 cm)
Height:	3.0 in (7.6 cm)
Depth:	10.0 in (25.4 cm)
Weight:	8 lbs (3.7 kg)

Electrical Characteristics

90–132V or 180–264V (selectable)
47–63 Hz
12 amps (AC)

Environmental Requirements

Operating Temperature:	0°C to 40°C (32°F to 104°F)
Operating Humidity:	Maximum of 85% relative humidity, non-condensing

PRELIMINARY

ICE™-186 IN-CIRCUIT EMULATOR

HIGH PERFORMANCE REAL-TIME EMULATION

Intel's ICE-186 emulator delivers real-time emulation for the 80C186 microprocessor at speeds up to 12.5 MHz. The in-circuit emulator is a versatile and efficient tool for developing, debugging and testing products designed with the Intel 80C186 microprocessor. The ICE-186 emulator provides real time, full speed emulation in a users system. Popular features such as symbolic debug, 2K bytes trace memory, and single-step program execution are standard on the ICE-186 emulator. Intel provides a complete development environment using assembler (ASM86) as well as high-level languages such as Intel's C86, PL/M86 or Fortran 86 to accelerate development schedules.

The ICE-186 emulator supports a subset of the 80C186 features at 12.5 MHz and at the TTL level characteristics of the component. The emulator is hosted on IBM's Personal Computer AT, already available as a standard development solution in most of today's engineering environments. The ICE-186 emulator operates in prototype or standalone mode, allowing software development and debug before a prototype system is available. The ICE-186 emulator is ideally suited for developing real-time applications such as industrial automation, computer peripherals, communications, office automation, or other applications requiring the full power of the 12.5 MHz 80C186 microprocessor.

ICE™-186 FEATURES

- Full 12.5 MHz Emulation Speed
- 2K Bytes Deep Trace Memory
- Two-Level Breakpoints with Occurrence Counters
- Single-Step Capability
- 128K Bytes Zero Wait-State Mapped Memory
- Supports DRAM Refresh
- High-Level Language Support
- Symbolic Debug
- Coprocessor Support
- RS-232-C and GPIB Communication Links
- Crystal Power Accessory
- Interface for Intel Performance Analysis Tool (iPAT)
- Interface for Optional General Purpose Logic Analyzer
- Tutorial Software
- Complete Intel Service and Support

intel®

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.

AUGUST, 1987
Order Number: 280726-001

HIGHEST EMULATION SPEED AVAILABLE TODAY

The ICE-186 emulator supports development and debug of time-critical hardware and software using Intel's 12.5 MHz 80C186 microprocessor.

RETRACE SOFTWARE TRACKS

This emulator captures up to 2,048 frames of processor activity, including both execution and data bus activity. With this trace memory, large blocks of program code can be traced in real time and viewed for program flow and behavior characteristics.

HARDWARE BREAKPOINTS FOR COMPLEX DEBUG

User-defined "TIL-THEN" breakpoint statements stop emulation at specific execution addresses or bus events. During the hardware and software integration phase, breakpoint statements can be defined as execution addresses and/or bus addresses and/or bus access types such as memory and I/O reads or writes. Additionally, event counters provide another level of breakpoint control for sophisticated state machine constructs used to specify emulation breakpoints/tracepoints.

SMALL OR LARGE STEPS

A stepping command can be used to view program execution one frame at a time or in preset frame blocks. When used in conjunction with symbolic debug, code execution can be monitored quickly and precisely.

DEBUG CODE WITHOUT A PROTOTYPE

Even before prototype hardware is available, the ICE-186 emulator working in conjunction with the Crystal Power Accessory (CPA) creates a "virtual" application environment. 128K bytes of zero wait-state memory is available for mapped memory and I/O resource addressing in 4K increments. The CPA provides emulator diagnostics as well as the ability to use the emulator without a prototype.

DON'T LOSE MEMORY

The ICE-186 emulator continues DRAM refresh signals even when emulation has been halted, thus ensuring DRAM memory will not be lost. During interrogation mode the ICE-186 emulator will keep the timers functioning and correctly respond to interrupts in real-time.

HIGH LEVEL LANGUAGE SUPPORT OPTIMIZED FOR INTEL TOOLS

The ICE-186 supports emulation for programs written in Intel's ASM86 or any of Intel's high-level languages:

PL/M-86	Fortran-86
Pascal-86	C-86

These languages are optimized for Intel component architectures to deliver a tightly integrated, high performance development environment.

USER-FRIENDLY SYMBOLICS AID IN DEBUG

Symbolics allow access to program symbols by name rather than cumbersome physical addresses. Symbolic debug speeds the debugging process by reducing reliance on memory maps. In a dynamic development process, user variables can be used as parameters for ICE-186 commands resulting in a consistent debug environment.

COPROCESSOR SUPPORT

Coprocessor support enables applications to run faster due to off loading of the main CPU. The ICE-186 emulator supports alternate coprocessors such as LAN controllers and graphic engines, however it does not have built in support for the 8087 coprocessor.

MULTIPLE HIGH-SPEED COMMUNICATION LINKS

Two communication links are available for use in conjunction with the host IBM PC AT. The ICE-186 emulator uses either serial (RS-232-C) or a parallel (GPIB) link. A user supplied National Instruments (IEEE-488) GPIB communication board provides parallel transfers at rates up to 300K bytes per second.

SOFTWARE ANALYSIS (iPAT)

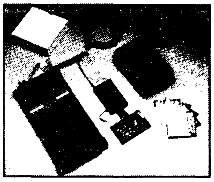
Intel's Performance Analysis Tool (iPAT) is designed to increase team productivity with features like interrupt latency measurement, code coverage analysis and software module performance analysis. These features enable the user to design reliable, high performance embedded control products. The ICE-186 emulator has an external 60 pin connector for iPAT.

BUILT-IN SUPPORT FOR LOGIC ANALYSIS

General-purpose logic analyzers can be used in conjunction with the ICE-186 to provide detailed timing of specific events. The ICE-186 emulator provides an external sync signal for triggering logic analysis, making complex trigger sequence programming easy. An additional 60 pin connector is included for the logic analyzer.

WORLDWIDE SERVICE AND SUPPORT

The ICE-186 emulator is supported by Intel's worldwide service and support organization. Total hardware and software support is available including a hotline number when the need is there.



SPECIFICATIONS

PERSONAL COMPUTER REQUIREMENTS

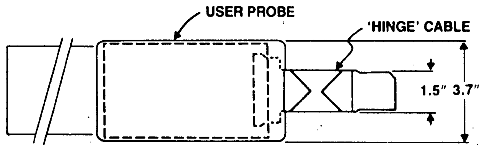
The ICE-186 emulator is hosted on an IBM PC AT. The emulator has been tested and evaluated on an IBM PC AT. The PC AT must meet the following minimum requirements:

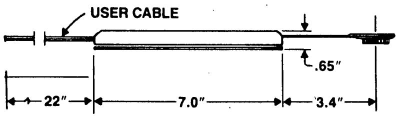
- 640K Bytes of Memory
- Intel Above Board with at Least 1M Byte of Expansion Memory
- One 360K Bytes or One 1.2M Bytes floppy Disk Drive
- One 20M Bytes Fixed-Disk Drive
- PC DOS 3.2 or Later
- A serial Port (COM1 or COM2) Supporting Minimally at 9600 Baud Data Transfers, or a National Instruments GPIB-PC2A board.
- IBM PC AT BIOS

PHYSICAL DESCRIPTION AND CHARACTERISTICS

The ICE-186 Emulator consists of the following components:

	Width		Height		Length	
Unit	**Inches**	**Cm.**	**Inches**	**Cm.**	**Inches**	**Cm.**
Emulator Control Unit	10.40	26.40	1.70	4.30	20.70	52.60
Power Supply	2.80	7.10	4.15	10.70	11.00	27.90
User Probe	3.70	9.40	.65	1.60	7.00	17.80
User Cable/ Plcc					22.00	55.90
Hinge Cable					3.40	8.60
Crystal Power Accessory	4.30	10.90	.60	1.50	6.70	17.00
CPA Power Cable					9.00	22.90





ELECTRICAL CONSIDERATIONS

Icc 1050mA

ENVIRONMENTAL SPECIFICATIONS

Operating Temperature 10°C-40°C Ambient
Storage Temperature −40°C-70°C

ORDERING INFORMATION

ICE186	ICE-186 NMOS System including ICE software (Requires DOS 3.XX PC AT with Above Board)
ICE186PAT	ICE-186 NMOS System including ICE S/W packages and the iPAT system (Requires DOS 3.XX PC AT with Above Board)
D86ASM86NL	86 macro assembler 86 builder/binder/ mapper utilities for DOS 3.XX.
D86C86NL	86 C compiler and run time libraries for DOS 3.XX.
D86PLM86NL	86 PL/M compiler for DOS 3.XX.
D86FOR86NL	86 Fortran compiler for DOS 3.XX.

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26

FRANCE
Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin-en-Yvelines Cedex

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Seidlstrasse 27
D-8000 Muenchen 2

HONG KONG
Intel Semiconductor Ltd.
1701-3 Connaught Centre
1 Connaught Road

CANADA
Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8

Printed in U.S.A./0188/74K/RRD/SM
Embedded Controller

ISBN 1-55512-072-5