HITACHI

HD68450 DMAC APPLICATION NOTES

#U60

HITACHI HD68450 DMAC APPLICATION NOTES #U60

HD68450 DMAC (Direct Memory Access Controller) APPLICATION NOTE

HITACHI

#U60

H680DMAC-EAN

HD68450 DMAC

The HD68450 DMAC is a 16-bit microprocessor that is bus-compatible with HMCS68000 systems, and has the following features:

- 4 independent DMA channels (programmable priority order)
- Maximum Transfer Rate is 4M Bytes/sec (8MHz)
- Various Multi-Data-Block Transfer Modes: Continue Mode, Array Chaining Mode, and Linked Array Chaining Mode
- High Reliability of Data Transfer facilitated by Error Detect, Error Interrupt Vector, and Exception features.
- 16M-Byte Address Space (same as the HD68000)
- Memory-to-I/O Device Transfer, Memory-to-Memory Transfer
- Programmable Operation Mode and Transfer Mode
- External Transfer Request, Internal Transfer Request (Auto-Request)
- Programmable System Bus Bandwidth Utilization

The HD68450 is also applicable in other processor systems (the 8086 system).

CONTENTS

1. HD68450 DMAC Operation

1.1 HD68450 Operating State

The HD68450 has internal control registers and performs required operations through control words written into the registers by the MPU. The DMAC state is divided into three modes:

1) MPU Mode: A bus master (MPU, DMAC) chip-selects the DMAC, or the MPU acknowledges the DMAC's interrupt request, reading or writing the contents of the DMAC's internal registers.
2) DMA Mode: The DMAC owns bus mastership, and is transferring data or preparing for data transfer.
3) IDLE Mode: The DMAC is waiting for a transfer request or MPU access, and most of the bus control signals are three-stated.

In normal operation, the DMAC transfers operands in the following sequence:

(1) The initiation phase, in which the MPU sets up control registers, transfer address, and transfer counts. The DMAC is enabled to accept transfer request.
(2) The transfer phase; the DMAC receives requests, transfers data, and writes the transfer status into the error register and internal status register after completion of the transfer.
(3) The termination phase; the MPU checks the post-transfer status.

The MPU determines the operation types and checks the transfer state by writing and reading the contents of the internal registers.

In addition to normal operations, bus exceptions are also prepared (see Chapter 1.5 Exceptions).

1.2 Transfer Types

1.2.1 Classification of the transfer modes in terms of request generation methods.

Transfer modes which the DMAC supports are shown in Table 1.1.

The External Request is generated by asserting the $\overline{REQ}$ pin (transfer request pin), and has two modes: Cycle Steal Mode which is edge-sense, and Burst Mode which is level-sense. Auto-Request is generated internally and the transfer starts by the DMAC itself. This is suitable where an external device has no transfer request mechanism (e.g., memory-to-memory transfer), or where an external device can not determine the timing to make a transfer request.

If the request generation method of "Auto-Request + External Request" is used, the DMAC transfers the 1st operand by the Auto-Request when a certain internal condition is satisfied. The $\overline{REQ}$ signal outputted can then inform an external device of the start of transfer. The 2nd and succeeding operands can be transferred with External Request.

1.2.2 Block Transfer Classification

The DMAC supports data block transfers by request generation methods shown in Table 1.1.

In Continue Mode, the DMAC transfers a pair of blocks without software intervention. It can transfer multi blocks by giving the next block information (address and word count) to the DMAC internal registers, and setting CNT bit again during the transfer of the second block.

In Array Chaining Mode, the MPU prepares for the array table (transfer address and word count listed in main memory). The DMAC transfers multi data blocks up to "2^{16} = 64K" according to the order in the array.

Linked Array Chaining Mode is almost the same as Array Chaining Mode, except the block information in the array need not be listed in the transfer order sequentially. Instead, linked address (block information which is going to be transferred next) is given as a part of the block information.

Examples of array tables are shown in Figure 1.2. The Linked Array Chaining Mode is more flexible in composing an array table, to change the order of transfer, or to skip blocks in the transfer order. For example, when block #2 is skipped in Array Chaining Mode, block #2 address and word counts must be replaced by block #3 information in an Array Table, and the former block #3 must be replaced by block #4, etc.

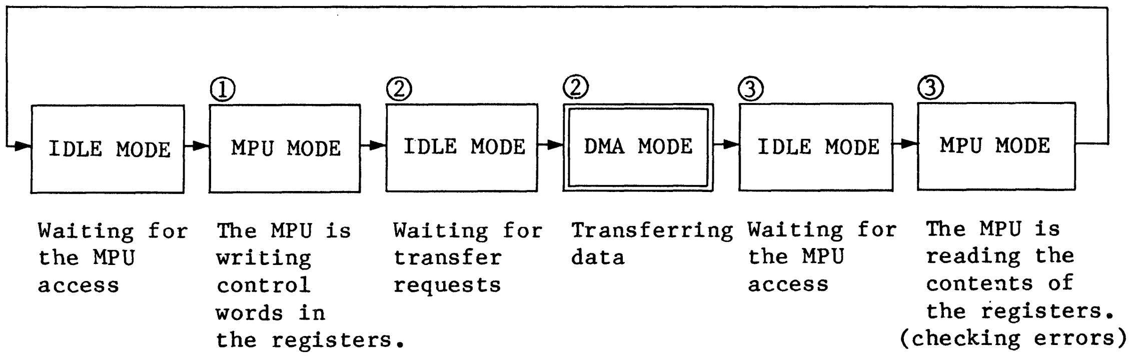
Linked Array Chaining Mode provides an easy method of changing only "block #2 information address" in block #1 information to "block #3 information address." When one block transfer has been completed, the DMAC automatically reads the next transfer block information to the internal registers. Array Chaining has 3 word read cycles, whereas Linked Array Chaining has 5 word read cycles (larger overhead).

In Continue Mode, fewer clock cycles are required to transfer information between the DMAC internal registers. The MPU, however, must write the next block information in those DMAC internal registers when 3 or more blocks are transferred.

Selection of a suitable mode for multi block transfers should consider such factors as time, I/O device speed, and program developing effort. Table 1.3 shows overhead clock cycles for each mode:

1.2.3 Transfer Classification by I/O Device Type

The DMAC can select a transfer mode as follows: For devices which are chip-selected with $\overline{ACK}$ signal, Single Addressing Mode is used, and an operand is transferred in one bus cycle



①
②
②
③
③
IDLE MODE
MPU MODE
IDLE MODE
DMA MODE
IDLE MODE
MPU MODE
Waiting for the MPU access
The MPU is writing control words in the registers.
Waiting for transfer requests
Transferring data
Waiting for the MPU access
The MPU is reading the contents of the registers. (checking errors)

FIGURE 1.1 HD68450 Operation State

TABLE 1.1 Classification of Transfer Modes in Terms of Request Generation

Request Method	External Request (using an $\overline{\text{REQ}}$ pin)	Cycle Steal Mode The DMAC transfers a single operand and relinquishes the bus after each transfer.
		Cycle Steal with Hold Mode The DMAC transfers a single operand and relinquishes the bus after each transfer, but holds the bus for a specified period of time after completion of the single operand transfer.
		Burst Mode The DMAC transfers plural operands continuously.
	Auto-Request (*not* using an REQ pin)	Limited Rate The DMAC transfers plural operands continuously, but relinquishes the bus during the operation.
		Maximum Rate The DMAC transfers plural operands continuously, but does not relinquish the bus until the end of the operation.
	Auto-Request (only the first transfer) + External Request (transfers after the second)	

TABLE 1.2 Block Transfer Classification

Single data block transfer		The MPU gives transfer address and transfer counts to the DMAC internal registers.
Multi-data block transfer	Continue Mode	The MPU gives transfer address and transfer counts to the DMAC internal registers, and sets CNT bit to inform the DMAC of the existence of the next data block.
	Array Chaining Mode	The MPU installs an array table in main memory.
	Linked Array Chaining Mode	The MPU installs a linked array table in main memory.

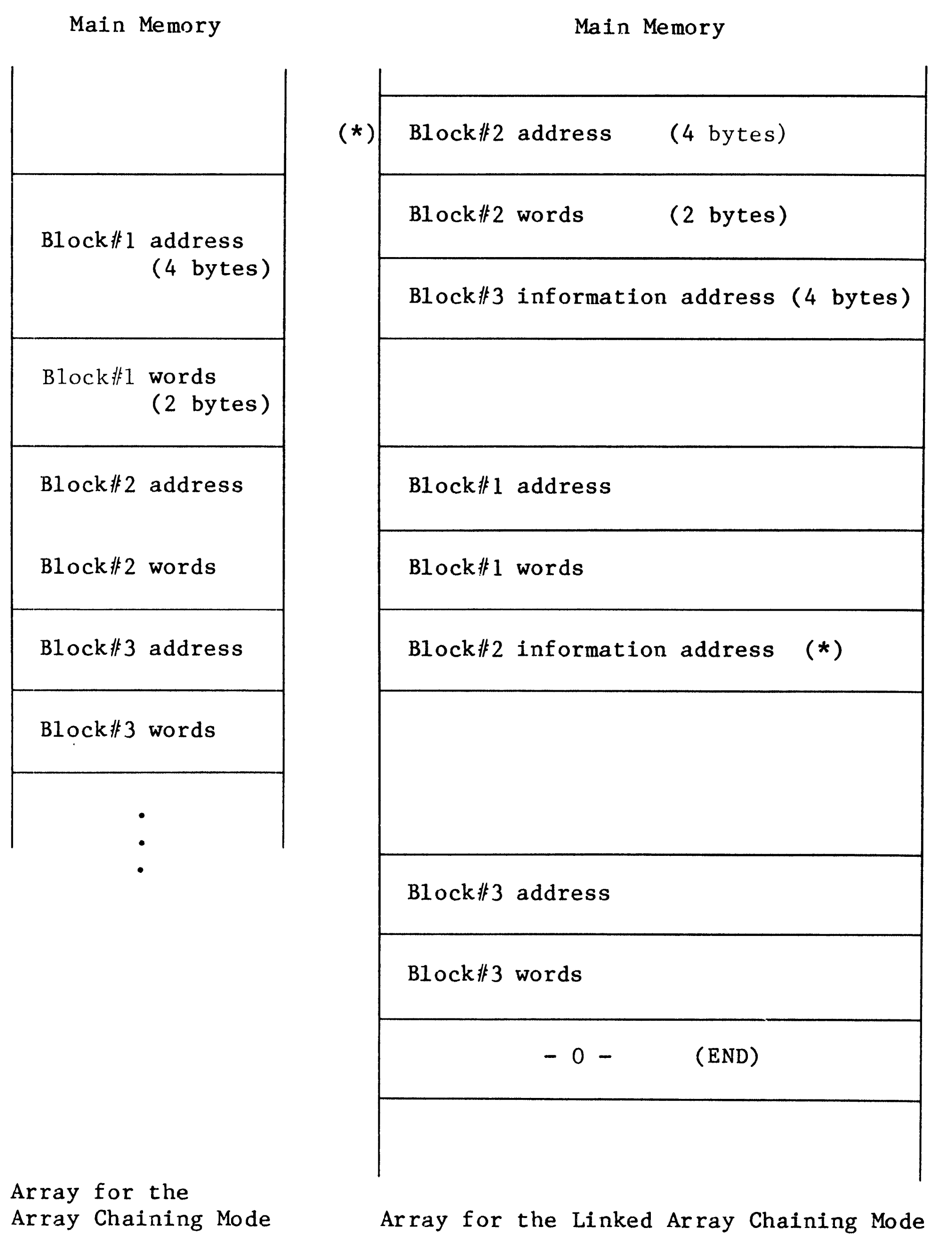


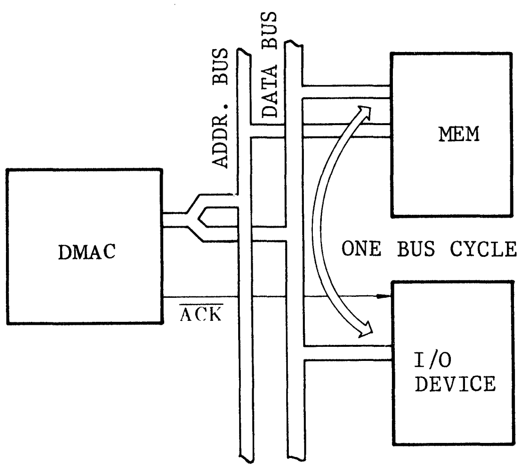
FIGURE 1.2 Example of Chaining Mode Array Tables

TABLE 1.3 Overhead Required for Loading Block Information

Transfer Mode	Overhead Clock Cycles	Note
1. Continue Mode	24 clock cycles	Overhead for loading the 2nd block information
2. Array Chaining Mode	38 clock cycles	Read Cycle : 4 clock cycles
3. Linked Array Chaining Mode	50 clock cycles	(NO wait state)

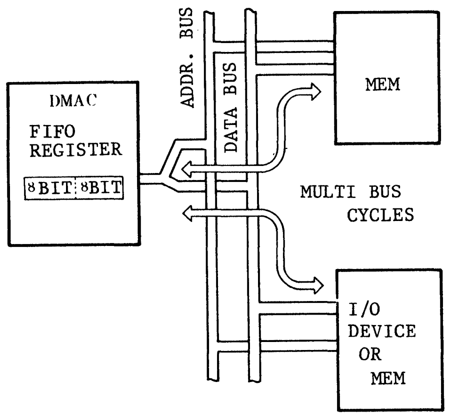
TABLE 1.4 Classification of Transfers by I/O Device Types

- Single Addressing
  - The device chip-selected by $\overline{\text{ACK}}$.
  - The device chip-selected by $\overline{\text{ACK}}$ and outputs $\overline{\text{READY}}$ to transfer.
- Dual Addressing
  - The device chip-selected by decoding address lines (68000-type).
  - The device chip-selected by decoding address lines; synchronous transfer (6800-type).



OPERAND SIZE = I/O PORT SIZE
(BYTE OR WORD)

Figure 1.3 Single Addressing Mode



```
OPERAND SIZE = or ≠ I/O PORT SIZE
EX) I/O : 8 BIT PORT,  DEVICE→MEM TRANSFER
     ⎧1ST CYCLE        I/O→FIFO 8 BIT READ
     ⎨2ND CYCLE        I/O→FIFO 8 BIT READ
     ⎩3RD CYCLE        FIFO→MEM 16 BIT WRITE
```

Figure 1.4 Dual Addressing Mode

(Figure 1.3). In this mode, the DMAC outputs memory address and $\overline{\text{ACK}}$ signal in the same bus cycle, informing the I/O device of the transfer start, and transfers data between memory and the device.

Futhermore, when the I/O device has $\overline{\text{READY}}$ signal to inform the DMAC of the completion of a transfer, the DMAC finishes the bus cycle, confirming the $\overline{\text{READY}}$ signal. When the I/O device is chip-selected by decoding address lines (68000 bus compatible device), the DMAC requires bus cycles for addressing to memory and the I/O device respectively. This transfer mode is called Dual Addressing, in which the DMAC uses the internal FIFO register (First In First Out), which temporarily keeps the operand inputted from the memory or device source, and transfers it to the destination in the following bus cycles (see Figure 1.4). The $\overline{\text{ACK}}$ signal is usually outputted when the DMAC addresses the I/O device, and not outputted when it addresses memory. For 68000-type devices, and when the request is Auto-Request, ACK signal is not outputted. For Single Addressing, the port size of the I/O device and operand size must be the same, whereas in Dual Addressing, they need not be the same because of the FIFO register. The relative data is shown in Table 1.5.

Users can independently designate each mode described in sections 1.2.1 through 1.2.3. For example, users can transfer multi data blocks (1) in Continue Mode, (2) with request generation of Cycle Steal with Hold, and (3) by means of single Addressing. These operation modes are designated by writing control words into the DMAC internal registers.

1.3 Internal Registers

The DMAC internal registers shown in Figs. 1.5 and 1.6 can be addressed with address lines A1-A7, $\overline{\text{LDS}}$, and $\overline{\text{UDS}}$.

DCR is a register to designate an external I/O device. It designates the external request generation method, device type and port size, and $\overline{\text{PCL}}$, line operation (described further on).

OCR designates the transfer operation. It designates the data transfer direction, operand size, chain operation types, and request generation method.

SCR designates the increment/decrement sequence of both memory and device (source and destination) addresses.

CCR designates the channel operation. It designates the operation start, the continuous operation presence, HALT, abort, and interrupt enable/disable.

CSR has the channel status. It shows the channel operation completion, block transfer completion, normal termination, error status, channel active state, and $\overline{\text{PCL}}$ signal line information.

CER indicates occurrence of error types.

CPR determines the priority of the channel.

MTC is a 16-bit register to hold transfer counts. The block size (transfer counts) is written when one data block is transferred. When multi blocks are transferred in Continue Mode and Chaining Mode, the next block size is automatically loaded in *MTC* after completion of the previous block transfer.

BTC is used in Continue Mode and Array Chaining Mode. In Continue Mode, the first block size is stored in MTC after completion of the first block transfer. When more than two blocks are transferred in this mode, BTC and BAR (described further on) are rewritten, and CNT bit in CCR is set again during the second or third block transfer. In Array Chaining Mode, BTC holds the number of blocks being transferred.

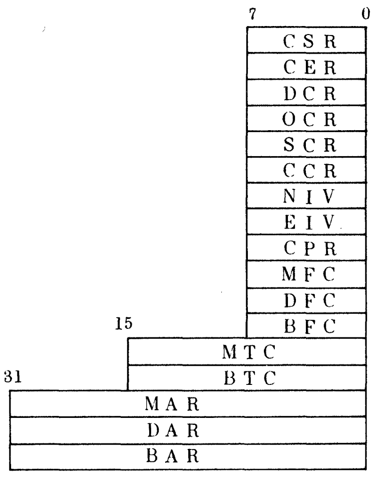
MAR contains the memory address being outputted at each transfer cycle. In block transfer, the beginning address of the block is written in MAR as an initial value. The content of MAR varies according to the contents of OCR and the SIZE bits (operand size) in SCR after one operand transfer. In Continue Mode and Chain Modes, MAR is rewritten according to BAR or the array information in memory when a block transfer completes.

DAR is used to address an I/O device (or to address memory, in memory-to-memory transfer). DAR is used only in Dual Addressing Mode, and changes its content according to SCR and SIZE bits in OCR.

BAR is used in Continue Mode and Chain Modes. In Continue Mode, the start address of the 2nd block is written in

TABLE 1.5 Possible Choice of Port Size & Operand Size

Transfer Mode	Device Port Size	Operand size			Transfer Request
	bit	8	16	32	
Dual Addressing	8	OK	OK	OK	External Request Auto-Request
	16	OK	OK	OK	Auto-Request
	16	NG	OK	OK	External Request Auto-Req+External Req.
Single Addressing	8	OK	NG	NG	External Request Auto-Request
	16	NG	OK	NG	



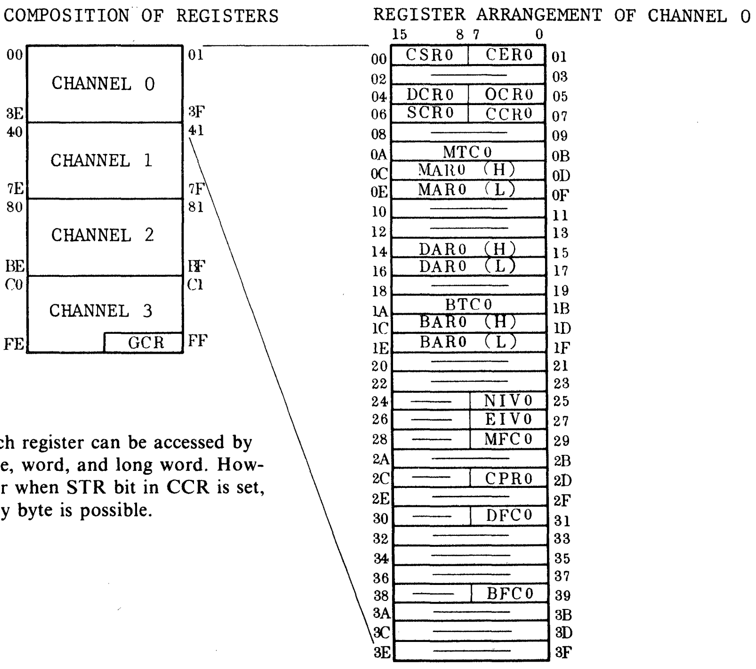
Register	7	6	5	4	3	2	1	0	Mode	
	Address Bits									
Channel Status Register	c	c	0	0	0	0	0	0	R/W*	one set per channel
Channel Error Register	c	c	0	0	0	0	0	1	R	
Device Control Register	c	c	0	0	0	1	0	0	R/W	
Operation Control Register	c	c	0	0	0	1	0	1	R/W	
Sequence Control Register	c	c	0	0	0	1	1	0	R/W	
Channel Control Register	c	c	0	0	0	1	1	1	R/W	
Normal Interrupt Vector	c	c	1	0	0	1	0	1	R/W	
Error Interrupt Vector	c	c	1	0	0	1	1	1	R/W	
Channel Priority Register	c	c	1	0	1	1	0	1	R/W	
Memory Function Codes	c	c	1	0	1	0	0	1	R/W	
Device Function Codes	c	c	1	1	0	0	0	1	R/W	
Base Function Codes	c	c	1	1	1	0	0	1	R/W	
Memory Transfer Counter	c	c	0	0	1	0	1	b	R/W	
Base Transfer Counter	c	c	0	1	1	0	1	b	R/W	
Memory Address Register	c	c	0	0	1	1	s	s	R/W	
Device Address Register	c	c	0	1	0	1	s	s	R/W	
Base Address Register	c	c	0	1	1	1	s	s	R/W	

cc:00-Channel #0,01-Channel #1
10-Channel #2,11-Channel #3

ss:00-high-order,01-upper middle
10-lower middle,11-low-order
b :0-high-order, 1-low-order
* :"Write" is valid only for resetting the register.

General Control Register	1	1	1	1	1	1	1	1	R/W	one per DMAC

FIGURE 1.5 Internal Registers and Address Assignment



NOTE: Each register can be accessed by byte, word, and long word. However when STR bit in CCR is set, only byte is possible.

FIGURE 1.6 Whole Arrangement of Registers

BAR. This BAR is used in the same way as BTC. In Chain Modes, it keeps the address where the information of the next block is contained.

MFC, DFC and BFC are used with MAR, DAR, and BAR, respectively. The MFC,, DFC, and BFC are used with the same purpose as the FC outputted from the MPU.

Since the FC registers in the DMAC can be written, the DMAC can also transfer data between the supervisor program area (FC = 110) and the user program area (FC = 010).

NOTE: Each register can be accessed by byte, word, and long word. However, when STR bit in CCR is set, only byte is possible.

NIV and EIV keep the vector numbers outputted in the vector number fetch cycle (Interrupt Acknowledge Cycle), which the MPU performs for the interrupt requested by the DMAC. If no error (ERR bit of CSR is not set) occurs, the DMAC outputs NIV contents. When error occurs (ERR = 1), the DMAC outputs EIV contents. In both cases, the DMAC does not output the vector address containing software routine for the interrupt process, and instead outputs the necessary data for the vector address calculation. Therefore, the contents of NIV and EIV are outputted onto the lower data bus (D_0 - D_7). This scheme is equivalent to HMCS68000 bus protocol.

GCR is common to all four channels and determines the DMAC's bus use ratio and sample interval in Limited Rate Auto-Request Mode. During transfer operation in this mode, the DMAC supervises the bus bandwidth by dividing the transfer time into the equal time interval called "sample interval." This sample interval consists of $2^{(BT+BR+5)}$ clock cycles. BT and BR have 2 bits respectively in GCR and a sample interval can be 32 to 2048 clock cycles. The DMAC performs the DMA cycles during the first $2^{(BT+4)}$ clock cycles in the sample interval, and relinquishes the bus in the latter part (see Figure 1.7).

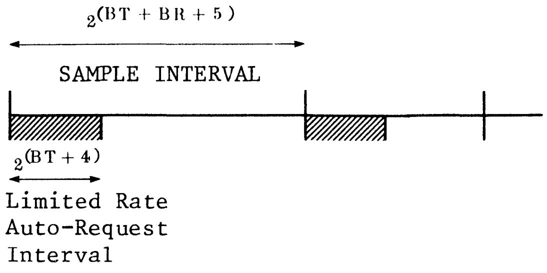


FIGURE 1.7 SAMPLE INTERVAL in Limited Rate Auto-Request

The DMAC monitors $\overline{\text{BGACK}}$ signal (described later). When $\overline{\text{BGACK}}$ is asserted, the DMAC starts counting the clock cycle. The DMAC compares the count with $2^{(BT+4)}$. When the $2^{(BT+4)}$ clock cycle is in the middle of a bus cycle, the DMAC continues the operation (overruns) until the end of the bus cycle, and relinquishes the bus. When the DMAC overruns, it does not transfer any operands in the subsequent sample interval, because the Limited Rate Auto-Request Mode has the premise to return the bus to the MPU. This mechanism enables the MPU bus cycles even in multi DMAC environment.

In HMCS68000, $\overline{\text{BGACK}}$ signals that a device other than the MPU is using the bus. Since all system DMAC's monitor the common $\overline{\text{BGACK}}$ signal, they each count the $\overline{\text{BGACK}}$ clock cycles as bus masters, even if only one DMAC is the bus master, and determine whether to transfer operands in the following sample interval.

In Maximum Rate Auto-Request Mode, the DMAC takes the bus mastership and transfers all operands until they are exhausted. When the higher priority channels request transfer in this mode, the channel with the Maximum Rate Auto-Request stops the transfer temporarily, and the higher priority channel is serviced. The Maximum Rate channel resumes the transfer after that.

1.4 Signals

HD68450 is bus-compatible with the HMCS68000. Signal lines are shown in Figure 1.8. The address lines A1 through A7 are used to address the DMAC internal registers. A8 through A23 and D0 through D15 are time multiplexed.

The 68000 and system bus control signals and bus arbitration lines are compatible. Chip select ($\overline{\text{CS}}$) is made by decoding address lines. Since the DMAC monitors the bus status through $\overline{\text{BGACK}}$ (Bus Grant Acknowledge) line, the $\overline{\text{BGACK}}$ line is the input/output.

Figure 1.9 shows the bus arbitration timings. The DMAC starts data transfer by 16-20 clock cycles after the transfer request recognition. The interrupt request/acknowledge lines are used to interrupt the MPU according to the interrupt request from I/O devices, or to prepare the vector number ouput by obtaining the interrupt acknowledge cycle from the MPU. An I/O device can request the DMAC for an interrupt through the $\overline{\text{PCL}}$ line (mentioned further on).

The DMAC requests the MPU for an interrupt in the following cases:

1) When the channel operation completes
2) When the block transfer completes
3) When the PCL lines are asserted

When the DMAC receives $\overline{\text{IACK}}$ signal from the MPU, it outputs the vector number D0 to D7. The address/data demultiplex lines are used to demultiplex the time-multiplexed address/data bus.

The $\overline{\text{HIBYTE}}$ signal is asserted when the operand size is byte in Single Addressing Mode, and when the operand is on the upper 8 bits in the data bus; i.e., when the operand in even address is accessed. This signal is used to switch a byte data position between the upper data bus and the lower data bus. $\overline{\text{BEC0-BEC2}}$ are the encoded signals for Exceptions (Refer to Chapter 1.5). FC0-FC2 are function code output signals and are compatible with the HMCS68000 function codes.

An I/O device in each channel is controlled with $\overline{\text{REQ}}$, $\overline{\text{ACK}}$, and $\overline{\text{PCL}}$ lines. $\overline{\text{REQ}}$ is a transfer request signal which is sensed by the edge in Cycle Steal Mode, and sensed by the level in Burst Mode. The $\overline{\text{ACK}}$ signal informs the I/O device of the transfer start, and is used for device chip select, or for negating $\overline{\text{REQ}}$. It is usually outputted when the DMAC addresses an I/O device, but it is not outputted when a 68000 compatible device and Auto request are programmed. By making use of this feature, any channel can operate Memory-to-Memory transfer without addressing the I/O device.

$\overline{\text{PCL}}$ (Peripheral Control Line) is a multiple purposed signal to control a peripheral device. $\overline{\text{PCL}}$ is designated by the $\overline{\text{PCL}}$ bits and DTYP bits of DCR, and can be used as status, interrupt, abort, $\overline{\text{READY}}$, and (E) enable clock inputs, and as start pulse output.

Abort input is used to abort the channel operation, and abort error is recorded in CER. The $\overline{\text{READY}}$ input is used when the I/O device has the $\overline{\text{READY}}$ output, and the DMAC completes the bus cycle after the recognition of the $\overline{\text{READY}}$ signal. The Enable (E) clock input is used when the device is programmed as a 6800 compatible device, and the data transfer becomes synchronous.

The start pulse is outputted when the STR bit of CCR is set and the channel is activated. This is a single active low pulse asserted during four clock cycles which informs the I/O device of the transfer start. $\overline{\text{DONE}}$ and $\overline{\text{DTC}}$ signals indicate the transfer completion. $\overline{\text{DONE}}$ indicates block transfer completion, which is

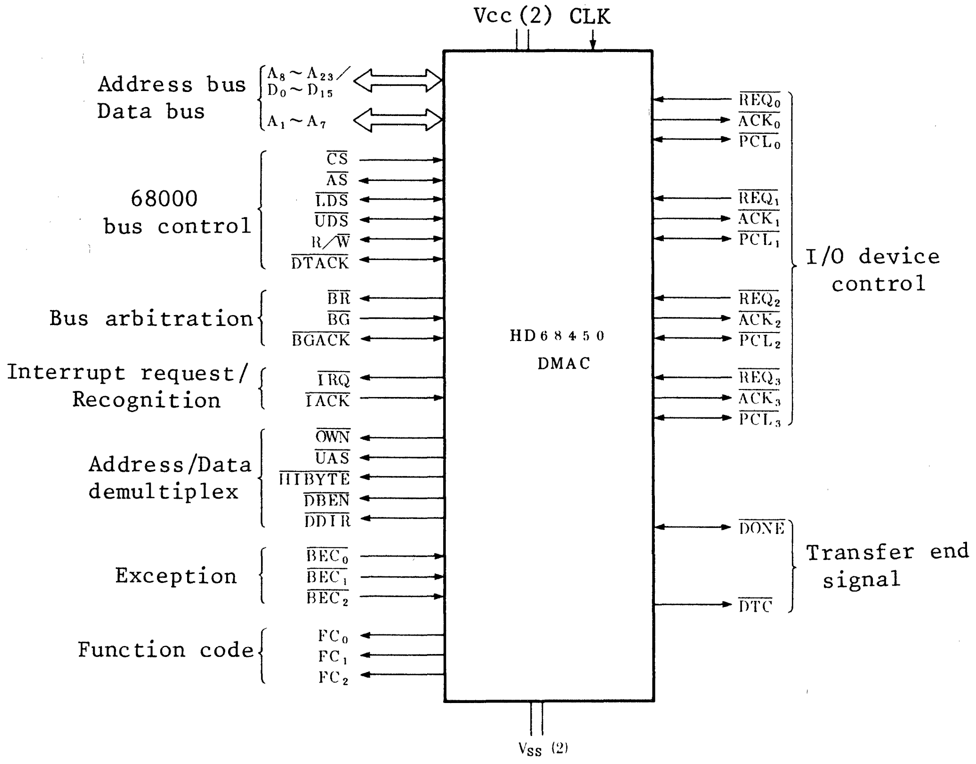


FIGURE 1.8 HD68450 Signal Lines

outputted at the end of each block transfer in Continue Mode, and when all blocks are completely transferred in Chain Modes. This signal is asserted at the same time as the last ACK signal of the transfer. $\overline{DONE}$, therefore, is not outputted in the transfer cycle to the memory in the very last bus cycle when the transfer is from device to memory Dual Addressing.

$\overline{DONE}$ is also used as an input signal in order that the I/O device informs the DMAC of the transfer completion. The DMAC monitors the signal during asserting $\overline{ACK}$ signal. After the $\overline{DONE}$ assertion, the DMAC stops data transfer when the operand transfer is completed, and the channel operation terminates. When the DMAC and I/O device simultaneously assert $\overline{DONE}$, the $\overline{DONE}$ inputted from the device is ignored. The DMAC outputs $\overline{DTC}$ whenever it recognizes $\overline{DTACK}$. In the case of a 6800 compatible device, the DMAC detects the trailing edge of E clock to output $\overline{DTC}$. I/O devices can latch the data by using the falling edge of the $\overline{DTC}$ assertion ($\overline{DTACK}$ is also useful). The $\overline{DTC}$ negation indicates the bus cycle completion. This signal is not outputted when $\overline{DTACK}$ is not inputted, or if exceptions are entered, in order that the I/O device can detect transfer abnormality.

1.5 Exceptions

To be sure of data transfer, the DMAC can stop the bus cycle and retry it, or leave the recovery to the other bus master if an abnormal transfer occurs. The Exceptions are requested by the external devices and are encoded into 3 signals. $\overline{BEC0\text{-}BEC2}$, and inputted into the DMAC. BEC exception conditions are shown in Table 1.6.

The DMAC samples $\overline{BEC}$ signals with the rising edge of the clock and recognizes an exception condition if the $\overline{BEC}$ signals remain in the same level for two or more clock cycles. The DMAC carries out $\overline{BEC}$ exceptions only when $\overline{BEC}$ assertion starts simultaneously, or before $\overline{DTACK}$ assertion, and the $\overline{BEC}$ values remain in the same level for two or more clock cycles. The HALT exception is not implemented until $\overline{DTACK}$ input. If $\overline{BEC}$'s are asserted after $\overline{DTACK}$, the bus cycle occurs normally.

Halt

When Halt is asserted during DMA transfer, the DMAC relinquishes the bus after receiving $\overline{DTACK}$, and after normal bus cycle completion. The DMAC does not arbitrate the bus until HALT is negated.

Halt is useful in the following cases:

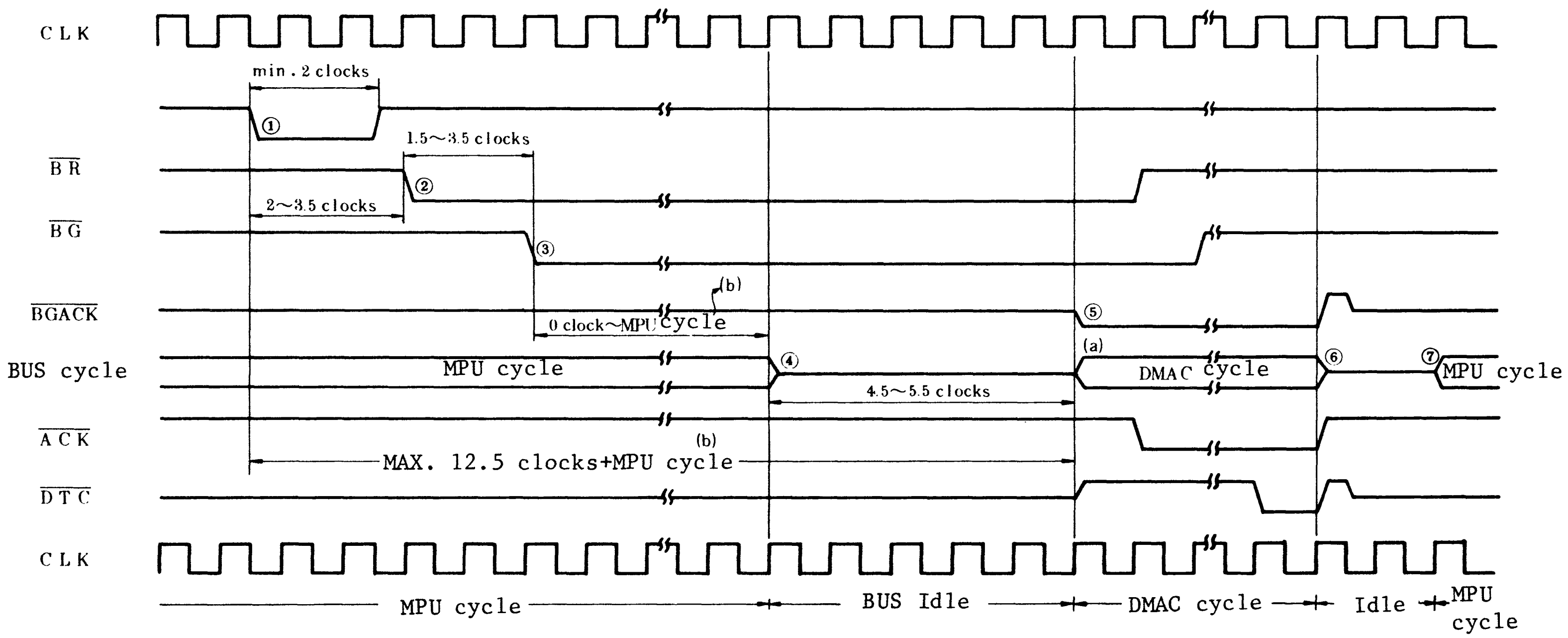
(1) When DMAC turns over the mastership to another bus master without changing the number of the DMAC's bus cycles. Even when the DMAC is using the bus continuously and does not relinquish it, another bus master can get the mastership by halting the DMAC. In this case the DMAC resumes the bus cycle after the bus arbitration (total number of the DMAC's bus cycles does not change).
(2) When transfer "trace" is performed by executing single step bus cycle.

Bus error

When an error occurs during transfer, and the DMAC can not continue the operation or can not get the correct results, Bus error is asserted to stop the transfer abnormality.

The DMAC Bus error sequence is as follows.

① stops the transfer and sets COC bit and ERR bit in CSR.
② checks INT bit in CCR. If INT = 1, the DMAC asserts $\overline{IRQ}$ signal to interrupt the MPU.
③ Keeps the address where the bus error took place and the transfer count left over in the Address Register and Transfer Counter respectively in the channel.
④ relinquishes the bus without other channels' transfer requests.



Cycle Steal Mode
(sensed by
rising edge of REQ)

FIGURE 1.9 Bus Arbitration Timing

TABLE 1.6 (BEC) Exception Condition Types

$\overline{BEC2}$ $\overline{BEC1}$ $\overline{BEC0}$	Exception Conditions	Applications
1 1 1	No exceptions	Usual operation
1 1 0	Halt	Used when DMA trnsfer is stopped temporarily by external circuits.
1 0 1	Bus error	Used when a serious system error occurs. For example, the DMAC bus cycle does not terminate.
1 0 0	Retry	Used when the DMAC bus cycle has not been carried out correctly, and needs retry.
0 1 1	Relinquish and Retry	Used when the MPU uses the bus before the termination of the DMAC bus cycle,and when the DMAC cycle must be continued from the following cycle.
0 1 0	Not used	——
0 0 1	Not used	——
0 0 0	Reset	Power on reset. System reset.

Bus error is useful in the following cases:

(1) When preventing system dead lock (not receiving $\overline{DTACK}$ signal), "a watch dog timer" is used, and the Bus error is asserted when the time is out.

(2) When page fault is recognized in virtual memory environment, Bus error is asserted.

Retry

When Retry is recognized during the DMAC bus cycle, the DMAC stops the bus cycle and repeats the same bus cycle right after the negation of the Retry signal. During the whole sequence, the DMAC holds the bus ($\overline{OWN}$ and $\overline{BGACK}$ are kept asserting).

When the DMAC accesses memory or device, and an error is detected in the transferred operand, external circuitry asserts Retry to transfer the operand again. For example, when an error is found through parity information during a bus cycle, or when $\overline{DTACK}$ does not return in spite of correct address, Retry can be performed.

Relinquish and Retry

When the DMAC recognizes Relinquish and Retry, it sets all control lines, data bus, and address bus to three state, and releases the bus temporarily. If the $\overline{BEC}$ exceptions are negated, the DMAC outputs $\overline{BR}$ again to get the bus mastership and retries the bus cycle in which Relinquish and Retry are asserted.

Relinquish and Retry can be used when the MPU service is necessary to correctly transfer the operand after the bus cycle starts. If the I/O device asserts Relinquish and Retry while requesting an interrupt to the MPU, the DMAC releases the bus so that the MPU may service the interrupt routine, and negates Relinquish and Retry—recovering the fault with minimum overhead. The DMAC obtains the bus again and resumes the transfer.

Reset

When the DMAC recognizes Reset, it relinquishes the bus, clears GCR, and resets DCR, OCR, SCR, CCR, CSR, CPR, and CER of all channels. The interrupt vector registers are set to $ OF(HEX), uninitialized interrupt vector number.

2. System Example

HD68450 DMAC in HMCS68000 is shown in Figure 2.1. Since only basic signals are shown, users are required to add necessary circuitry to an actual system (See Chapter 3). If whole address space is managed with a memory management unit (MMU), the MPU physical address space is the system address bus. The Circuit example is shown in Figure 2.2. The MMU's page fault is encoded to be the DMAC's Bus error input signal. Refer to Chapter 3 for further examples of each circuit.

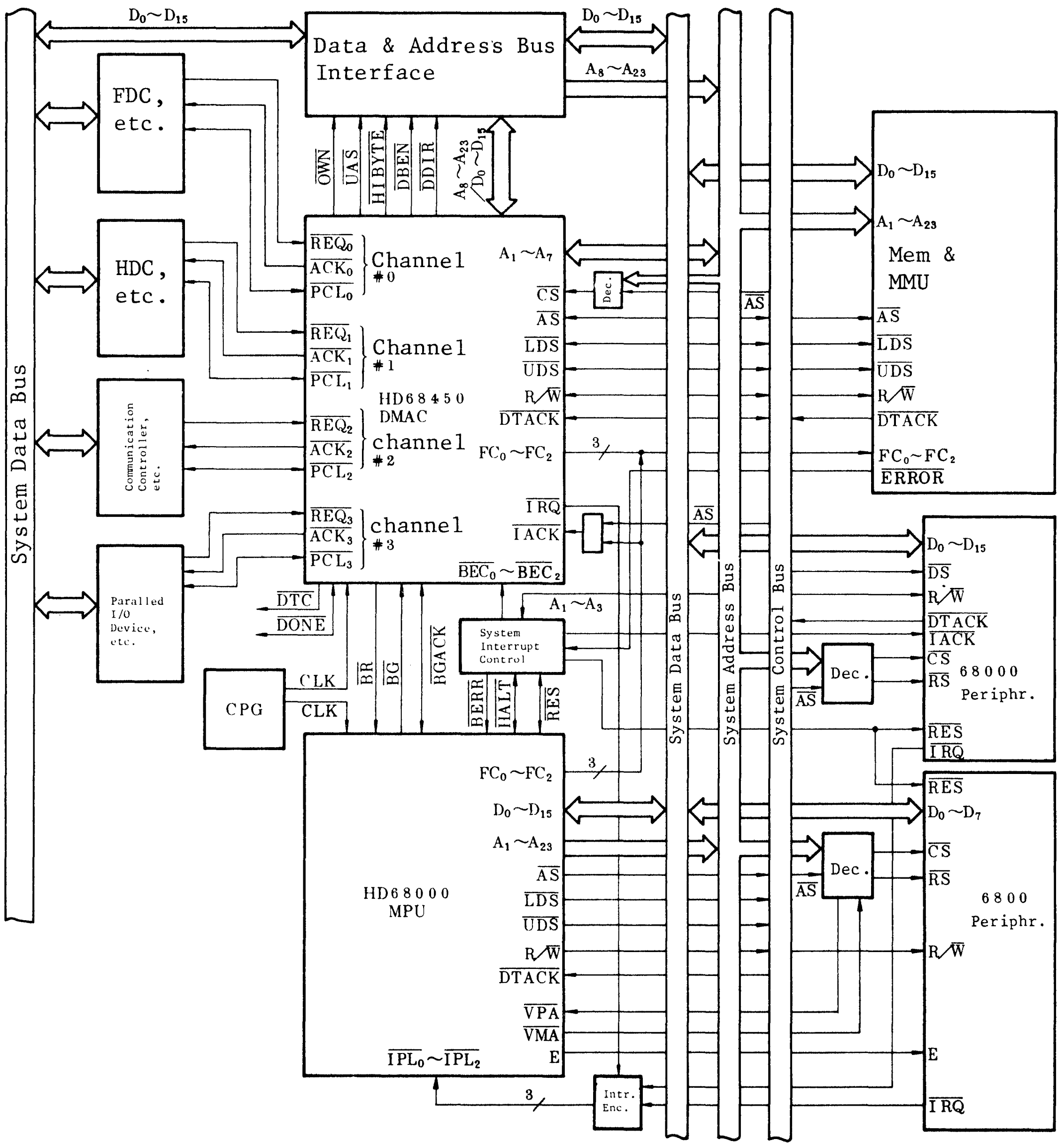


FIGURE 2.1 Basic System Configuration

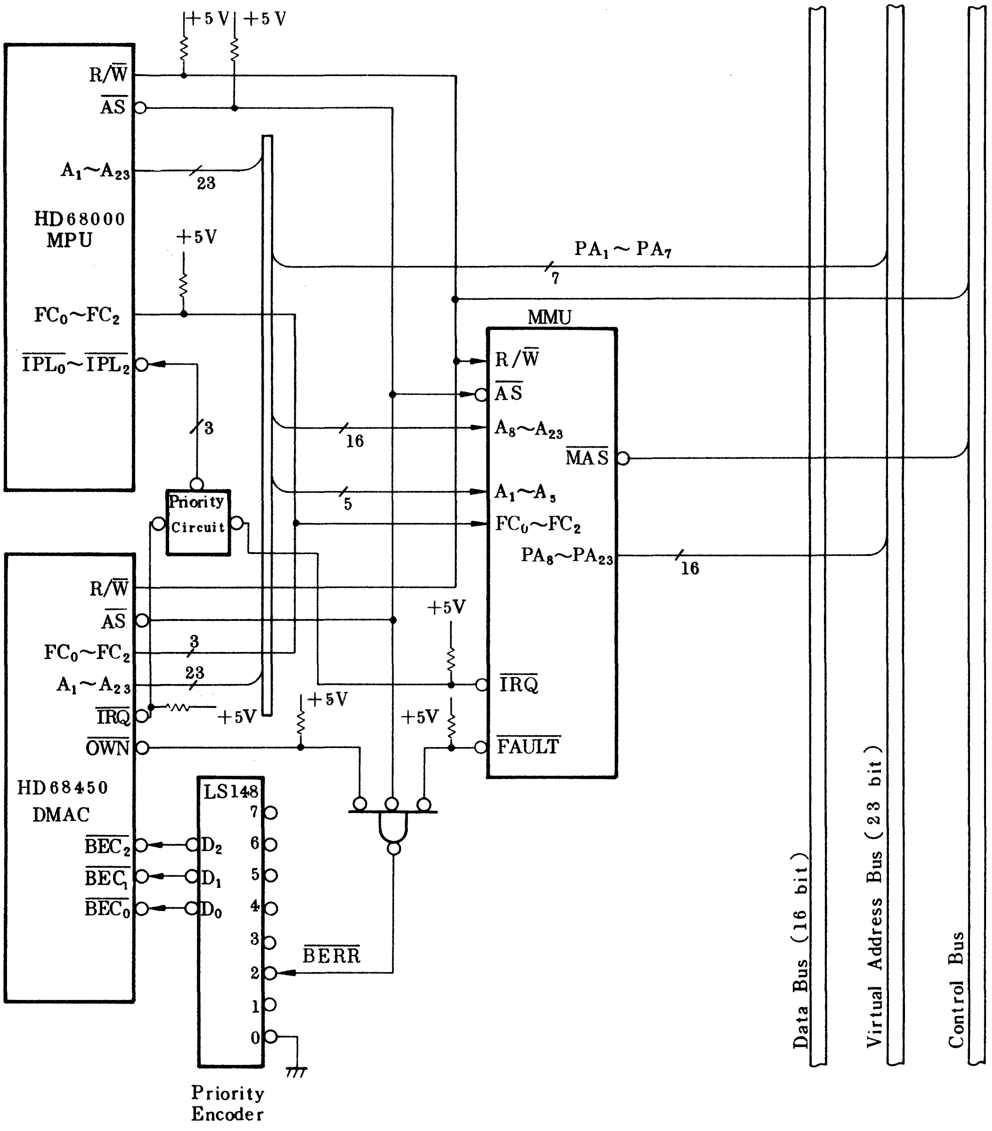


FIGURE 2.2 Conceptual Diagram of the DMAC in virtual address

3. HD68450 Transfer Operation and Circuit Examples

3.1 FIFO Register Operation (Data Pack and Unpack)

As shown in Figure 3.1, the DMAC possesses a 3-byte FIFO (First In First Out) register, which reads and writes an operand in byte or word unit. The FIFO register makes it possible to operate on various operand sizes (abbreviated as OP), and to operate on I/O devices with various port sizes (data bus bit length, abbreviated as P) for memory to I/O transfer. In these operations, the transfer mode is Dual Addressing.

In Figure 3.1, I/O is an I/O device with P=8, and even address. When the DMAC transfers operands from I/O-1 to memory 1 to 6, it reads two byte-operands in the first and second bus cycles from I/O-1 into the FIFO, and writes a word operand in the third bus cycle from FIFO to memory. Thus, the bus efficiency of DMA transfer is increased with PACK operation (to transfer two byte-operands as one word). When the transfer is from memory to I/O-1, a word operand is read from memory 1 and 2 into the FIFO, and is written as two byte-operands into I/O-1 by UNPACK operation (one word into two bytes).

3.2 FC Application Examples

The DMAC possesses the following three registers in each channel:

- MFC (Memory Function Code register)
- DFC (Device Function Code register)
- BFC (Base Function Code register)

In memory access bus cycles in both Single Addressing Mode and Dual Addressing Mode, the MFC contents are outputted through FC0-FC2 pins at the same time as address output. In device access bus cycles in Dual Addressing Mode, the DFC contents are outputted. In Array Chain and Linked Array Chain Modes, the BFC contents are outputted in the bus cycles which load the block information from the Array Table in memory. Because arbitrary values can be written in those function code registers, the data transfer between different memory spaces assigned in a 68000 system (e.g., the supervisor data area or the user data area) becomes possible in Dual Addressing Mode. (See Table 3.1)

TABLE 3.1 68000 Function Code Table

Function Code			Classification
FC2	FC1	FC0	
0	0	0	(Unassigned)
0	0	1	User Data
0	1	0	User Program
0	1	1	(Unassigned)
1	0	0	(Unassigned)
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	Interrupt Acknowledge

FC0-FC2=111 indicates the interrupt acknowledge cycle. The DMAC should not output this code. When $\overline{IACK}$ input is asserted during DMA transfer, address error occurs.

3.3 DMAC Interrupt Request Examples

The DMAC can output $\overline{IRQ}$ to request an interrupt to the MPU under the conditions shown in Table 3.2. "L" means $\overline{IRQ}$ assertion. $\overline{IRQ}$ is asserted as long as those conditions are satisfied. To negate $\overline{IRQ}$ (make "H" level), INT bit in CCR must be reset, or "FF(HEX)" must be written in CSR to reset CSR.

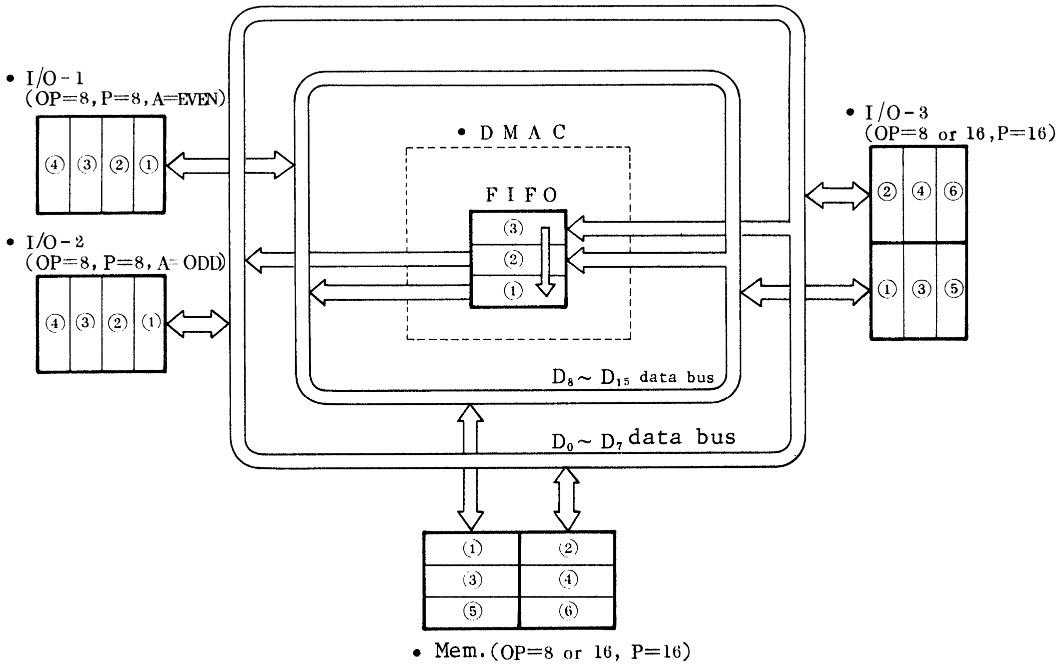


FIGURE 3.1 Data Bus Connection Example in Dual Addressing Mode

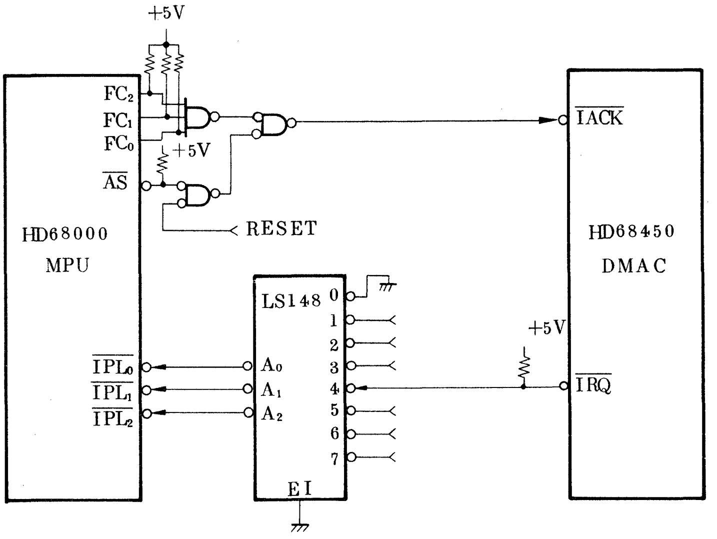


FIGURE 3.2 Connection Example of $\overline{\text{IRQ}}$ and $\overline{\text{IACK}}$

Various transfer examples using FIFO are given in the followings.

Example 1) I/O (OP=8, P=8, A=EVEN, ① to ③) ⟶ Memory (② to ④)

DMAC bus cycle
R-B --- 1 byte read from I/O (①)
W-B --- 1 byte write to memory (②)
R-B --- 1 byte read from I/O (②)
W-B --- 1 byte write to memory (③)
R-B --- 1 byte read from I/O (③) *
W-B --- 1 byte write to memory (④)

* When TC (Transfer word Counter) $\leqq$ 2, and P=8, PACK does not occur.

Example 2) I/O (OP=8, P=8, A=EVEN, ④ to ①) ⟶ Memory (④ to ①)

DMAC bus cycle
R-B --- 1 byte read from I/O (④)
R-B --- 1 byte read from I/O (③)
W-W --- 1 word write to memory (④ ③) *
R-B --- 1 byte read from I/O (②)
W-B --- 1 byte write to memory (②)
R-B --- 1 byte read from I/O (①)
W-B --- 1 byte write to memory (①)

* Data inputs in the order which address decreases.

Example 3) I/O (OP=8, P=8, A=ODD, ① ① ①*) ⟶ Memory (① ① ①*)

```
DMAC bus cycle
  R-B --- 1 byte read from I/O ( ① )
  W-B --- 1 byte write to memory ( ① )
  R-B --- 1 byte read from I/O ( ① )
  W-B --- 1 byte write to memory ( ① )
  R-B --- 1 byte read from I/O ( ① )
  W-B --- 1 byte write to memory ( ① )
```

* does not count the address

Example 4) Memory (① to ④) ⟶ I/O (OP=8, P=8, A=EVEN, ④ to ①)

```
DMAC bus cycle
  R-W --- 1 word read from memory ( ① ② )
  W-B --- 1 byte write to I/O ( ④ )
  W-B --- 1 byte write to I/O ( ③ )
  R-W --- 1 word read from memory ( ③ ④ )
  W-B --- 1 byte write to I/O ( ② )
  W-B --- 1 byte write to I/O ( ① )
```

Example 5) Memory (① to ④) ⟶ I/O (OP=8, P=16, ② to ⑤)

```
DMAC bus cycle
  R-W --- 1 word read from memory ( ① ② )
  W-B --- 1 byte write to I/O ( ② )
  R-W --- 1 word read from memory ( ③ ④ )
  W-W --- 1 word write to I/O ( ③ ④ )
  W-B --- 1 byte write to I/O ( ⑤ )
```

Example 6) Memory (③ to ①) ⟶ I/O (OP=8, P=16, ② to ④)

```
DMAC bus cycle
  R-B --- 1 byte read from memory ( ③ )
  R-W --- 1 word read from memory ( ② ① )
  W-B --- 1 byte write to I/O ( ② )
  W-W --- 1 word write to I/O ( ③ ④ )
```

Example 7) Memory (① ① ① ①) ⟶ I/O (OP=8, P=16, ② ② ② ②)

```
DMAC bus cycle
  R-B --- 1 byte read from memory ( ① )
  R-B --- 1 byte read from memory ( ① )
  W-B --- 1 byte write to I/O ( ② )
  W-B --- 1 byte write to I/O ( ② )
  R-B --- 1 byte read from memory ( ① )
  W-B --- 1 byte write to I/O ( ② )
  R-B --- 1 byte read from memory ( ① )
  W-B --- 1 byte write to I/O ( ② )
```

Example 8) Memory (① to ④) ⟷ I/O (OP=16 or 32, P=16, ① to ④)

```
DMAC bus cycle
  R-W --- 1 word read from memory ( ① ② ) or I/O ( ① ② )
  W-W --- 1 word write to I/O ( ① ② ) or memory ( ① ② )
  R-W --- 1 word read from memory ( ③ ④ ) or I/O ( ③ ④ )
  W-W --- 1 word write to I/O ( ③ ④ ) or memory ( ③ ④ )
```

Figure 3.2 shows $\overline{\text{IRQ}}/\overline{\text{IACK}}$ examples in the DMAC and the MPU system, where the interrupt level of the DMAC is four. However, this level is arbitrary.

When the multi block transfer is in Continue Mode or in Chaining Modes, the transfer status needs to be checked between block transfers in some applications. In Continue Mode, since the BTC bit is set after the first block transfer completes, the DMAC can request interrupt according to Table 3.2.

In Chaining Modes the DMAC cannot request interrupt at the end of each block transfer. Instead, when the last block transfer completes, interrupt request is possible because the COC bit is set. In Chaining Modes, if the DMAC needs to request interrupt at the end of each block transfer, circuits shown in Figure 3.3. are required. Appropriate values have been written in BFC, MFC, and DFC, and the PCL signal is formed by decoding the function codes, to enable the DMAC to request interrupt. (It should be determined whether the FC's are used by the Memory Management Unit (MMU).

Figure 3.4 shows $\overline{\text{BG}}$ mask example. Because an interrupt has a higher priority than a data transfer, BG should be masked in $\overline{\text{IACK}}$ cycle.

3.4 Peripheral Control Line (PCL) Operations

$\overline{\text{PCL}}$ pin of each channel can be used for four different functions realized by setting PCL bits and DTYP bits in DCR as shown in Table 3.3. However, Mode 3 becomes invalid when the device type is 6800, or $\overline{\text{ACK}}$ type with $\overline{\text{READY}}$, or 68000-type in Auto-Request Mode. Similarly, Mode 4 becomes invalid when the device type is 6800, or $\overline{\text{ACK}}$ type with $\overline{\text{READY}}$.

In Mode 1, PCT bit in CSR is set when $\overline{\text{PCL}}$ line is asserted ("H" to "L"). Mode 1 is useful to record a status change of an I/O device. The timing chart for setting the PCT bit is shown in Figure 3.5.

Mode 2 is the function to interrupt the MPU via the DMAC from the I/O device, using the $\overline{\text{PCL}}$ signal change from "H" to "L". In this case, the INT bit of CCR should be set. The timing

TABLE 3.2 $\overline{\text{IRQ}}$ Output Condition

CCR	CSR							$\overline{\text{IRQ}}$ Output
INT	COC	BTC	NDT	ERR	ACT	PCT*	PCS	
0	X	X	X	X	X	X	X	H
1	0	0	0	0	X	0	X	H
1	1	X	X	X	0	X	X	L
1	0	1	0	0	1	X	X	L
1	0	0	0	0	X	1	X	L

*: When the PCL function is set on interrupt input.
X: don't care.

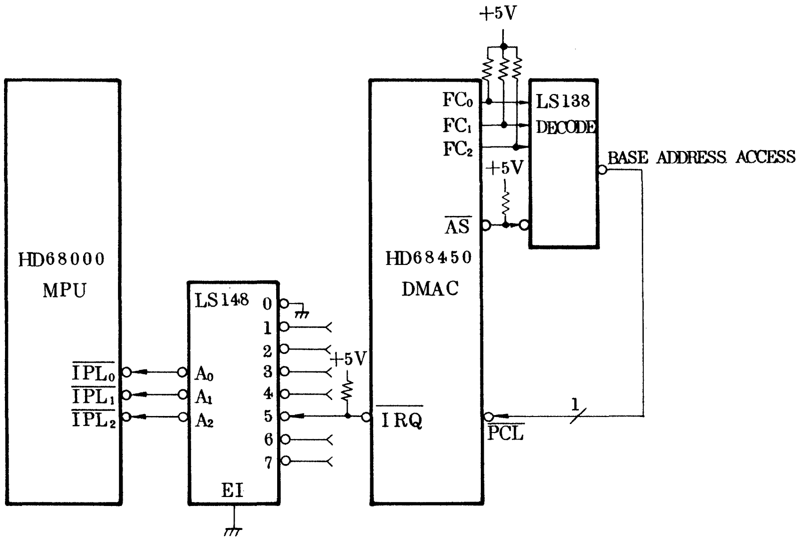


FIGURE 3.3 Circuit Example to Generate Interrupt at the end of each block transfer in Chaining Modes

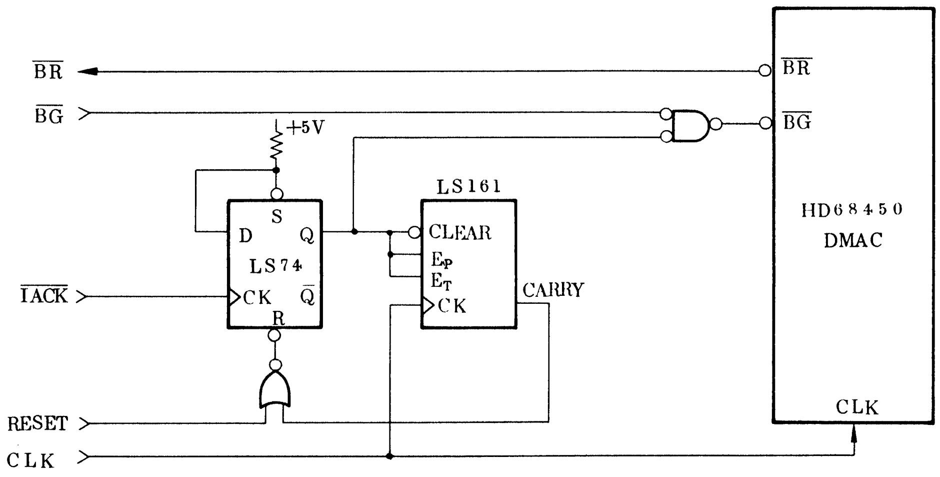


FIGURE 3.4 $\overline{\text{BG}}$ input Mask example

chart from PCL signal change to IRQ output is shown in Figure 3.5.

Mode 3 is used to ascertain the internal process time interval to activate channels, since the STR bit of CCR is set. Table 3.4 shows the necessary CLK cycles in Mode 3 from the MPU write cycle to set STR bit until start pulse output.

Mode 4 aborts the current transfer. This signal is inputted through $\overline{\text{PCL}}$, and EXTERNAL ABORT ERROR is recorded in CER, and ERR bit is set in CSR. Timing is shown in Figure 3.5.

3.5 Demultiplex Examples for Address/Data Multiplexed Bus

As described in Chapter 1.4, $\overline{\text{OWN}}$, $\overline{\text{UAS}}$, $\overline{\text{DBEN}}$, and $\overline{\text{DDIR}}$ are used for bus demultiplexing. $\overline{\text{OWN}}$ is used for bi-directional buffer control. Signal application examples are shown in Figure 3.6.

3.6 $\overline{\text{HIBYTE}}$ Application Example (Bus Matching)

Data transfer between devices with different port sizes in Dual Addressing Mode is described in Chapter 3.1. In Single

TABLE 3.3 Conditions to set $\overline{\text{PCL}}$ functions

Mode	PCL Function Mode	DCR				OCR
		PCL,H	PCL,L	DTYP,H	DTYP,L	REQG,H
1	Status Input	0	0	×	×	×
2	Status Input with Interrupt	0	1	×	×	×
3	Start Pulse, Negative 1/8 CLK	1	0	1	0	×
				×	0	1
4	Abort Input	1	1	×	0	×

× : don't care

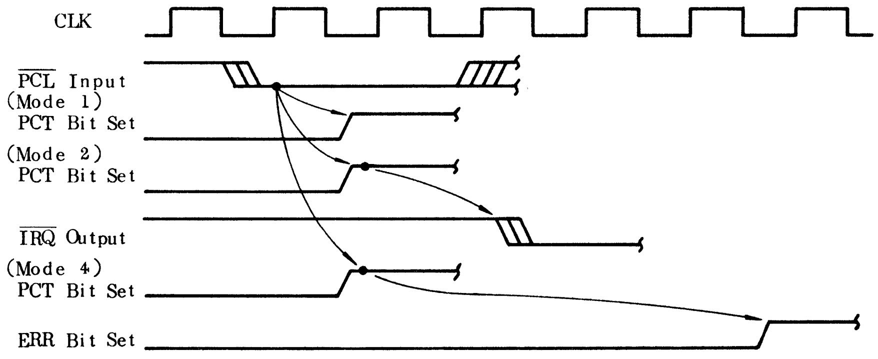


FIGURE 3.5 Timings for Mode 1, 2, and 4

TABLE 3.4 Clock Cycles from the MPU Write Cycle to set STR bit to output Start Pulse (Mode 3)

Transfer Mode	CLK Numbers*
No Chain	39
Array Chain	59
Link Array Chain	61

*MPU write cycle: 14 clock cycles
DMAC memory read cycle: 4 clock cycles

Addressing Mode, $\overline{\text{HIBYTE}}$ is used for bus matching.

Figure 3.7 gives an example of bus matching between an 8-bit I/O device and a 32-bit memory system. As shown, the I/O device must be in the lowest byte of the data bus. $\overline{\text{HIBYTE}}$ is outputted only when even address is accessed, and when the DMAC operates byte operand in the Single Addressing Mode. See Figure 3.5.

The example shown in Figure 3.8 is between a 16-bit I/O device and a 32-bit memory system.

3.7 Low Speed I/O Device Circuit Example

Figure 3.9 shows a circuit for a low speed I/O device; e.g., floppy disc controller. Figure 3.10 gives the timing chart. Since a DMA transfer request signal ($\overline{\text{DRQ}}$) from a low speed I/O device is generated in every DMA transfer cycle, the channel is programmed to be External Request and Cycle Steal Mode. The data latch timing in write cycle (memory→device) is the timing when the write enable signal ($\overline{\text{WE}}$) changes from "L"→"H". Data on the data bus is valid only while the data strobe signal ($\overline{\text{UDS}}$ or $\overline{\text{LDS}}$) is "L"; therefore, the data latch timing must be made from DTC assertion timing ("H"→"L"). This assertion occurs at least 30ns earlier than the $\overline{\text{UDS}}$ or $\overline{\text{LDS}}$ negation ("L"→"H").

3.8 High Speed I/O Device Circuit Example

FIFO is used as external data buffer in the example shown. Figure 3.11 shows the application of the DMAC and FIFO. Figure 3.12 gives the control timing chart in read and write cycles to FIFO. Since data of several words is continuously transferred in DMA transfer between FIFO and memory, the external request mode should be set to Burst Mode. The data write timing to FIFO is derived from $\overline{\text{DTC}}$ output, and the timing to negate the Burst request from "L" to "H" is made with up/down counter.

In write cycles to FIFO, the Burst request is negated synchronously with $\overline{\text{DTC}}$ assertion, when the counter number reaches "the operand number transferred in a burst" ("16" in Figure 3.11).

In read cycles from FIFO, the Burst request is negated synchronously with $\overline{\text{DTC}}$ when the counter number becomes two. In Burst Mode, the Burst request in both read and write cycles should be negated before the last transfer starts. In the last DMA transfer when TC=0 (transfer words counter = 0), $\overline{\text{DONE}}$ is outputted at the same timing as $\overline{\text{ACK}}$. This signal is used to reset the Burst request.

3.9 6800 Family Application Examples

Since 6800 family devices are given their addresses on 68000 memory, and are used by memory mapping, the transfer mode is Dual Addressing. The block diagram is shown in Figure 3.13. Please note:

1) E clock is inputted from the $\overline{\text{PCL}}$ pin, and is used to synchronize 6800 devices and the DMAC.
2) 6800 devices close the data bus at the falling edge of E clock in read cycle from the 6800 device. The DMAC, however, latches the data when $\overline{\text{DTC}}$ is asserted. Therefore, the data outputted from the 6800 device needs to be latched by the external latch.
3) For 6800 device chip select, the address decoder and the address strobe are used.

Figure 3.14 shows an application of HD68A43 (FDC) and HD68B21 (PIA). The FDC makes a request by setting TxRQ High. The negated TxRQ is inputted to $\overline{\text{PCL}}$ as $\overline{\text{READY}}$.

3.10 Encode Example for Exceptions

An Exception request is made by external circuits and is inputted into the DMAC's $\overline{\text{BEC}_0}$ ~ $\overline{\text{BEC}_2}$. Figure 3.15 indicates an encode example.

Exception Examples: Figure 3.16 shows the bus cycle time out error example. The transfer stop example is given in Figure 3.17.

If the DMAC does not have the bus, do not input the bus Exceptions. Exceptions should be inputted after the $\overline{\text{AS}}$ output (or $\overline{\text{UAS}}$ negation), as shown in Figure 3.15.

3.11 Priority Circuit Example (Daisy Chaining)

When multi DMAC's are used, priority circuits like Daisy Chain are required. In the following example, the DMAC nearer the MPU has higher priority.

3.12 8086 System Application Examples

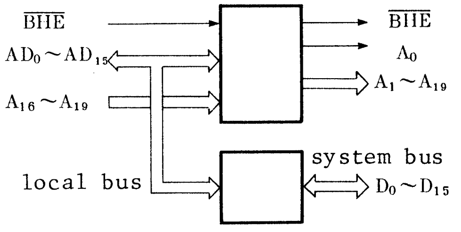
Applied in an 8086 system, the HD68450 is superior to other DMAC alternatives because of the following features:

1) High speed data transfer operation by Single Addressing Mode
2) Ease of operation for multi block transfer
3) Maximum bus exception utilization

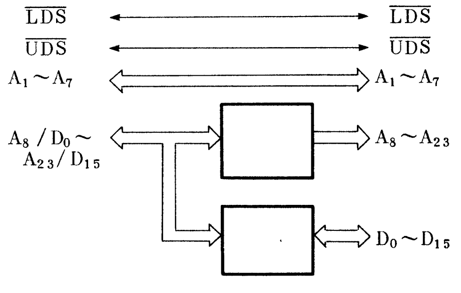
Basic differences between the 8086 system and the HD68450 are as follows:

1) Address bus, data bus
2) Memory Structure

The HD68450 and the 8086 are different in arrangement of address and data bus. Address bus is connected to the system bus through LS373 latch. Data bus is connected to the system bus through LS245, bi-directional transceiver.

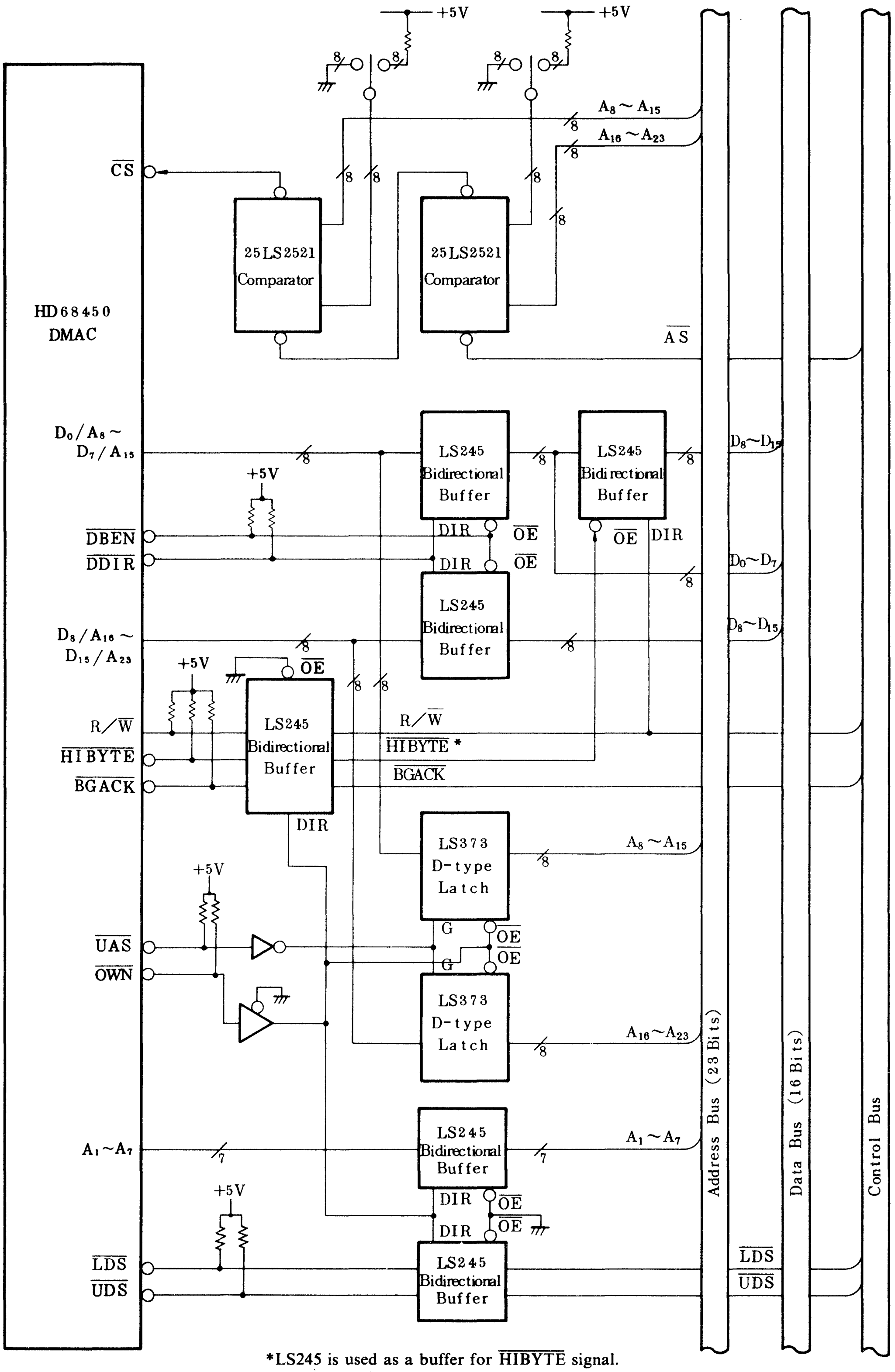


8086



HD68450

The HD68450 and the 8086 have different ways to address memory. When HD68450 is used in the 8086 system, $\overline{\text{UDS}}$ (Upper Data Strobe) should be connected to A0 and $\overline{\text{LDS}}$ (Lower Data Strobe) to $\overline{\text{BHE}}$. For data bus, the upper byte bus and the lower byte bus must be switched. In this configuration, the 8086 can access the internal registers of the HD68450 by the same method as memory.



*LS245 is used as a buffer for HIBYTE signal.
FIGURE 3.6 Demultiplex Examples for Time Multiplexed Bus

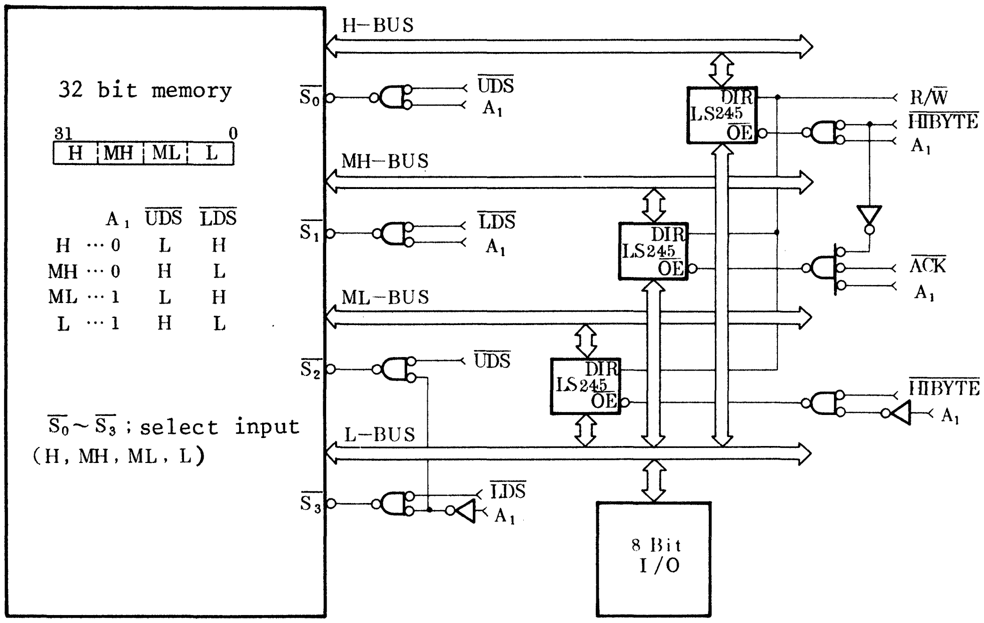


FIGURE 3.7 Bus Matching (8 bit I/O-32 bit memory)

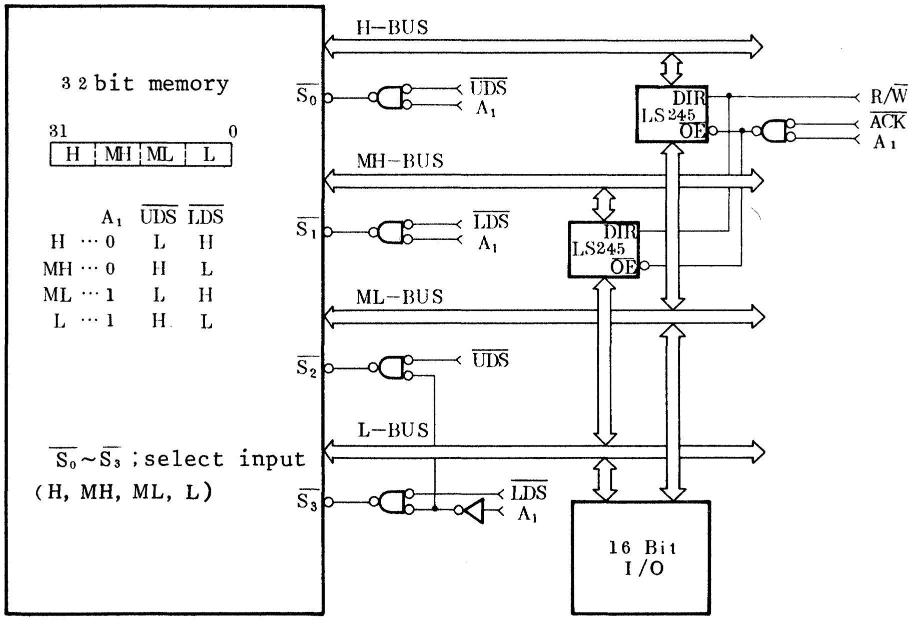
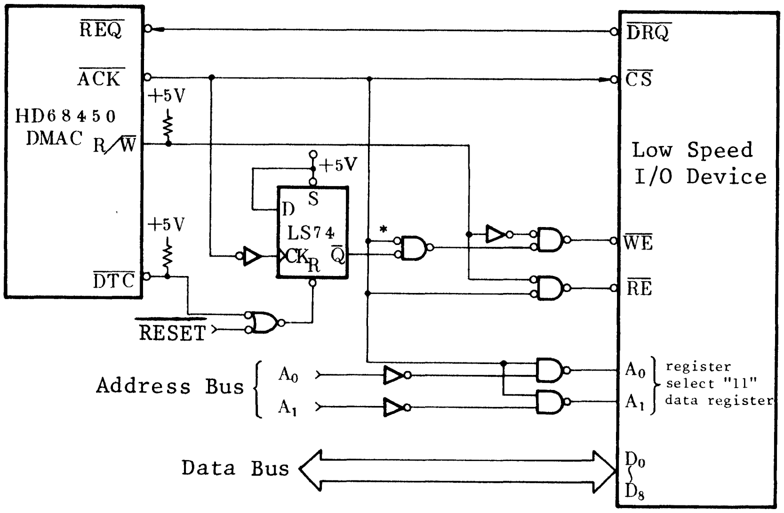


FIGURE 3.8 Bus Matching (16 bit I/O-32 bit memory)



* to negate $\overline{WE}$ when $\overline{DTC}$ is not outputted

FIGURE 3.9 Low Speed I/O Device Application

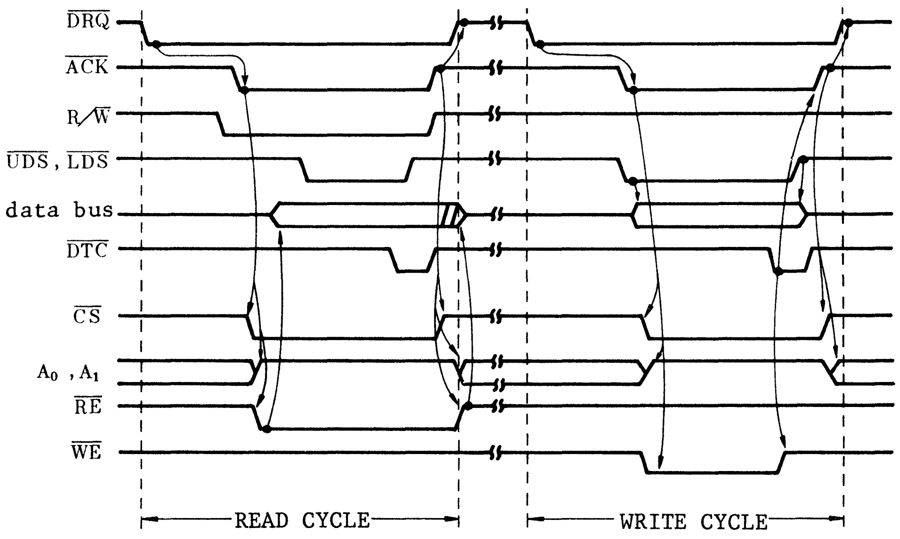
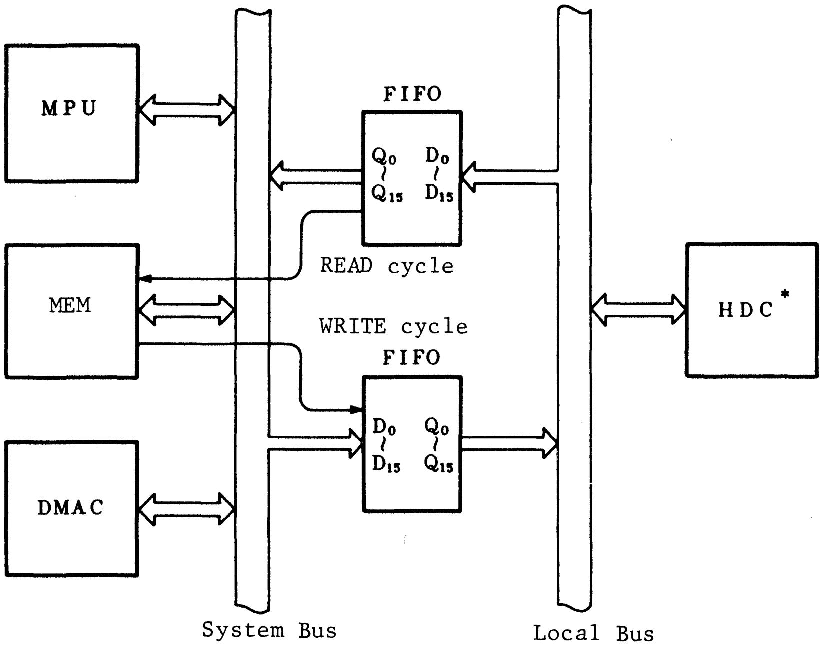


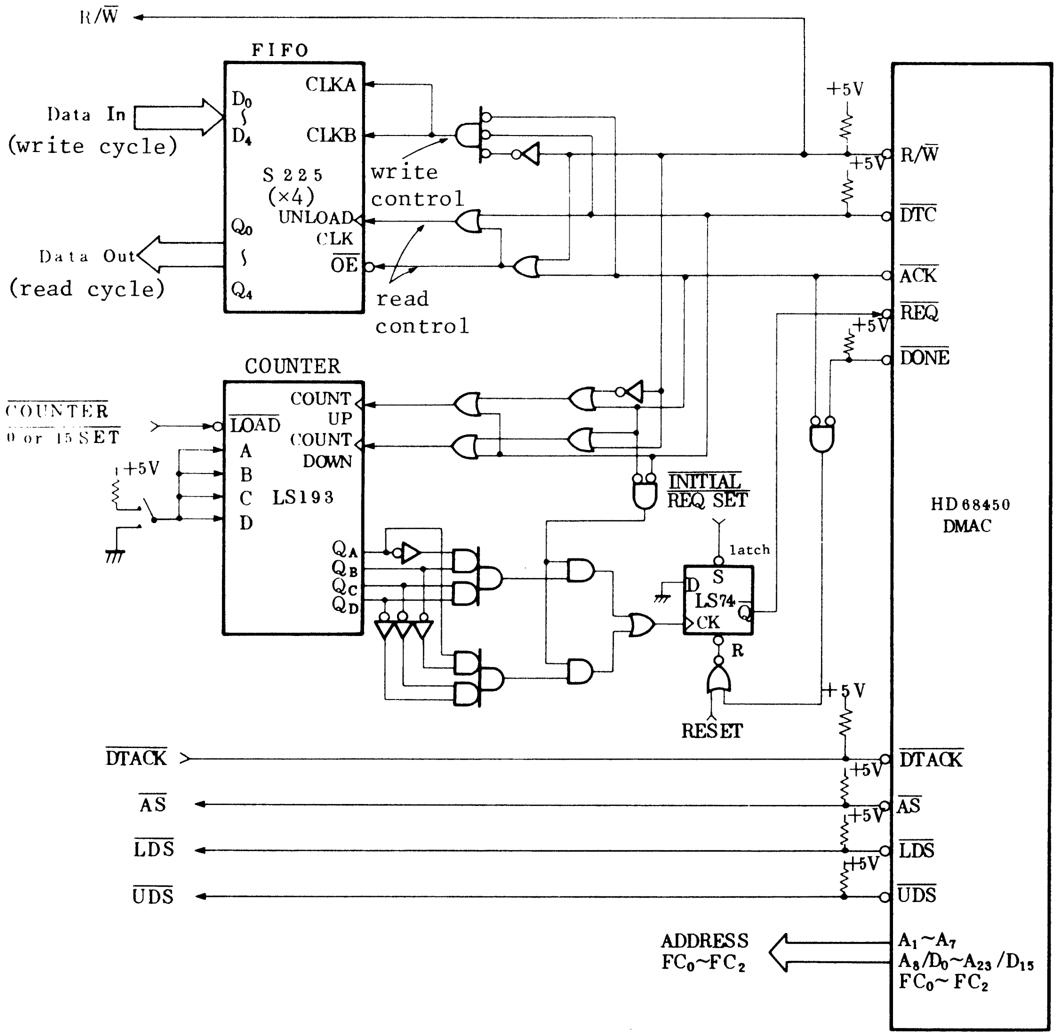
FIGURE 3.10 Timing chart of Fig. 3.9



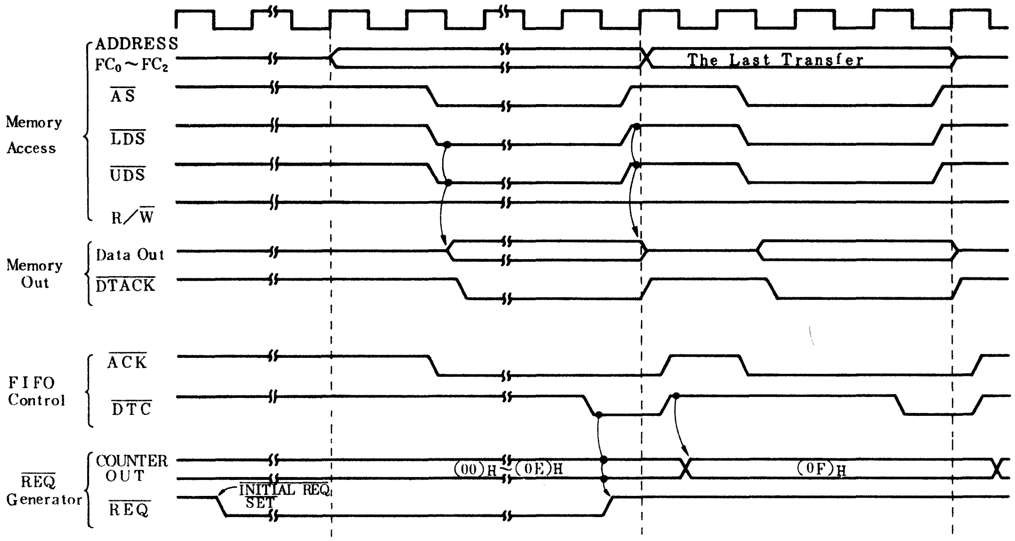
* Hard Disc Controller

(a) System Example of FIFO used as HDC Buffers

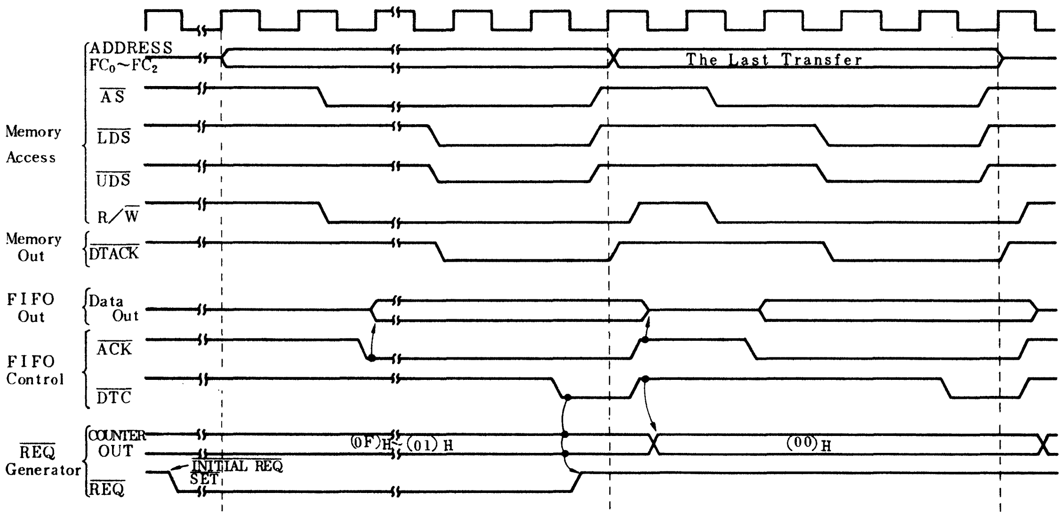
FIGURE 3.11 FIFO Application Examples



(b) Circuit Example Between the DMAC and FIFO



(a) Write (MEM→FIFO)



(b) Read (FIFO→MEM)

FIGURE 3.12 Control Timing Chart of Figure 3.11 (b)

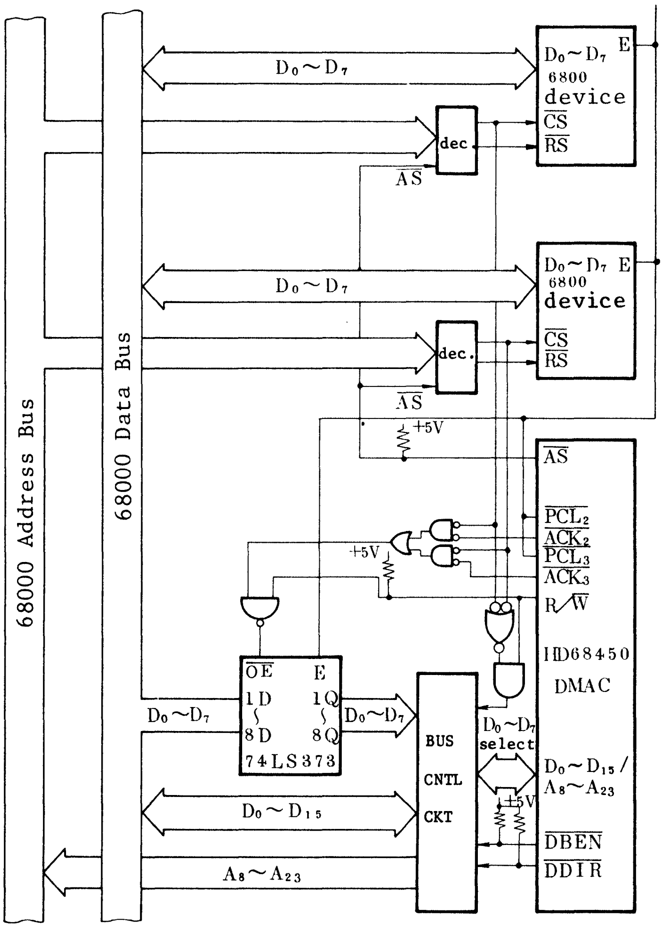


FIGURE 3.13 6800 Device Application Example

LS 74 : D-type Positive Edge-Trigger F-F

LS 161 : Synchronous 4-bit Binary Counter

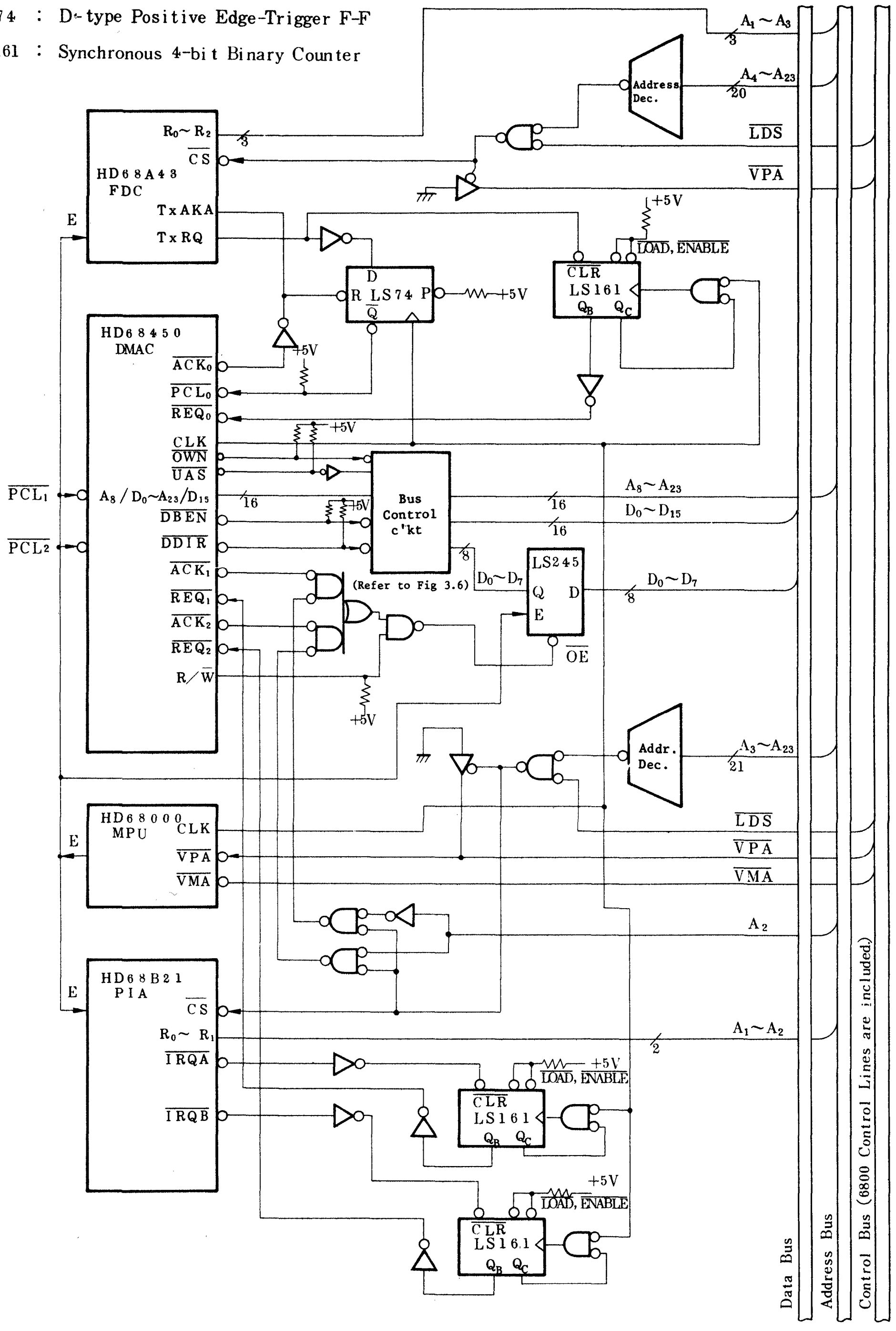


FIGURE 3.14 Circuit Example of HD68450, HD68A43 (FDC) and HD68B21 (PIA)

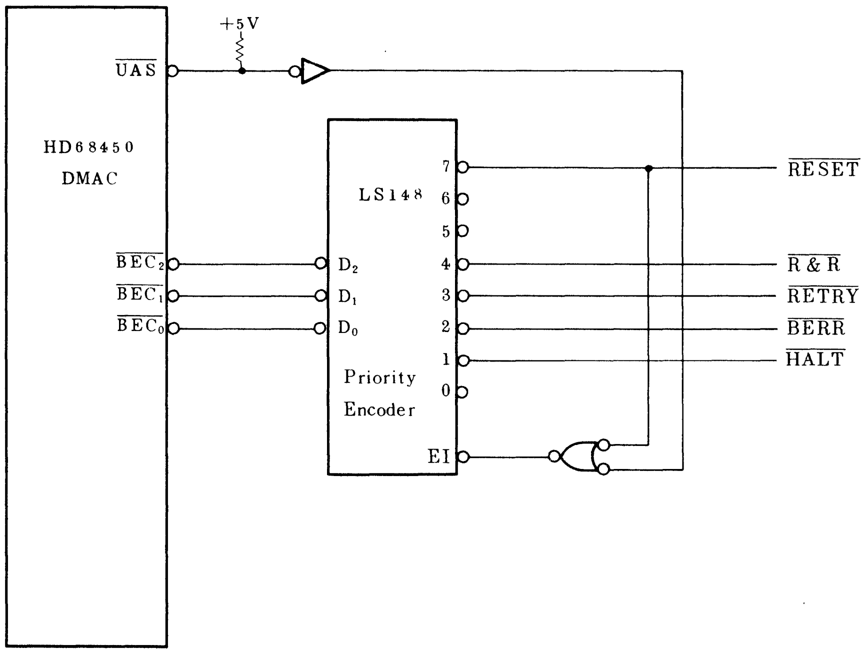


FIGURE 3.15 Exception Encode Example

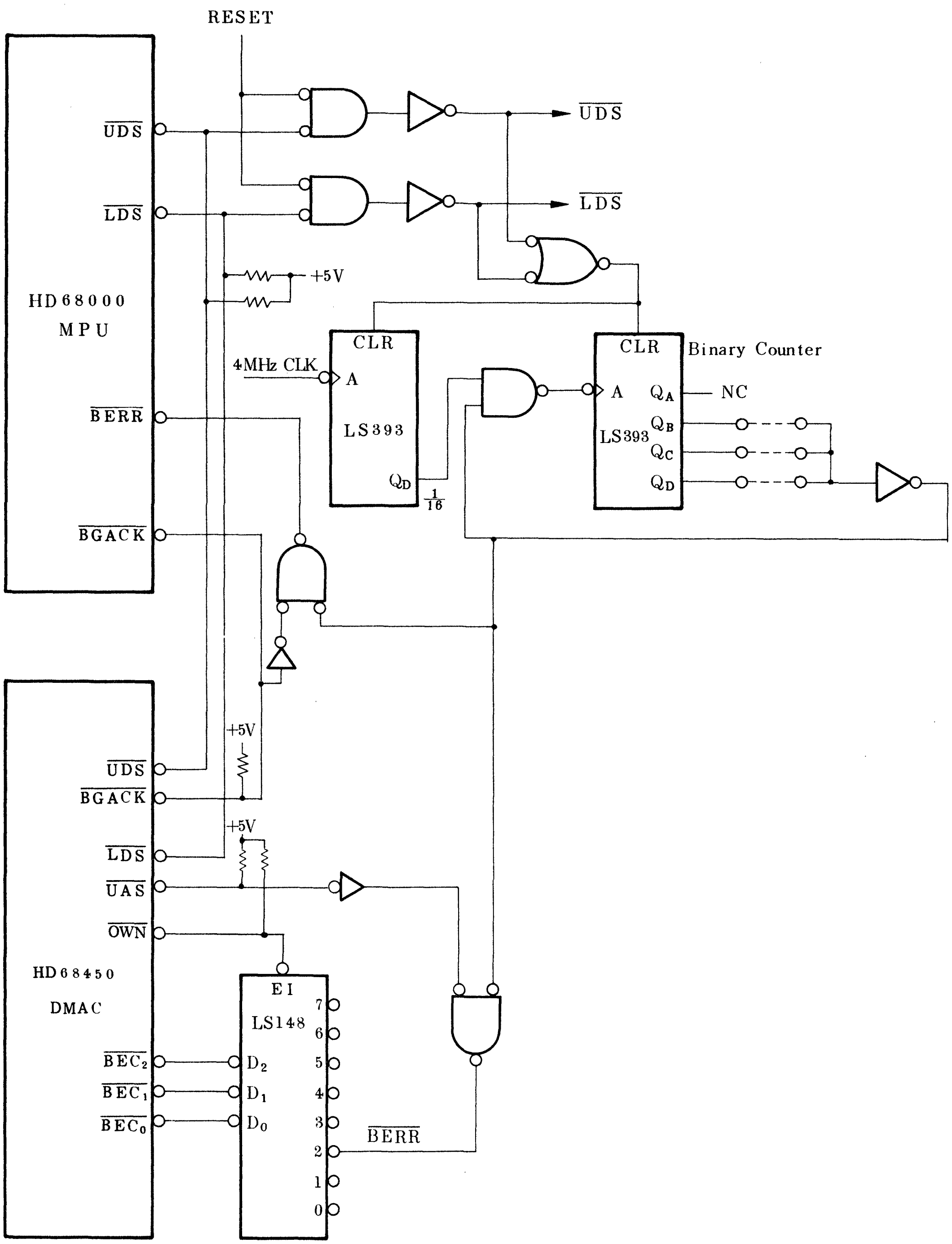
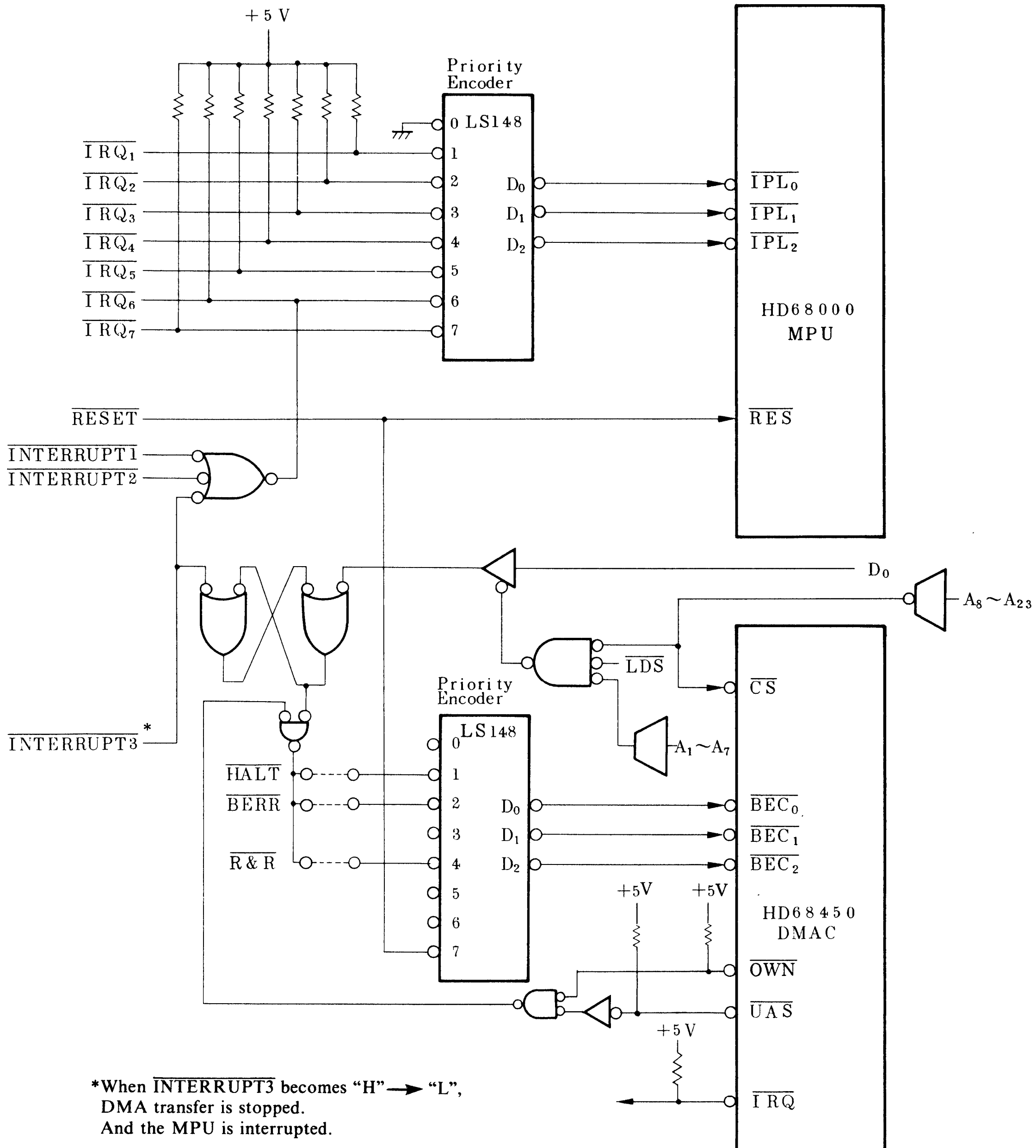
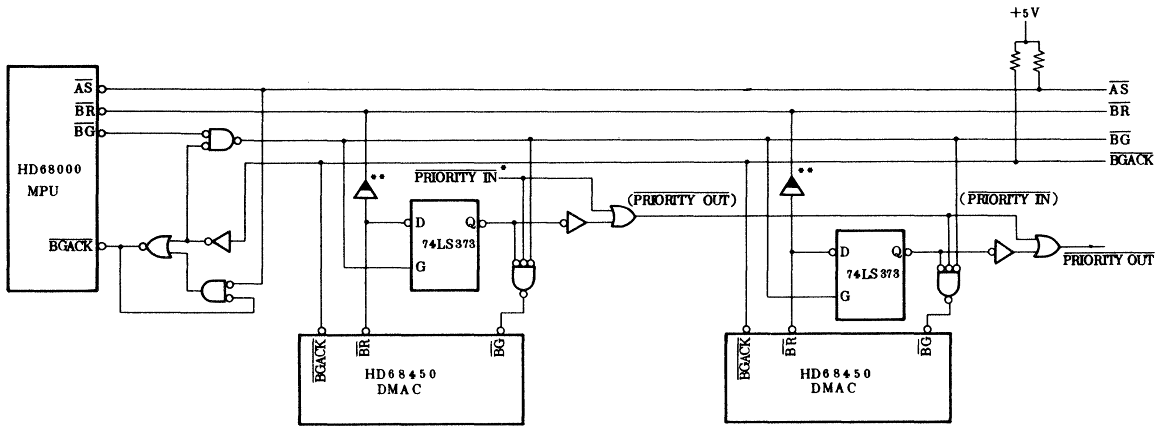


FIGURE 3.16 Bus Cycle Time Out Error Example



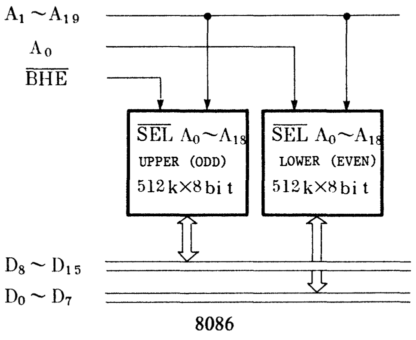
*When $\overline{\text{INTERRUPT3}}$ becomes "H" ⟶ "L",
DMA transfer is stopped.
And the MPU is interrupted.

FIGURE 3.17 Transfer Stop Example



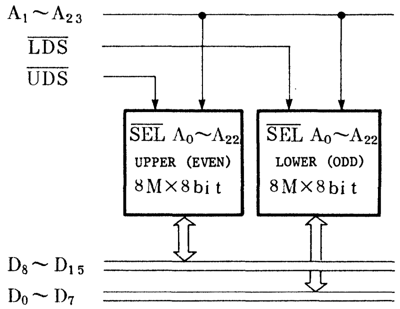
*This $\overline{\text{PRIORITY IN}}$ must be grounded.
**Open collector buffer.

FIGURE 3.18 Daisy Chain Example



$A_1 \sim A_{19}$
A_0
$\overline{BHE}$
$\overline{SEL}$ $A_0 \sim A_{18}$ UPPER (ODD) 512k×8bit
$\overline{SEL}$ $A_0 \sim A_{18}$ LOWER (EVEN) 512k×8bit
$D_8 \sim D_{15}$
$D_0 \sim D_7$

8086



$A_1 \sim A_{23}$
$\overline{LDS}$
$\overline{UDS}$
$\overline{SEL}$ $A_0 \sim A_{22}$ UPPER (EVEN) 8M×8bit
$\overline{SEL}$ $A_0 \sim A_{22}$ LOWER (ODD) 8M×8bit
$D_8 \sim D_{15}$
$D_0 \sim D_7$

HD68450

The 8086 system allows one word operand whose upper and lower bytes are located at both contiguous and diagonal position in memory, as in the figure at the top of page 31. HD68450 does not allow one word operand (see (2) in the figure). However, if the operand size is programmed as a byte, and memory count is programmed as increase in Dual Addressing Mode, the "diagonal" position can be supported by the HD68450.

In addition to the Dual Addressing Mode (Chapter 3.1), the HD68450 supports Single Addressing Mode, in which OP=P must be satisfied. For one word operand in diagonal position (2), OP=P=8 is required, and the I/O device must be connected to the upper byte. When an operand is transferred from the I/O devicε to the lower byte of memory, $\overline{\text{HIBYTE}}$ signal must be used. See Chapters 3.5 and 3.6 for circuit examples of $\overline{\text{HIBYTE}}$.

Figure 3.19 shows an application example of the HD68450 in the 8086 system, which requires the following circuits:

(1) $\overline{\text{CS}}$, $\overline{\text{IACK}}$ GENERATOR ...	Read/Write control for HD68450 internal registers
(2) BUS ARBITER	8086 bus arbitration control
(3) STATUS GENERATOR.....	Control for form status input to 8288 from FC0-FC2
(4) RDY GENERATOR	Synchronizing 8086 and HD68450 in internal register read/write cycles

(1) $\overline{\text{CS}}$, $\overline{\text{IACK}}$ GENERATOR

Figures 3.20 and 3.21 show a circuit example and timing chart of $\overline{\text{CS}}$ and $\overline{\text{IACK}}$ GENERATOR. $\overline{\text{CS}}$ and $\overline{\text{IACK}}$ are formed from the $\overline{\text{IORC}}$, $\overline{\text{ATOWC}}$, and $\overline{\text{INTA}}$ outputted from 8288. The read/write cycle of the 8086 MPU to the HD68450 starts when $\overline{\text{CS}}$, $\overline{\text{LDS}}$, $\overline{\text{UDS}}$, and R/$\overline{\text{W}}$ become valid, and ends when both $\overline{\text{LDS}}$ and $\overline{\text{UDS}}$ become inactive.

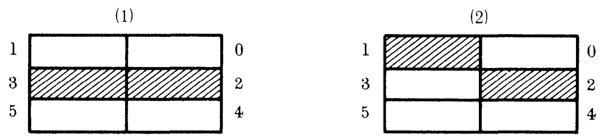
Since the HD68450 must output data to the lower byte of the data bus, both lower bytes of 8086 and HD68450 need to be directly connected, and the output from 8286 must be masked to avoid bus conflict.

(2) BUS ARBITER

Figures 3.22 and 3.23 show the bus arbiter circuit and its timing chart. As long as the HD68450 outputs $\overline{\text{BR}}$ or $\overline{\text{BGACK}}$, bus mastership is requested to the MPU, and bus conflict does not take place. $\overline{\text{BR}}$ becomes inactive one clock after $\overline{\text{BGACK}}$ output, and the bus request does not become inactive before the HD68450 becomes bus master.

(3) STATUS GENERATOR

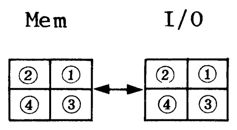
Figures 3.24, 3.25 and 3.26 show the Status Generator circuit and the DMA read/write cycle timing charts. This circuit generates status signals to inform the DMAC's bus ownership to 8288. The HD68450 outputs VC0-FC2 in every bus cycle. These values can be varied by writing different values into MFC, DFC, and



The following examples show various data transfer between memory and I/O device in Dual Addressing Mode.

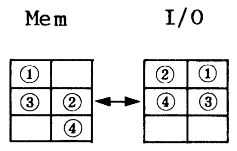
<u>Example 1</u> Memory ←→ I/O (P=16, OP=16, MTC=2)

R-W (① ②) --- 1 word read from memory (I/O)
W-W (① ②) --- 1 word write to I/O (memory)
R-W (③ ④) --- 1 word read from memory (I/O)
W-W (③ ④) --- 1 word write to I/O (memory)



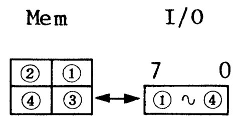
<u>Example 2</u> Memory ←→ I/O (P=16, OP=8, MTC=4)

R-W (① ②) --- 1 word read from I/O
W-B (①) --- 1 byte write to memory
R-W (③ ④) --- 1 word read from I/O
W-W (② ③) --- 1 word write to memory
W-B (④) --- 1 byte wirte to memory



<u>Example 3</u> Memory ←→ I/O (P=8, OP=8, MTC=4)

R-B (①) --- 1 byte read from I/O
R-B (②) --- 1 byte read from I/O
W-W (① ②) --- 1 word write to memory
R-B (③) --- 1 byte read from I/O
W-B (③) --- 1 byte write to memory
R-B (④) --- 1 byte read from I/O
W-B (④) --- 1 byte write to memory



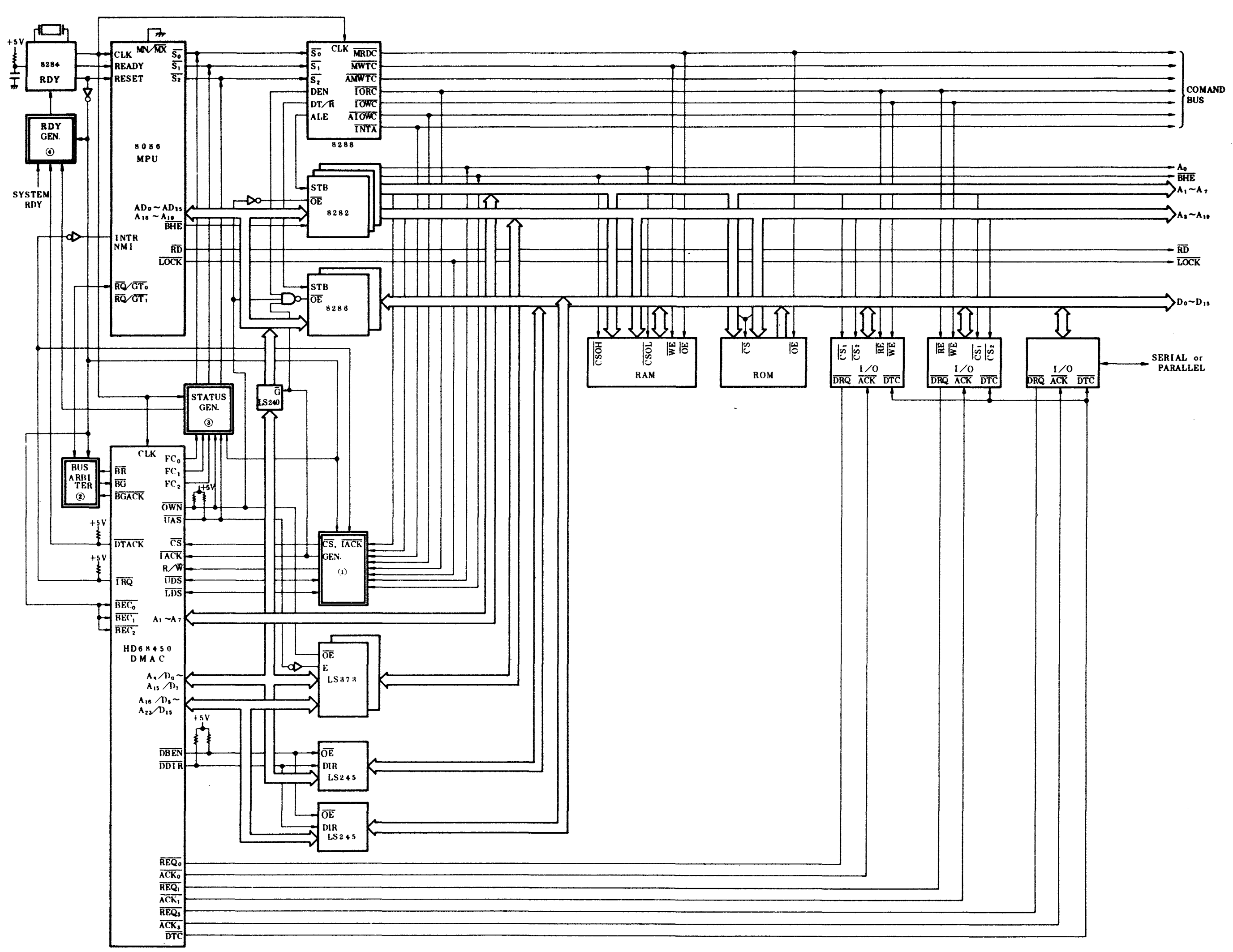


FIGURE 3.19 Application Example of HD68450 in the 8086 System

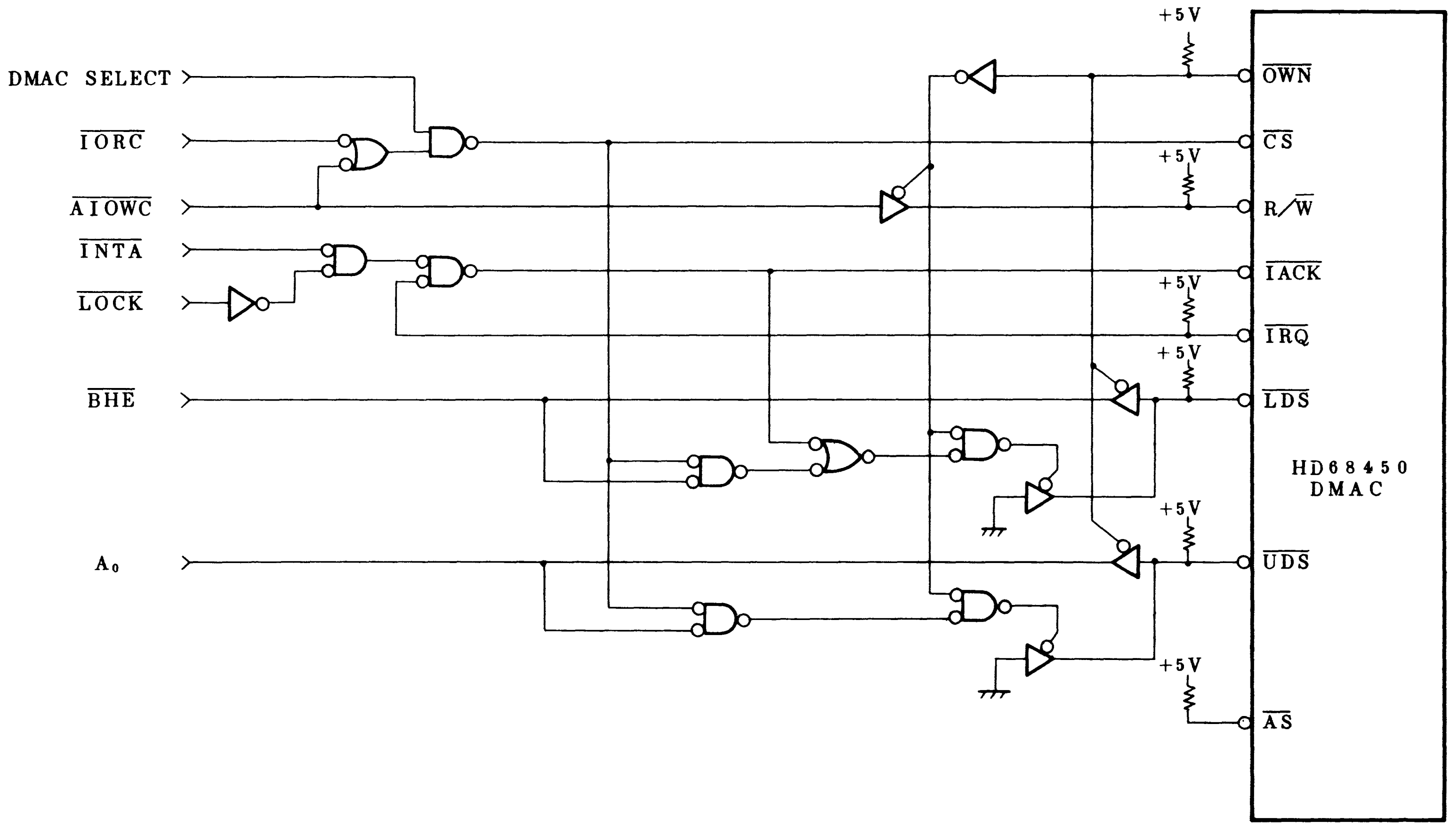
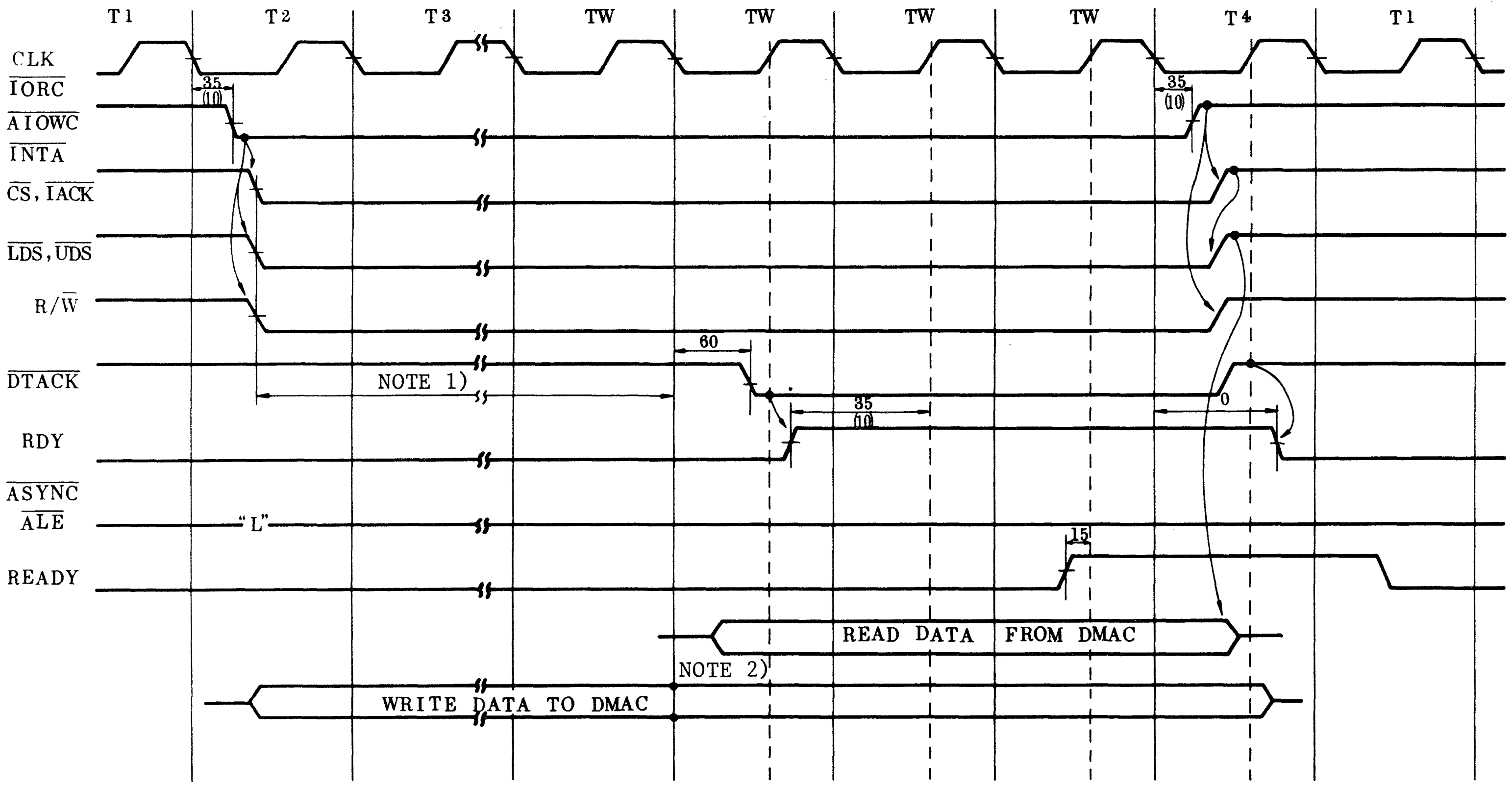


FIGURE 3.20 $\overline{\text{CS}}$ and $\overline{\text{IACK}}$ Generator



NOTE 1) Read and INTA cycles, consist of 13 clocks and write cycle consists of 10 clocks.
NOTE 2) DMAC Latches the data at a falling edge of this clock.

FIGURE 3.21 Timing Chart of Fig. 3.20

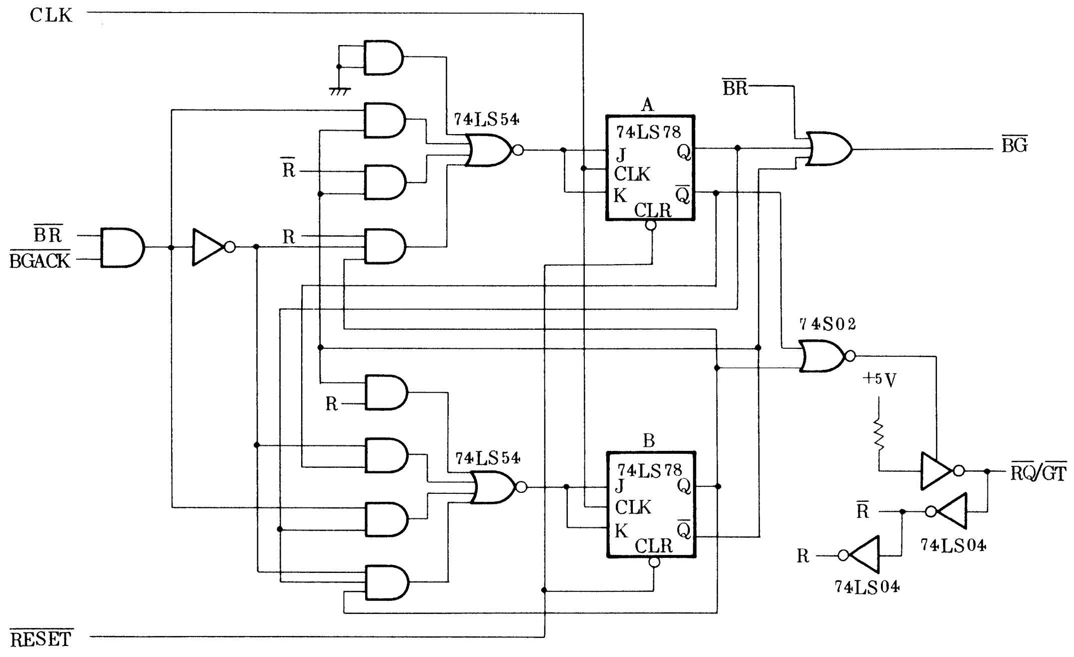


FIGURE 3.22 Bus Arbitration Circuit

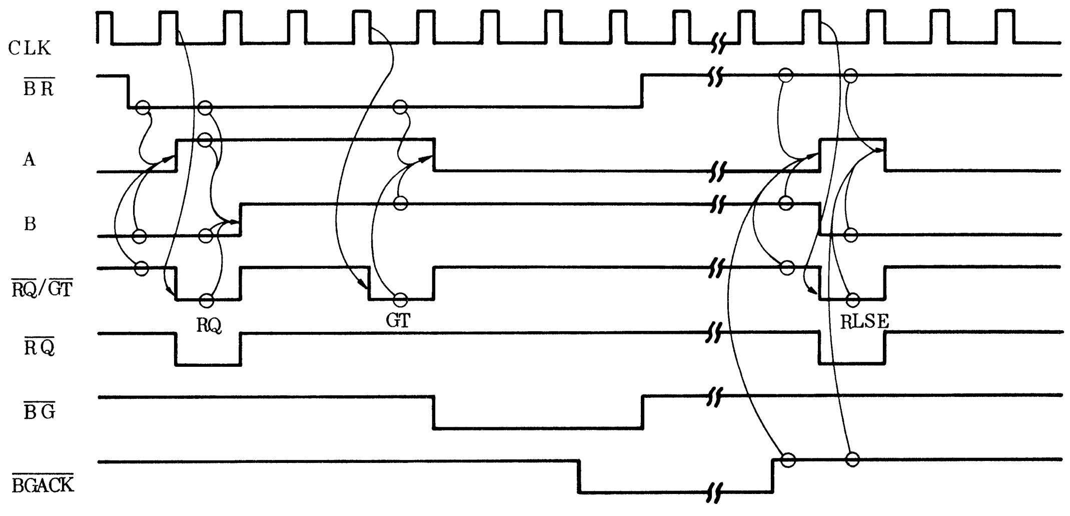


FIGURE 3.23 Bus Arbitration Timing

BFC (Memory Function Code register, Device Function Code register and Base Function Code register). When the values in the table are written in the registers, 8288 outputs bus commands synchronizing with the DMAC's bus cycle, and the DMAC can address devices on the 8086 system bus.

Figure 3.24 shows the shortest bus cycle, consisting of 5 clock cycles. $\overline{DS0}$–$\overline{DS2}$ turn idle when the outputs from LS191 are "3." When access to memory or I/O device is not in time for the bus cycle, it is possible to prolong the HD68450 bus cycle by changing the outputs of LS191 to "4."

(4) RDY GENERATOR

Figure 3.27 shows the RDY Generator circuit. See Figure 3.21 for the DMAC's RDY timing. In Figure 3.27, the STEM RDY signal is used when the 8086 accesses devices other than the HD68450.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	8086 STATUS	
0	0	0	interrupt acknowledge --------------	$\overline{INTA}$ output
0	0	1	read I/O port ---------------------	$\overline{IORC}$ output
0	1	0	write I/O port --------------------	$\overline{IOWC}$, $\overline{AIOWC}$ output
0	1	1	halt ------------------------------	None
1	0	0	code access -----------------------	$\overline{MRDC}$ output
1	0	1	read memory -----------------------	$\overline{MRDC}$ output
1	1	0	write memory ----------------------	$\overline{MWTC}$, $\overline{AMWTC}$ output
1	1	1	idle ------------------------------	None

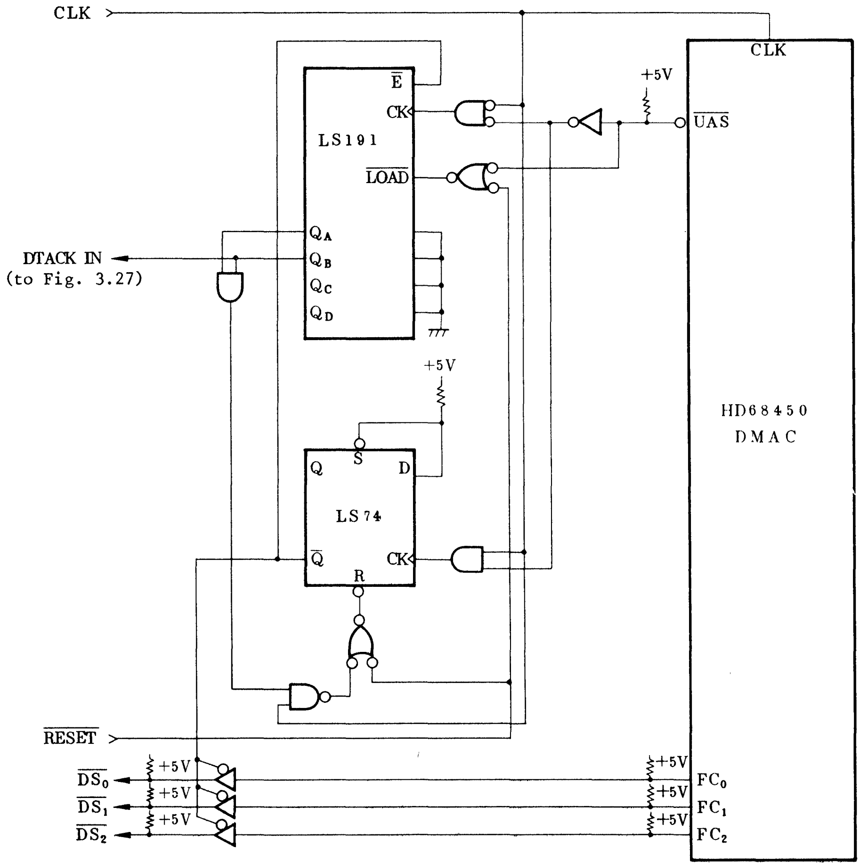
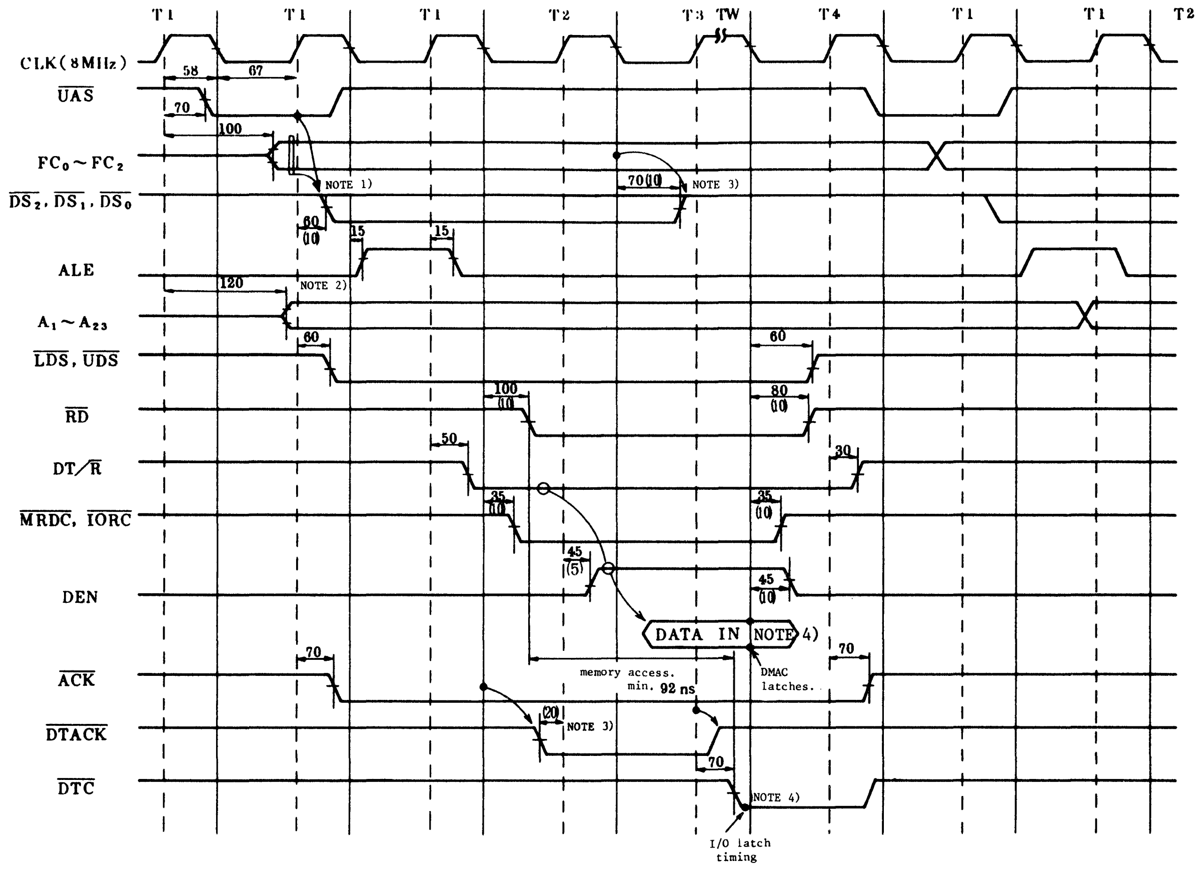
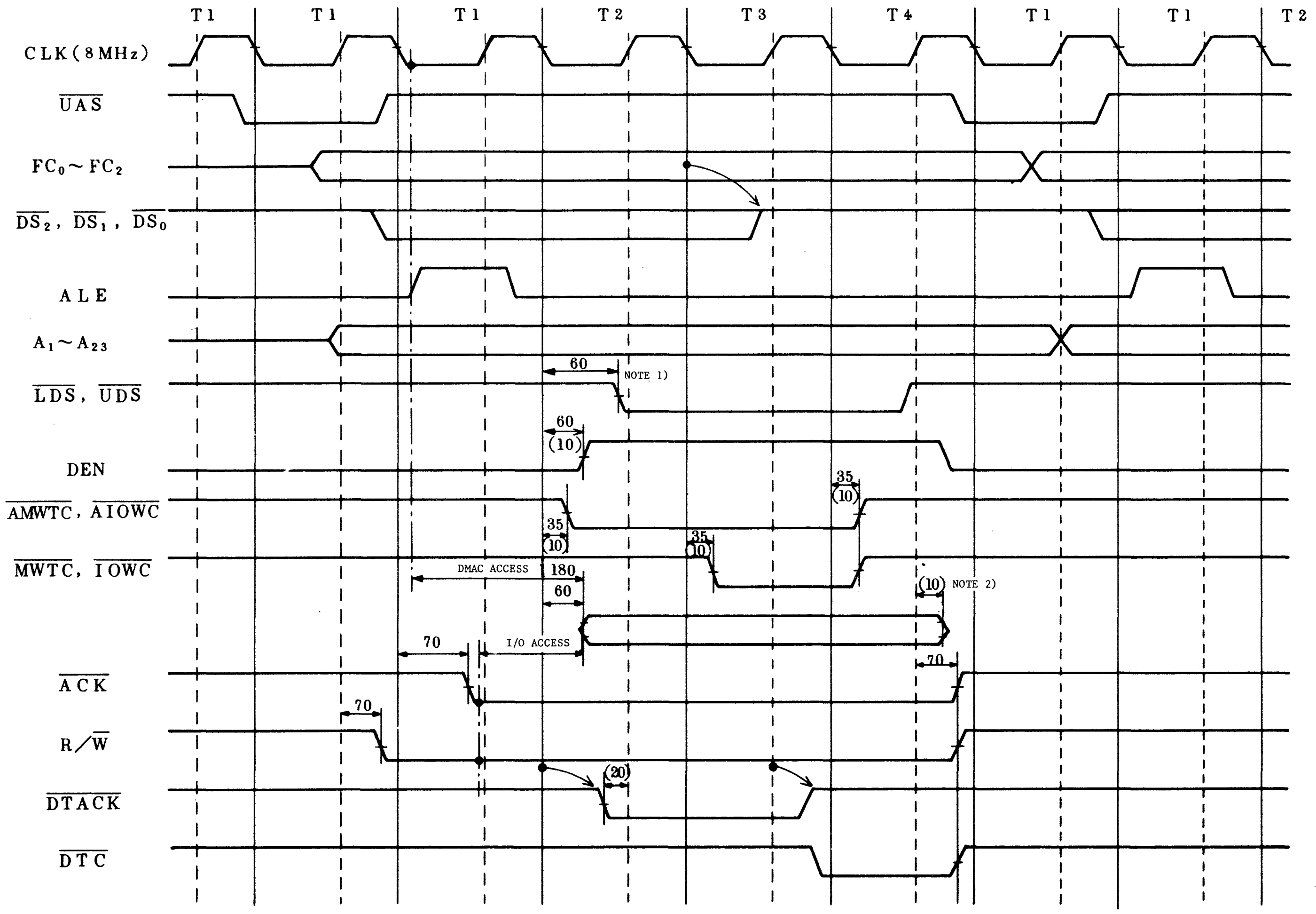


FIGURE 3.24 Status Generator



NOTE 1) $\overline{DS2}$, $\overline{DS1}$ and $\overline{DS0}$ correspond to $\overline{S2}$, $\overline{S1}$, and $\overline{S0}$ in the 8086 system, and are from FC0-FC2 of the DMAC. When the DMAC is used, each bus cycle needs one idle state (T1), and the basic bus cycle consists of five clock cycles.
NOTE 2) $\overline{OWN}$ and $\overline{UAS}$ of the DMAC are used, and ALE of the 8288 is not used to latch address A1-A23.
NOTE 3) $\overline{DS2}$, $\overline{DS1}$, and $\overline{DS0}$ are used to terminate the 8288 cycle, and $\overline{DTACK}$ is used to terminate the DMAC.
NOTE 4) Data latch in Dual Addressing Mode, and from I/O device in Single Addressing Mode is with the falling edge of $\overline{DTC}$.

FIGURE 3.25 HD68450 READ Cycle Timing Chart



NOTE 1) $\overline{LDS}$, $\overline{UDS}$ (corresponding A0, $\overline{BHE}$) become valid late.
NOTE 2) Data hold time is 10 ns.

FIGURE 3.26 HD68450 WRITE Cycle Timing Chart

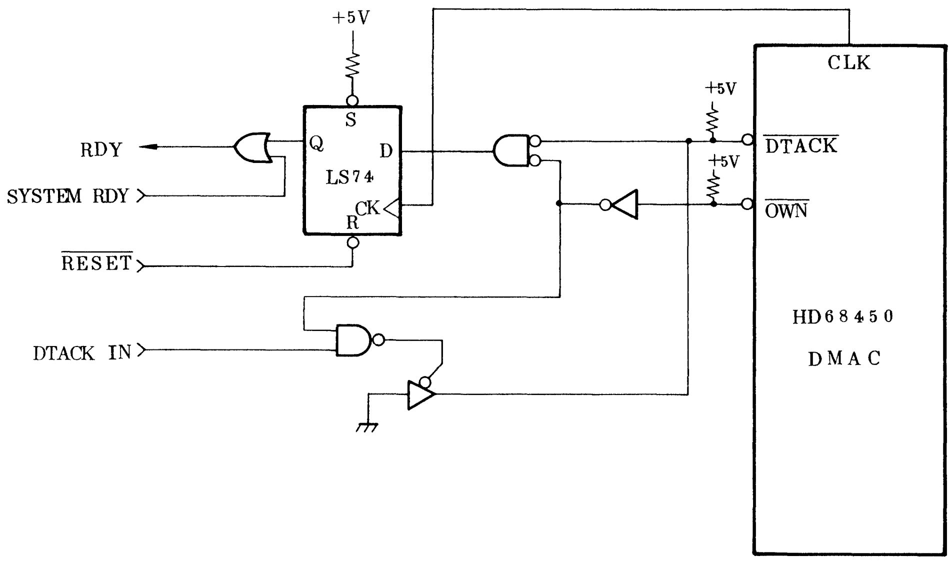


FIGURE 3.27 RDY Generator Circuit

4. **HD68450 DMAC Control Program**

4.1 **Basic Control Routine**

Figure 4.1 shows the flow chart for the DMAC control program by the MPU. The flow from START 1 sets the transfer mode on a channel and does the data transfer operation. Once the transfer mode is set, it is not necessary to set the mode again, as long as the data transfer is performed in the same mode. The data transfer for another data block in the same mode can be operated according to the flow from START 2.

The device address setting is necessary only for dual address mode (68000 and 6800 compatible devices). It is not necessary for devices with ACK, or ACK and READY.

Example 1 is of an HD68000 MPU program based on Figure 4.1. The DMAC's internal registers are mapped onto addresses from $1000 to $10FF. This program transfers 2000-word data from the I/O device to memory location from address $100000 and up.

When STR (START) bit in CCR is set, the DMAC sets ACT (Channel Active) bit in CSR, and at the same time resets STR bit in CCR automatically. After the internal initialization (operations like configuration error check, etc.), the DMAC can start the data transfer. REQ signals can be received by the DMAC while STR bit or ACT bit is set. Therefore, REQ signal can be accepted even during the internal initialization, but the data transfer for the request starts only after the initialization completion.

4.2 **Transfer Termination Routine**

When the DMAC completes a transfer operation, COC (Channel Operation Complete) bit in CSR is set. If an error occurs during the transfer, ERR bit is also set. The MPU can detect the DMA transfer completion by monitoring the COC bit. Figure 4.2 is the flow chart for transfer termination. If the MPU monitors COC bit set, the operation starts from START 1. This method requires many clock cycles because some MPU read cycles are associated. Instead, interrupt can be used to shorten the termination cycles. The DMAC issues interrupt request when COC bit is set, if INT (Interrupt enable) bit has been set. In order to enable the interrupt request, the 12th line instruction in Example 1 should be replaced by MOVE.B #$88, $1007.

In Example 1, since NIV (Normal Interrupt Vector) is set to $80, the MPU services the interrupt routine shown by vector number $80, unless error has occurred. For this routine, the program beginning from START 2 in Example 1 is applied. If error occurs, the MPU services the interrupt routine shown by vector number $81. The routine starting from START 2 in Figure 4.2 is used in this situation.

Error routines should be programmed case by case according to their applications. For bus error and address error, CER (Channel Error Register) can determine which address register caused the error, and the address where the error occurred is kept in the address register. CER also determines which of the transfer counters between MTC and BTC caused an error.

4.3 **Continue Mode Program Example**

Example 2 shows a program to start Continue Mode, setting the same operation modes as Example 1. The differences are to write information of the second data block (3000-word transfer to memory starting from $20000) in BAR, BTC, and BFC, and to set CNT (Continue) bit in CCR.

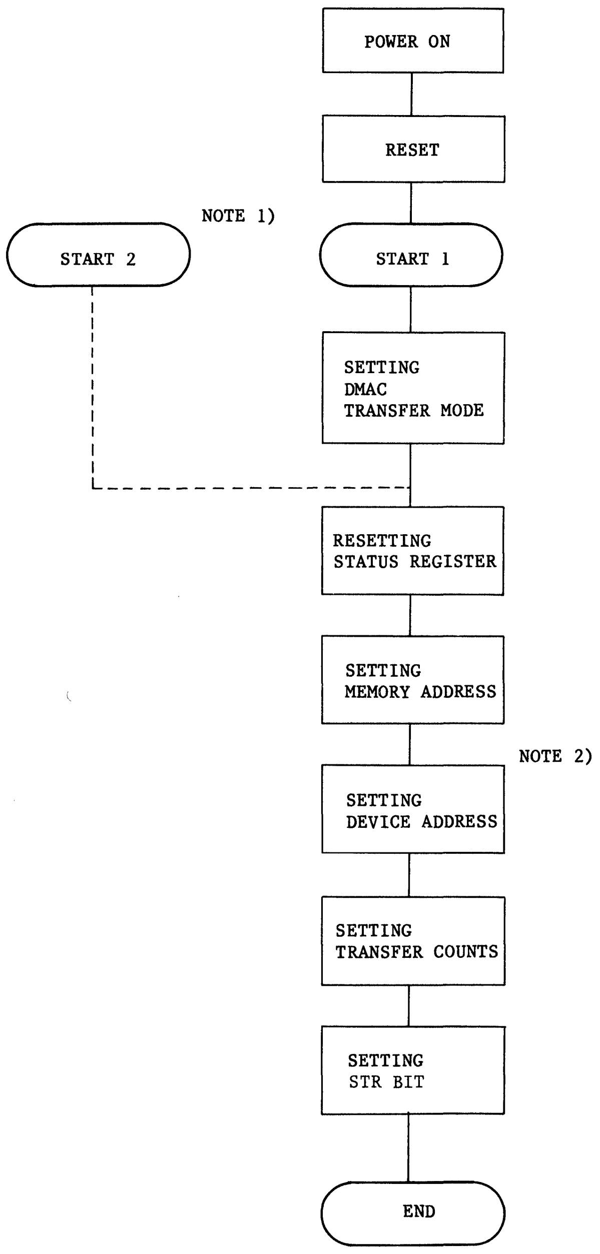
When CNT bit is set, the DMAC renews the transfer information of the first block which is specified by MAR, MTC, and MFC to that of the second block, at the end of the first block transfer by setting BTC (Block Transfer Complete) bit, and copying the data from BAR to MAR, BTC to MTC, and BFC to MFC. The DMAC resets CNT bit. If the INT (Interrupt) bit is set, it requests an interrupt to the MPU. If the DMAC receives transfer request, it starts the second block transfer.

To continue block transfer in this mode, it is necessary for the MPU to write the next block information in each base register, and to set CNT bit again by means of monitoring BTC bit, or receiving interrupt due to BTC. Figure 4.3 shows a flow chart for the routine executed by BTC interrupt. In this way the DMAC can transfer multi data blocks continuously in Continue Mode.

The multi block transfer in Continue Mode is usually done by Cycle Steal Mode because of the MPU access to the DMAC. Burst Mode or Auto Request Mode is also possible in Continue Mode if the number of blocks is two. When three or more blocks are transferred in Continue Mode, caution should be excercised, because an operation timing error will be caused if the MPU sets CNT bit after the completion of the second block transfer.

4.4 **A Program Example in Array Chaining Mode**

In Array Chaining Mode, the MPU forms an array table in memory which has memory addresses and transfer counts of the



NOTE 1) If the same transfer mode is used from START 1, the transfer mode setting can be skipped.
NOTE 2) Necessary only for Dual Address Mode.

FIGURE 4.1 Flow Chart of Control Program

Example 1: Basic Control Program

```
                                                                   Correspondance
Line number                                          Comment          to Fig.4.1
  1      START 1  EQU      *
  2               MOVE. W #$A892, $1004    setting DCR, OCR ⎫
  3               MOVE. B #$04, $1006         "    SCR      ⎪
  4               MOVE. B #$80, $1025         "    NIV      ⎬  Setting Transfer Mode
  5               MOVE. B #$81, $1027         "    EIV      ⎪
  6               MOVE. B #$01, $1029         "    MFC      ⎪
  7               CLR. B  $102D               "    CPR      ⎭
  8      START 2  EQU      *
  9               MOVE. B #$FF, $1000      resetting CSR       Resetting Status Register
 10               MOVE. L #100000, $100C   setting MAR         Setting Memory Address
 11               MOVE. W #2000, $100A        "    MTC         Setting Transfer Counts
 12               MOVE. B #$80, $1007      DMAC start routine  Setting STR bit
 13               RTS                      returning to
                                           main routine
```

(NOTE)
- The DMAC internal registers are mapped onto address $1000 through $10FF.
- Channel 0 is used.
- In Dual Addressing Mode, DAR and DFC should be set.

The DMAC transfer mode set in Example 1 is as follows:

- Cycle Steal Mode without Hold
- 16-bit I/O Device with $\overline{\text{ACK}}$
- $\overline{\text{PCL}}$ is the Status Input.
- Transfer from I/O Device to Memory
- Word data transfer
- No Chaining
- External Request through $\overline{\text{REQ}}$ pin
- Counts up Memory Address
- FC's output user data code (FC0=1, FC1=0, FC2=0)
- Channel Priority: 0 (the highest priority)

NOTE: This is necessary only in the case of PCL=ABORT. If PCL=ABORT, and the ABORT is inputted in the final bus cycle which is terminated by DONE signal from the I/O device, the ABORT signal is ignored and no error code is recorded in ERR bit, nor in CER. PCT bit should be monitored to determine the ABORT input.

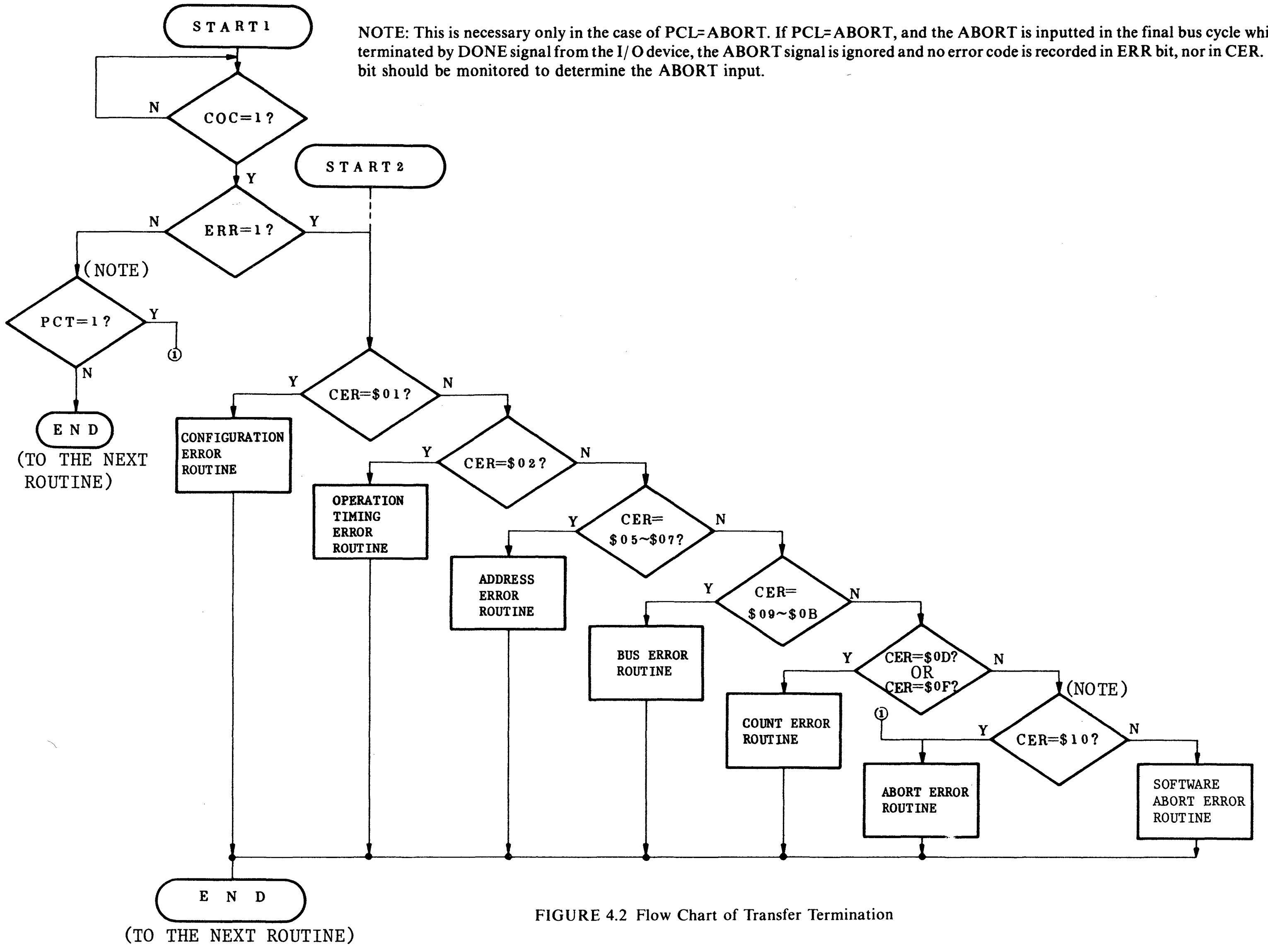


FIGURE 4.2 Flow Chart of Transfer Termination

Example 2: Start Program of Continue Mode

```
Line number                                                    Comment
    1        CONT     EQU        *
    2                 MOVE. W #$A892, $1004       setting DCR, OCR ⎤
    3                 MOVE. B #$04, $1006            "     SCR      ⎥ setting transfer
    4                 MOVE. B #$80, $1025            "     NIV      ⎬ mode
    5                 MOVE. B #$81, $1027            "     EIV      ⎥
    6                 CLR. B  $102D                  "     CPR      ⎦
    7                 MOVE. B #$FF, $1000         resetting CSR
    8                 MOVE. L #$100000, $100C     setting MAR      ⎤
    9                 MOVE. W #2000, $100A           "     MTC      ⎥ setting the 1st
   10                 MOVE. B #$01, $1029            "     MFC      ⎬ data block
   11                 MOVE. L #$200000, $101C        "     BAR      ⎦
   12                 MOVE. W #3000, $101A           "     BTC      ⎤ setting the 2nd
   13                 MOVE. B #$05, $1039            "     BFC      ⎦ data block
   14                 MOVE. B #$C8, $1007            "     STR, CNT, INT bits
   15                 RTS                         returning to main routine
```

(NOTE)
- The DMAC is mapped onto address $1000 through $10FF.
- Channel 0 is used.
- Modes are the same as those in Example 1.
- If the modes are already set, the lines from the 2nd through 6th are not necessary.
- The 1st data block is transferred to memory location from address $100000 plus 2000 words. In this case, FC0=1, FC1=0, and FC2=0 are outputted.
- The 2nd data block is transferred to memory location from address $200000 plus 3000 words. In this case, FC0=1, FC1=0, and FC2=1 are outputted.
- In Dual Addressing Mode, DAR and DFC should be set.

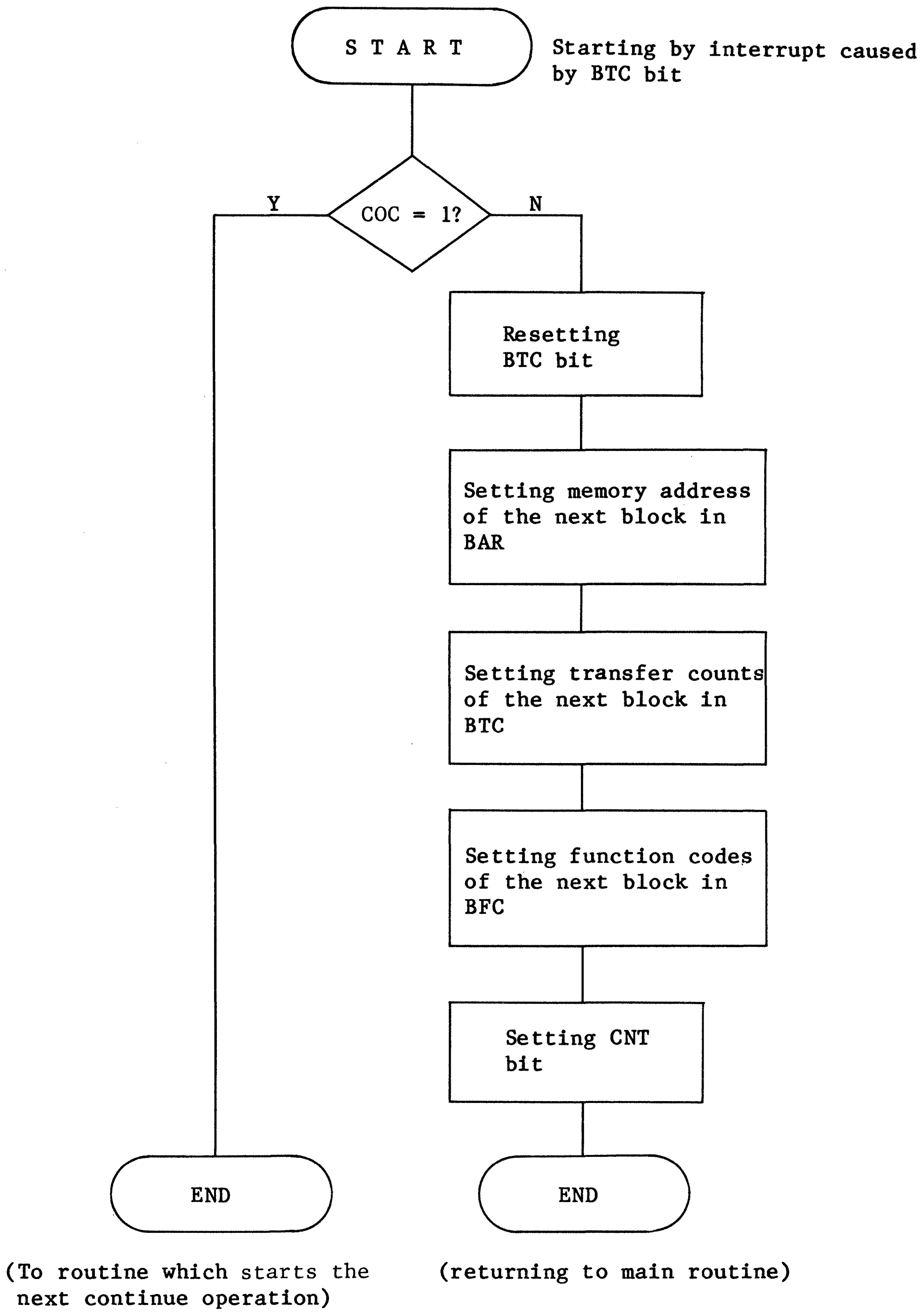

S T A R T
Starting by interrupt caused by BTC bit
COC = 1?
Y
N
Resetting BTC bit
Setting memory address of the next block in BAR
Setting transfer counts of the next block in BTC
Setting function codes of the next block in BFC
Setting CNT bit
END
END
(To routine which starts the next continue operation)
(returning to main routine)

FIGURE 4.3 Flow Chart of Continue Mode

multi blocks. The DMAC transfers the multi data blocks continuously by referring to the array table. The MPU does not have to access the DMAC in between block transfers in this mode.

The transfer example in Array Chaining Mode is shown in Figure 4.4. First, the MPU forms an array table for the multi block transfer. Second, it gives the device address, the number of blocks being transferred, and the top address of the array table to the DMAC's, DAR, BTC, and BAR, respectively. Third, the DMAC reads the memory address and the transfer count of the first block from the table into the DMAC's internal MAR and MTC, after the MPU sets STR bit in CCR. Fourth, the DMAC decrements the content of BTC (number of blocks) and starts the internal initialization process. Finally, the DMAC waits for a transfer request.

When the transfer of the first block is completed, the DMAC reads the second block information from the array table, renews MAR and MTC, and then transfers the second block. The DMAC repeats these chaining operations until BTC is exhausted (becomes zero). Example 3 is a program example for the transfer shown in Figure 4.4.

FCs (Function Codes) are not renewed in Array Chaining Mode. The contents in BFC are outputted when the DMAC reads the array table.

4.5 A Program Example in Linked Array Chaining Mode

Linked Array Chaining Mode is similar to Array Chaining Mode, but differs in the arrangement of the table for block transfer. In Array Chaining Mode, the array table must be prepared in contiguous space in memory, in the order of the block transfer. In Linked Array Chaining Mode, the array table does not have to be contiguous in memory block by block.

Figure 4.5 shows a transfer example in Linked Array Chaining Mode. The information of each block is linked with the linked address—i.e., the top address from where the information of the next block is stored. The information of each block can be distributed anywhere in memory by being linked with the linked address.

First, the MPU prepares for the linked array table in memory. Second, it gives the device address and the top address of the linked array table to the DMAC's DAR and BAR, respectively. Third, the DMAC reads the top address of the table designated by BAR and the memory address, and the transfer counts into MAR and MTC, respectively, after the MPU sets STR bit in CCR. Finally, the DMAC waits for a transfer request after initialization operation.

The DMAC transfers data blocks in the order of Block A, Block B, and Block C in Figure 4.5. When the DMAC reads "0" from linked address in the table, it terminates the chaining operation after the block transfer.

Example 4 is a program to execute the Linked Array Chaining in Figure 4.5. In this mode, BTC is not used. Contents of BFC are outputted when the DMAC refers to the table, but they are not renewed.

A linked array table is larger than an array table, but permits easier editing of the block transfer order. When a block is added or cancelled in the array table, the data in the table must be shifted. But in the linked array table, the editing is performed only by changing the linked address. For example, when Block B is cancelled in Figure 4.5, the linked address "X"(H)(L) is changed to the linked address "Y"(H)(L), and transfer counts "C" must be shifted to the location of the memory address "B"(H)(L) and transfer counts "B."

Example 3: Program Example in Array Chaining Mode (Corresponding to Fig. 4.4)

```
line number
     1      ARRAY   EQU      *
     2              MOVE. W #$A89A, $1004             setting OCR,DCR ┐
     3              MOVE. B #$04, $1006                  "    SCR     │
     4              MOVE. B #$80, $1025                  "    NIV     │
     5              MOVE. B #$81, $1027                  "    EIV     ├ setting transfer
     6              MOVE. B #$01, $1029                  "    MFC     │ mode
     7              MOVE. B #$05, $1039                  "    BFC     │
     8              CLR. B  $101D                        "    CPR     ┘
     9              MOVE. B #$FF, $1000               resetting CSR
    10              MOVE. L # table top address, $101C setting BAR
    11              MOVE. W #3, $101A                    "    BTC
    12              MOVE. B #$80, $1007                  "    STR bit
    13              RTS                               returning to main routine
```

(NOTE)
- The DMAC is mapped onto address from $1000 through $10FF.
- Channel 0 is used.
- The same modes are set as those in Example 1 except Array Chaining Mode.
- In Dual Addressing Mode, DAR and DFC should be set.

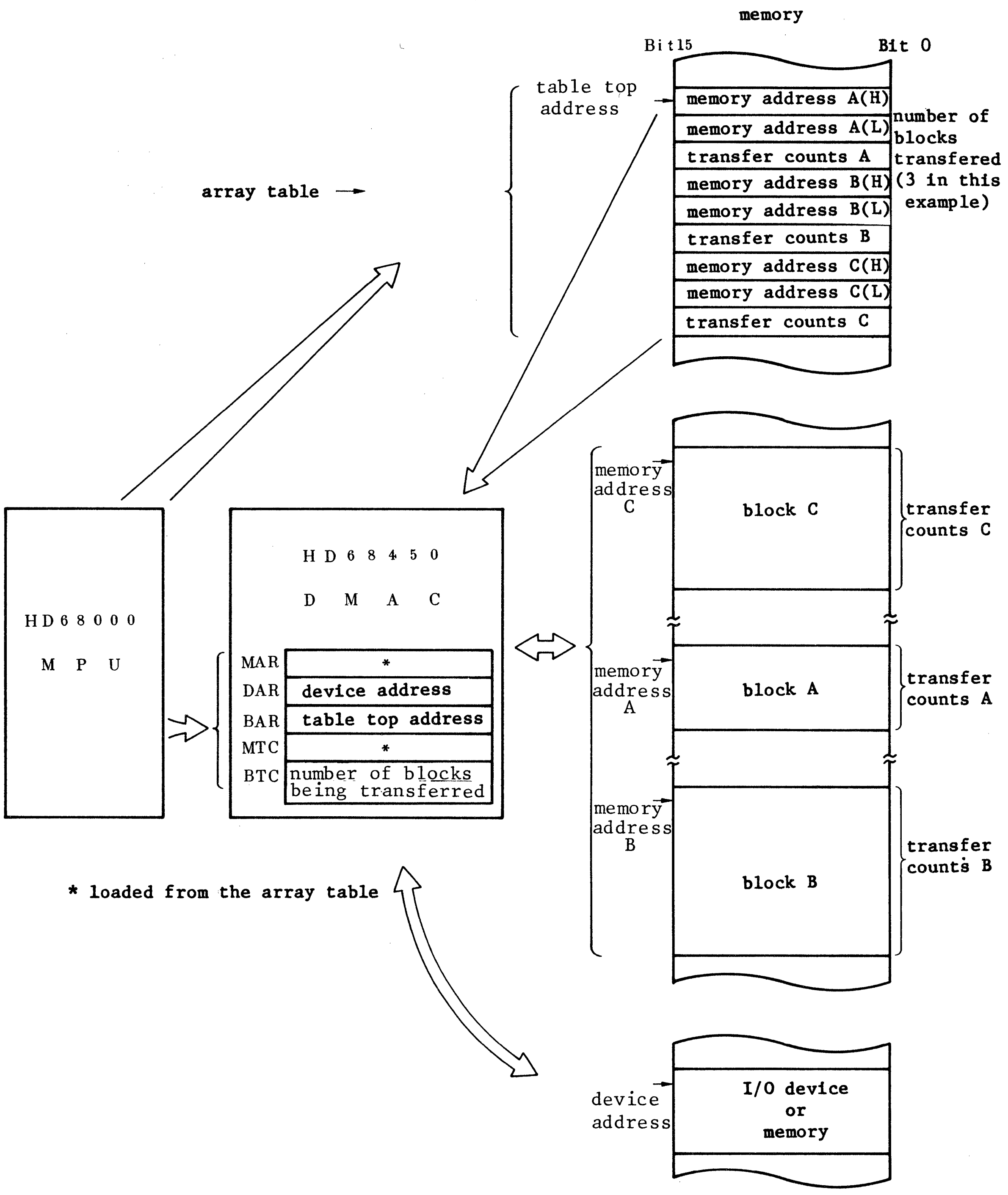


FIGURE 4.4 Transfer Example in Array Chaining Mode

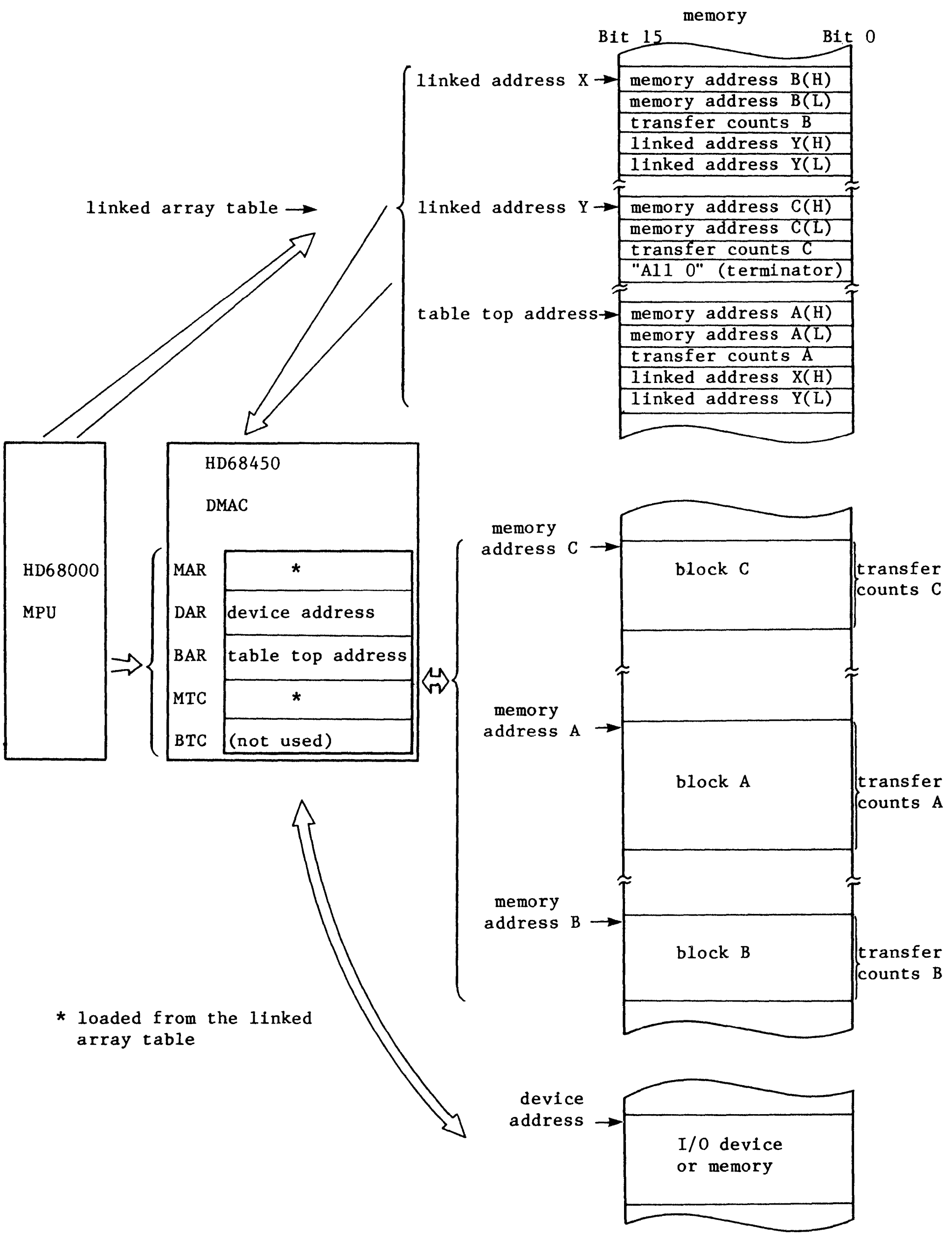


FIGURE 4.5 Linked Array Chaining Mode Transfer Examples

Example 4: Program Example in Linked Array Chaining Mode (corresponding to Fig. 4.5)

```
line number                                                   comment
    1     LINKA EQU      *
    2           MOVE. W  #$A89E, $1004                      setting OCR, DCR  ⎫
    3           MOVE. B  #$04, $1006                        setting SCR       ⎪
    4           MOVE. B  #$80, $1025                        setting NIV       ⎪
    5           MOVE. B  #$81, $1027                        setting EIV       ⎬ Setting Transfer
    6           MOVE. B  #$01, $1029                        setting MEC       ⎪ Mode
    7           MOVE. B  #$01, $1039                        setting BFC       ⎪
    8           CLR, B   $101D                              setting CPR       ⎭
    9           MOVE. B  #$FF, $1000                        resetting CSR
   10           MOVE. L  #table top address, $101C          setting BAR
   11           MOVE. B  #$80, $1007                        setting STR bit
   12           RTS                                         returning to
                                                            main routine
```

(NOTE)
- The DMAC is mapped onto address from $1000 through $10FF.
- Channel 0 is used.
- The same modes are set as those in example 1 except Linked Array Chaining Mode
- In Dual Addressing Mode, DAR and DFC should be set.

DATA SHEETS

HD68450, HD68450Y
DMAC (Direct Memory Access Controller)

APRIL 1984

Microprocessor implemented systems are becoming increasingly complex, particularly with the advent of high-performance 16-bit MPU devices with large memory addressing capability. In order to maintain high throughput, large blocks of data must be moved within these systems in a quick, efficient manner with minimum intervention by the MPU itself.

The HD68450 Direct Memory Access Controller (DMAC) is designed specifically to complement the performance and architectural capabilities of the HD68000 MPU by providing the following features:

- HMCS68000 Bus Compatible
- 4 independent DMA Channels
- Memory-to-Memory, Memory-to-Device, Device-to-Memory Transfers
- MMU Compatible
- Array-Chained and Linked-Array-Chained Operations
- On-Chip Registers that allow Complete Software Control by the System MPU
- Interface Lines for Requesting, Acknowledging, and Incidental Control of the Peripheral Devices
- Variable System Bus Bandwidth Utilization
- Programmable Channel Prioritization
- 2 Vectored interrupts for each Channel
- Auto-Request and External-Request Transfer Modes
- +5 Volt Operation

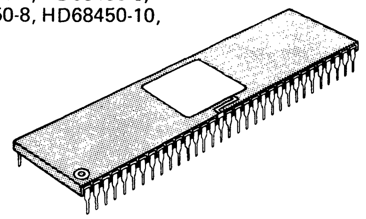
The DMAC functions by transferring a series of operands (data) between memory and peripheral device; operand sizes can be byte, word, or long word. A block is a sequence of operations; the number of operands in a block is determined by a transfer count. A single-channel operation may involve the transfer of several blocks of data between memory and device.

■ TYPE OF PRODUCTS

Type No.	Bus Timing	Packaging
HD68450-4	4MHz	DC-64
HD68450-6	6MHz	
HD68450-8	8MHz	
HD68450-10	10MHz	
HD68450Y4	4MHz	PGA-68
HD68450Y6	6MHz	
HD68450Y8	8MHz	
HD68450Y10	10MHz	

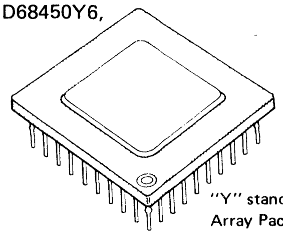
—The specifications for HD68450-10 and HD68450Y10 are preliminary.—

HD68450-4, HD68450-6,
HD68450-8, HD68450-10,



(DC-64)

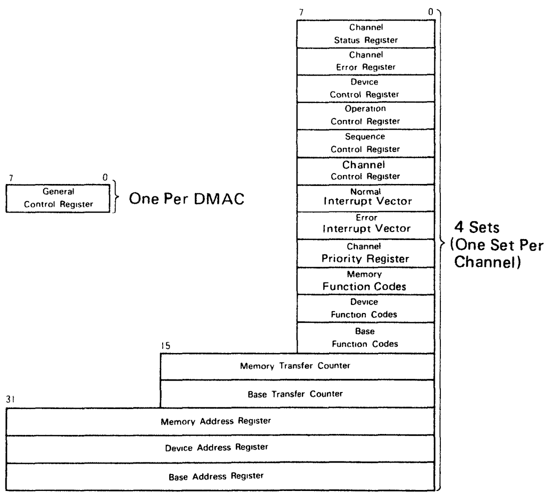
HD68450Y4, HD68450Y6,
HD68450Y8,
HD68450Y10,



"Y" stands for Pin Grid Array Package.

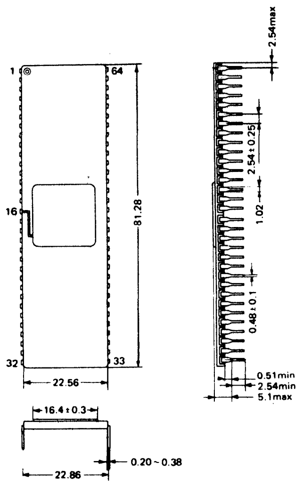
(PGA-68)

■ PROGRAMMING MODEL

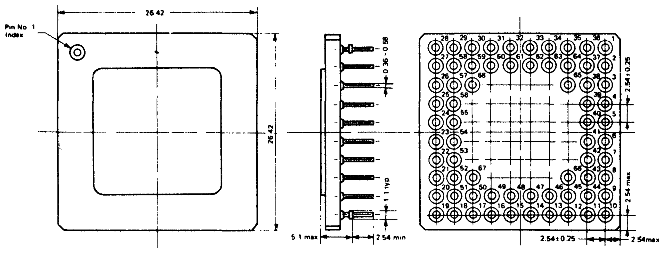


■ PACKAGING INFORMATION (Dimensions in mm)

● DC-64 (SIDE-BRAZED CERAMIC DIP)

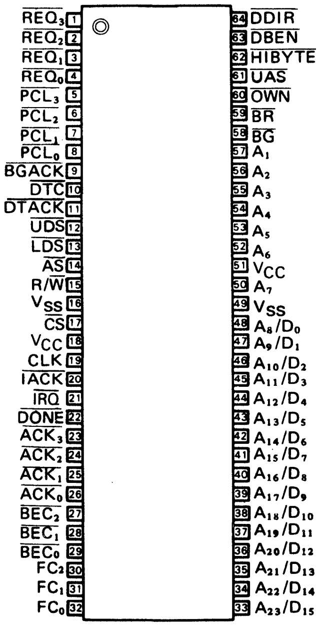


● PGA-68 (PIN GRID ARRAY PACKAGE)



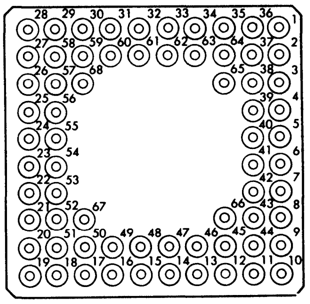
■ PIN ARRANGEMENT

● HD68450



(Top View)

● HD68450Y



(Bottom View)

Pin No.	Function	Pin No.	Function	Pin No.	Function	Pin No.	Function
1	N/C	18	$\overline{PCL_1}$	35	A_{19}/D_{11}	52	$\overline{BGACK}$
2	A_{13}/D_5	19	$\overline{DTACK}$	36	A_{17}/D_9	53	$\overline{LDS}$
3	A_{11}/D_3	20	$\overline{UDS}$	37	A_{15}/D_7	54	Vss
4	A_{10}/D_2	21	$\overline{AS}$	38	A_{12}/D_4	55	Vcc
5	A_8/D_0	22	$R/\overline{W}$	39	A_9/D_1	56	$\overline{DONE}$
6	A_7	23	N/C	40	V_{SS}	57	$\overline{IRQ}$
7	A_6	24	$\overline{CS}$	41	Vcc	58	$\overline{ACK_2}$
8	A_5	25	CLK	42	A_4	59	$\overline{BEC_2}$
9	A_3	26	$\overline{IACK}$	43	A_2	60	$\overline{BEC_0}$
10	N/C	27	$\overline{ACK_3}$	44	$\overline{BG}$	61	FC_0
11	$\overline{BR}$	28	$\overline{ACK_0}$	45	$\overline{OWN}$	62	A_{21}/D_{13}
12	$\overline{UAS}$	29	$\overline{BEC_1}$	46	$\overline{HIBYTE}$	63	A_{18}/D_{10}
13	$\overline{DBEN}$	30	FC_2	47	$\overline{DDIR}$	64	A_{16}/D_8
14	$\overline{REQ_3}$	31	FC_1	48	$\overline{REQ_1}$	65	A_{14}/D_6
15	$\overline{REQ_2}$	32	A_{23}/D_{15}	49	$\overline{PCL_2}$	66	A_1
16	$\overline{REQ_0}$	33	A_{22}/D_{14}	50	$\overline{PCL_0}$	67	$\overline{DTC}$
17	$\overline{PCL_3}$	34	A_{20}/D_{12}	51	N/C	68	$\overline{ACK_1}$

■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Value	Unit
Supply Voltage	V_{CC}*	-0.3 ~ +7.0	V
Input Voltage	V_{in}*	-0.3 ~ +7.0	V
Operating Temperature Range	T_{opr}	0 ~ +70	°C
Storage Temperature	T_{stg}	-55 ~ +150	°C

* With respect to V_{SS} (SYSTEM GND)

(NOTE) Permanent LSI damage may occur if maximum ratings are exceeded. Normal operation should be under recommended operating conditions. If these conditions are exceeded, it could affect reliability of LSI.

■ RECOMMENDED OPERATING CONDITIONS

Item	Symbol	min	typ	max	Unit
Supply Voltage	V_{CC}*	4.75	5.0	5.25	V
Input Voltage	V_{IH}*	2.0	–	V_{CC}	V
	V_{IL}*	-0.3	–	0.8	V
Operating Temperature	T_{opr}	0	25	70	°C

* With respect to V_{SS} (SYSTEM GND)

■ ELECTRICAL CHARACTERISTICS

● DC CHARACTERISTICS (V_{CC} = 5V ±5%, V_{SS} = 0V, Ta = 0 ~ +70°C, unless otherwise noted.)

Item		Symbol	Test Condition	min	typ	max	Unit
Input "High" Voltage		V_{IH}		2.0	–	V_{CC}	V
Input "Low" Voltage		V_{IL}		V_{SS}-0.3	–	0.8	V
Input Leakage Current	$\overline{CS}$, $\overline{IACK}$, $\overline{BG}$, CLK, $\overline{BEC_0}$ ~ $\overline{BEC_2}$ $\overline{REQ_0}$ ~ $\overline{REQ_3}$	I_{in}		–	–	10	μA
Three-State (Off State) Input Current	A_1 ~ A_7, D_0 ~ D_{15}/A_8 ~ A_{23}, $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, R/$\overline{W}$, $\overline{UAS}$, $\overline{DTACK}$, $\overline{BGACK}$, $\overline{OWN}$, $\overline{DTC}$, $\overline{HIBYTE}$, $\overline{DDIR}$, $\overline{DBEN}$, FC_0 ~ FC_2	I_{TSI}		–	–	10	μA
Open Drain (Off State) Input Current	$\overline{IREQ}$, $\overline{DONE}$	I_{ODI}		–	–	20	μA
Output "High" Voltage	A_1 ~ A_7, D_0 ~ D_{15}/A_8 ~ A_{23}, $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, R/$\overline{W}$, $\overline{UAS}$, $\overline{DTACK}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{OWN}$, $\overline{DTC}$, $\overline{HIBYTE}$, $\overline{DDIR}$, $\overline{DBEN}$, $\overline{ACK_0}$ ~ $\overline{ACK_3}$, $\overline{PCL_0}$ ~ $\overline{PCL_3}$, FC_0 ~ FC_2	V_{OH}	I_{OH} = -400μA	2.4	–	–	V
Output "Low" Voltage	A_1 ~ A_7, FC_0 ~ FC_2	V_{OL}	I_{OL} = 3.2 mA	–	–	0.5	V
	D_0 ~ D_{15}/A_8 ~ A_{23}, $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, R/$\overline{W}$, $\overline{DTACK}$, $\overline{BR}$, $\overline{OWN}$, $\overline{DTC}$, $\overline{HIBYTE}$, $\overline{DDIR}$, $\overline{DBEN}$, $\overline{ACK_0}$ ~ $\overline{ACK_3}$, $\overline{UAS}$, $\overline{PCL_0}$ ~ $\overline{PCL_3}$, $\overline{BGACK}$	V_{OL}	I_{OL} = 5.3 mA	–	–	0.5	V
	$\overline{IRQ}$, $\overline{DONE}$	V_{OL}	I_{OL} = 8.9 mA	–	–	0.5	V
Power Dissipation		P_D	f = 8 MHz, V_{CC} = 5.0 V Ta = 25°C	–	1.4	2.0	W
Capacitance		C_{in}	V_{in} = 0V, Ta = 25°C, f = 1 MHz	–	–	15	pF

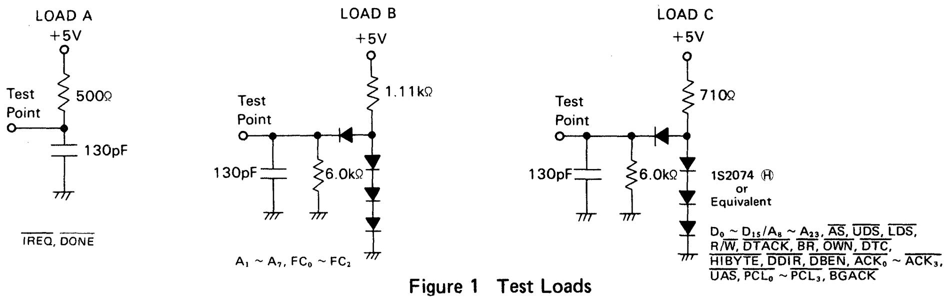

LOAD A
+5V
Test Point
500Ω
130pF
$\overline{IREQ}$, $\overline{DONE}$
LOAD B
+5V
1.11kΩ
Test Point
130pF
6.0kΩ
$A_1 \sim A_7$, $FC_0 \sim FC_2$
LOAD C
+5V
710Ω
Test Point
130pF
6.0kΩ
1S2074 Ⓗ or Equivalent
$D_0 \sim D_{15}/A_8 \sim A_{23}$, $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, $R/\overline{W}$, $\overline{DTACK}$, $\overline{BR}$, $\overline{OWN}$, $\overline{DTC}$, $\overline{HIBYTE}$, $\overline{DDIR}$, $\overline{DBEN}$, $\overline{ACK_0} \sim \overline{ACK_3}$, $\overline{UAS}$, $\overline{PCL_0} \sim \overline{PCL_3}$, $\overline{BGACK}$

Figure 1 Test Loads

• AC ELECTRICAL SPECIFICATIONS (V_{CC} = 5V ±5%, V_{SS} = 0V, Ta = 0~+70°C)

No.	Item	Symbol	Test Condition	4MHz HD68450-4 HD68450Y4		6MHz HD68450-6 HD68450Y6		8MHz HD68450-8 HD68450Y8		10MHz* HD68450-10 HD68450Y10		Unit
				min	max	min	max	min	max	min	max	
	Frequency of Operation	f		2	4	2	6	2	8	2	10	MHz
1	Clock Period	t_{cyc}		250	500	167	500	125	500	100	500	ns
2	Clock Width Low	t_{CL}		115	250	75	250	55	250	45	250	ns
3	Clock Width High	t_{CH}		115	250	75	250	55	250	45	250	ns
4	Clock Fall Time	t_{Cf}		–	10	–	10	–	10	–	10	ns
5	Clock Rise Time	t_{Cr}		–	10	–	10	–	10	–	10	ns
6	Asynchronous Input Setup Time	t_{ASI}		30	–	25	–	20	–	15	–	ns
7	Data in to $\overline{DBEN}$ Low	t_{DIDBL}		0	–	0	–	0	–	0	–	ns
8	$\overline{DTACK}$ Low to Data Invalid	t_{DTLDI}		0	–	0	–	0	–	0	–	ns
9	Address in to $\overline{AS}$ in Low	t_{AIASL}		0	–	0	–	0	–	0	–	ns
10	$\overline{AS}$, $\overline{DS}$ in High to Address in Invalid	t_{SIHAIV}		0	–	0	–	0	–	0	–	ns
11	Clock High to $\overline{DDIR}$ Low	t_{CHDRL}		–	90	–	80	–	70	–	60	ns
12	Clock High to $\overline{DDIR}$ High	t_{CHDRH}		–	90	–	80	–	70	–	60	ns
13	$\overline{DS}$ in High to $\overline{DDIR}$ High Impedance	t_{DSHDRZ}		–	160	–	140	–	120	–	110	ns
14	Clock Low to $\overline{DBEN}$ Low	t_{CLDBL}		–	90	–	80	–	70	–	60	ns
15	Clock Low to $\overline{DBEN}$ High	t_{CLDBH}		–	90	–	80	–	70	–	60	ns
16	$\overline{DS}$ in High to $\overline{DBEN}$ High Impedance	t_{DSHDBZ}		–	160	–	140	–	120	–	110	ns
17	Clock High to Data Out Valid (MPU read)	t_{CHDVM}		–	290	–	230	–	180	–	160	ns
18	$\overline{DS}$ in High to Data Out Invalid	t_{DSHDZn}		0	–	0	–	0	–	0	–	ns
19	$\overline{DS}$ in High to Data High Impedance	t_{DSHDZ}		–	160	–	140	–	120	–	110	ns
20	Clock Low to $\overline{DTACK}$ Low	t_{CLDTL}		–	90	–	80	–	70	–	60	ns
21	$\overline{DS}$ in High to $\overline{DTACK}$ High	t_{DSHDTH}		–	160	–	130	–	110	–	110	ns
22	$\overline{DTACK}$ Width High	t_{DTH}		10	–	10	–	10	–	10	–	ns
23	$\overline{DS}$ in High to $\overline{DTACK}$ High Impedance	t_{DSHDTZ}		–	220	–	200	–	180	–	160	ns
24	$\overline{DTACK}$ Low to $\overline{DS}$ in High	t_{DTLDSH}		0	–	0	–	0	–	0	–	ns
25	$\overline{REQ}$ Width Low	t_{REQL}		2.0	–	2.0	–	2.0	–	2.0	–	clk. per.
26	$\overline{REQ}$ Low to $\overline{BR}$ Low	t_{RELBRL}		500	–	334	–	250	–	200	–	ns
27	Clock High to $\overline{BR}$ Low	t_{CHBRL}	Fig. 1 ~ Fig. 8	–	90	–	80	–	70	–	60	ns
28	Clock High to $\overline{BR}$ High	t_{CHBRH}		–	90	–	80	–	70	–	60	ns
29	$\overline{BG}$ Low to $\overline{BGACK}$ Low	t_{BGLBL}		4.5	–	4.5	–	4.5	–	4.5	–	clk. per.
30	$\overline{BR}$ Low to MPU Cycle End ($\overline{AS}$ in High)	t_{BRLASH}		0	–	0	–	0	–	0	–	ns
31	MPU Cycle End ($\overline{AS}$ in High) to $\overline{BGACK}$ Low	t_{ASHBL}		4.5	5.5	4.5	5.5	4.5	5.5	4.5	5.5	clk. per.
32	$\overline{REQ}$ Low to $\overline{BGACK}$ Low	t_{REQLBL}		12.0	–	12.0	–	12.0	–	12.0	–	clk. per.
33	Clock High to $\overline{BGACK}$ High	t_{CHBL}		–	90	–	80	–	70	–	60	ns
34	Clock High to $\overline{BGACK}$ High	t_{CHBH}		–	90	–	80	–	70	–	60	ns
35	Clock Low to $\overline{BGACK}$ High Impedance	t_{CLBZ}		–	120	–	100	–	80	–	70	ns
36	Clock High to FC Valid	t_{CHFCV}		–	140	–	120	–	100	–	90	ns
37	Clock High to Address Valid	t_{CHAV}		–	160	–	140	–	120	–	110	ns
38	Clock High to Address/FC/Data High Impedance	t_{CHAZx}		–	140	–	120	–	100	–	100	ns
39	Clock High to Address/FC/Data Invalid	t_{CHAZn}		0	–	0	–	0	–	0	–	ns
40	Clock Low to Address High Impedance	t_{CLAZ}		–	140	–	120	–	100	–	90	ns
41	Clock High to $\overline{UAS}$ Low	t_{CHUL}		–	90	–	80	–	70	–	60	ns
42	Clock High to $\overline{UAS}$ High	t_{CHUH}		–	90	–	80	–	70	–	60	ns
43	Clock Low to $\overline{UAS}$ High Impedance	t_{CLUZ}		–	120	–	100	–	80	–	70	ns
44	$\overline{UAS}$ High to Address Invalid	t_{UHAI}		50	–	40	–	30	–	20	–	ns
45	Clock High to $\overline{AS}$, $\overline{DS}$ Low	t_{CHSL}		–	80	–	70	–	60	–	55	ns
46	Clock Low to $\overline{DS}$ Low (write)	t_{CLDSL}		–	80	–	70	–	60	–	55	ns
47	Clock Low to $\overline{AS}$, $\overline{DS}$ High	t_{CLSH}		–	90	–	80	–	70	–	60	ns
48	Clock Low to $\overline{AS}$, $\overline{DS}$ High Impedance	t_{CLSZ}		–	120	–	100	–	80	–	70	ns
49	$\overline{AS}$ Width Low	t_{ASL}		545	–	350	–	255	–	195	–	ns
50	$\overline{DS}$ Width Low	t_{DSL}		420	–	265	–	190	–	145	–	ns
51	$\overline{AS}$, $\overline{DS}$ Width High	t_{SH}		285	–	180	–	150	–	105	–	ns
52	Address/FC Valid to $\overline{AS}$, $\overline{DS}$ Low	t_{AVSL}		50	–	40	–	30	–	20	–	ns
53	$\overline{AS}$, $\overline{DS}$ High to Address/FC/Data Invalid	t_{SHAZ}		50	–	40	–	30	–	20	–	ns
54	Clock High to $R/\overline{W}$ Low	t_{CHRL}		–	90	–	80	–	70	–	60	ns
55	Clock High to $R/\overline{W}$ High	t_{CHRH}		–	90	–	80	–	70	–	60	ns

* Preliminary

(continued)

No.	Item	Symbol	Test Condition	4MHz HD68450-4 HD68450Y4		6MHz HD68450-6 HD68450Y6		8MHz HD68450-8 HD68450Y8		10MHz* HD68450-10 HD68450Y10		Unit
				min	max	min	max	min	max	min	max	
56	Clock Low to R/$\overline{\text{W}}$ High Impedance	t_{CLRZ}		–	120	–	100	–	80	–	70	ns
57	Address/FC Valid to R/$\overline{\text{W}}$ Low	t_{AVRL}		100	–	40	–	20	–	10	–	ns
58	R/$\overline{\text{W}}$ Low to $\overline{\text{DS}}$ Low (write)	t_{RLSL}		285	–	170	–	120	–	90	–	ns
59	$\overline{\text{DS}}$ High to R/$\overline{\text{W}}$ High	t_{SHRH}		60	–	50	–	40	–	20	–	ns
60	Clock Low to $\overline{\text{OWN}}$ Low	t_{CLOL}		–	90	–	80	–	70	–	60	ns
61	Clock Low to $\overline{\text{OWN}}$ High	t_{CLOH}		–	90	–	80	–	70	–	60	ns
62	Clock High to $\overline{\text{OWN}}$ High Impedance	t_{CHOZ}		–	120	–	100	–	80	–	70	ns
63	$\overline{\text{OWN}}$ Low to $\overline{\text{BGACK}}$ Low	t_{OLBL}		50	–	40	–	30	–	20	–	ns
64	$\overline{\text{BGACK}}$ High to $\overline{\text{OWN}}$ High	t_{BHOH}		50	–	40	–	30	–	20	–	ns
65	$\overline{\text{OWN}}$ Low to $\overline{\text{UAS}}$ Low	t_{OLUL}		50	–	40	–	30	–	20	–	ns
66	Clock High to $\overline{\text{ACK}}$ Low	t_{CHACL}		–	90	–	80	–	70	–	60	ns
67	Clock Low to $\overline{\text{ACK}}$ Low	t_{CLACL}		–	90	–	80	–	70	–	60	ns
68	Clock High to $\overline{\text{ACK}}$ High	t_{CHACH}		–	90	–	80	–	70	–	60	ns
69	$\overline{\text{ACK}}$ Low to $\overline{\text{DS}}$ Low	t_{ACLDSL}		230	–	140	–	100	–	80	–	ns
70	$\overline{\text{DS}}$ High to $\overline{\text{ACK}}$ High	t_{DSHACH}		50	–	40	–	30	–	20	–	ns
71	Clock High to $\overline{\text{HIBYTE}}$ Low	t_{CHHIL}		–	90	–	80	–	70	–	60	ns
72	Clock Low to $\overline{\text{HIBYTE}}$ Low	t_{CLHIL}		–	90	–	80	–	70	–	60	ns
73	Clock High to $\overline{\text{HIBYTE}}$ High	t_{CHHIH}		–	90	–	80	–	70	–	60	ns
74	Clock Low to $\overline{\text{HIBYTE}}$ High Impedance	t_{CLHIZ}		–	120	–	100	–	80	–	70	ns
75	Clock High to $\overline{\text{DTC}}$ Low	t_{CHDTL}		–	90	–	80	–	70	–	60	ns
76	Clock High to $\overline{\text{DTC}}$ High	t_{CHDTH}	Fig. 1 ~ Fig. 8	–	90	–	80	–	70	–	60	ns
77	Clock Low to $\overline{\text{DTC}}$ High Impedance	t_{CLDTZ}		–	120	–	100	–	80	–	70	ns
78	$\overline{\text{DTC}}$ Width Low	t_{DTCL}		230	–	147	–	105	–	80	–	ns
79	$\overline{\text{DTC}}$ Low to $\overline{\text{DS}}$ High	t_{DTLDH}		95	–	50	–	30	–	20	–	ns
80	Clock High to $\overline{\text{DONE}}$ Low	t_{CHDOL}		–	90	–	80	–	70	–	60	ns
81	Clock Low to $\overline{\text{DONE}}$ Low	t_{CLDOL}		–	90	–	80	–	70	–	60	ns
82	Clock High to $\overline{\text{DONE}}$ High	t_{CHDOH}		–	150	–	140	–	130	–	120	ns
83	Clock Low to $\overline{\text{DDIR}}$ High Impedance	t_{CLDRZ}		–	120	–	100	–	80	–	70	ns
84	Clock Low to $\overline{\text{DBEN}}$ High Impedance	t_{CLDBZ}		–	120	–	100	–	80	–	70	ns
85	$\overline{\text{DDIR}}$ Low to $\overline{\text{DBEN}}$ Low	t_{DRLDBL}		50	–	40	–	30	–	20	–	ns
86	$\overline{\text{DBEN}}$ High to $\overline{\text{DDIR}}$ High	t_{DBHDRH}		50	–	40	–	30	–	20	–	ns
87	$\overline{\text{DBEN}}$ Low to Address/Data High Impedance	t_{DBLAZ}		–	17	–	17	–	17	–	17	ns
88	Clock Low to $\overline{\text{PCL}}$ Low (1/8 clock)	t_{CLPL}		–	90	–	80	–	70	–	60	ns
89	Clock Low to $\overline{\text{PCL}}$ High (1/8 clock)	t_{CLPH}		–	90	–	80	–	70	–	60	ns
90	$\overline{\text{PCL}}$ Width Low (1/8 clock)	t_{PCLL}		4.0	–	4.0	–	4.0	–	4.0	–	clk. per.
91	$\overline{\text{DTACK}}$ Low to Data In (setup time)	t_{DALDI}		–	320	–	200	–	150	–	115	ns
92	$\overline{\text{DS}}$ High to Data Invalid (hold time)	t_{SHDI}		0	–	0	–	0	–	0	–	ns
93	$\overline{\text{DS}}$ High to $\overline{\text{DTACK}}$ High	t_{SHDAH}		0	240	0	160	0	120	0	90	ns
94	Data Out Valid to $\overline{\text{DS}}$ Low	t_{DOSL}		0	–	0	–	0	–	0	–	ns
95	Data In to Clock Low (setup time)	t_{DICL}		30	–	25	–	15	–	15	–	ns
96	$\overline{\text{BEC}}$ Low to $\overline{\text{DTACK}}$ Low	t_{BECDAL}		50	–	50	–	50	–	50	–	ns
97	$\overline{\text{BEC}}$ Width Low	t_{BECL}		2.0	–	2.0	–	2.0	–	2.0	–	clk. per.
98	Clock High to $\overline{\text{IRQ}}$ Low	t_{CHIRL}		–	90	–	80	–	70	–	60	ns
99	Clock High to $\overline{\text{IRQ}}$ High	t_{CHIRH}		–	150	–	140	–	130	–	120	ns
100	$\overline{\text{READY}}$ In to $\overline{\text{DTC}}$ Low (Read)	t_{RALDTL}		270	–	180	–	145	–	120	–	ns
101	$\overline{\text{READY}}$ In to $\overline{\text{DS}}$ Low (Write)	t_{RALDSL}		395	–	240	–	205	–	170	–	ns
102	$\overline{\text{DS}}$ High to $\overline{\text{READY}}$ High	t_{DSHRAH}		0	240	0	160	0	120	0	90	ns
103	$\overline{\text{DONE}}$ In Low to $\overline{\text{DTACK}}$ Low	t_{DOLDAL}		50	–	50	–	50	–	50	–	ns
104	$\overline{\text{DS}}$ High to $\overline{\text{DONE}}$ In High	t_{DSHDOH}		0	240	0	160	0	120	0	90	ns
105	Asynchronous Input Hold Time	t_{ASIH}		15	–	15	–	15	–	15	–	ns

* Preliminary

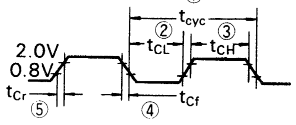
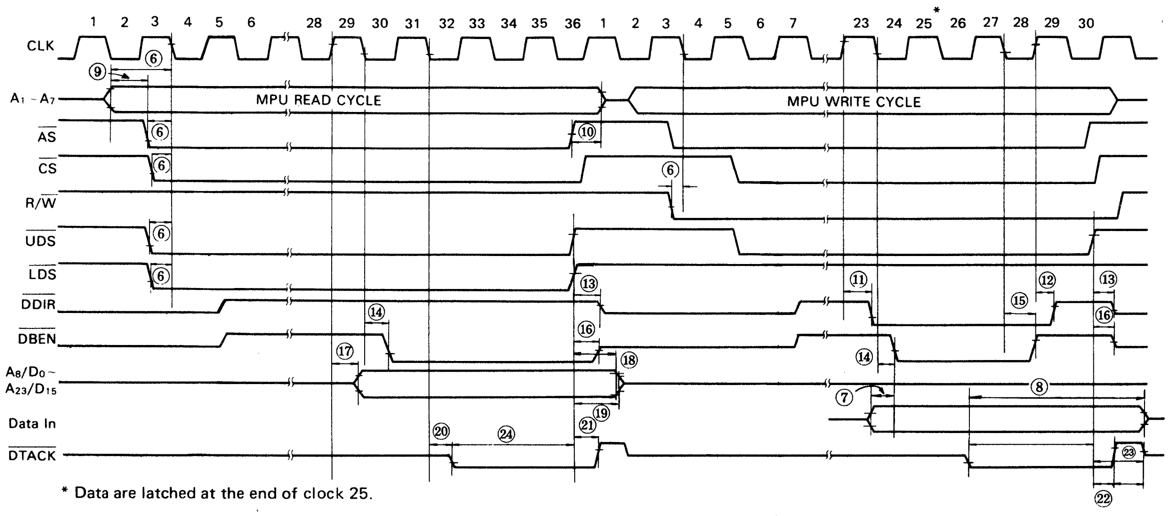
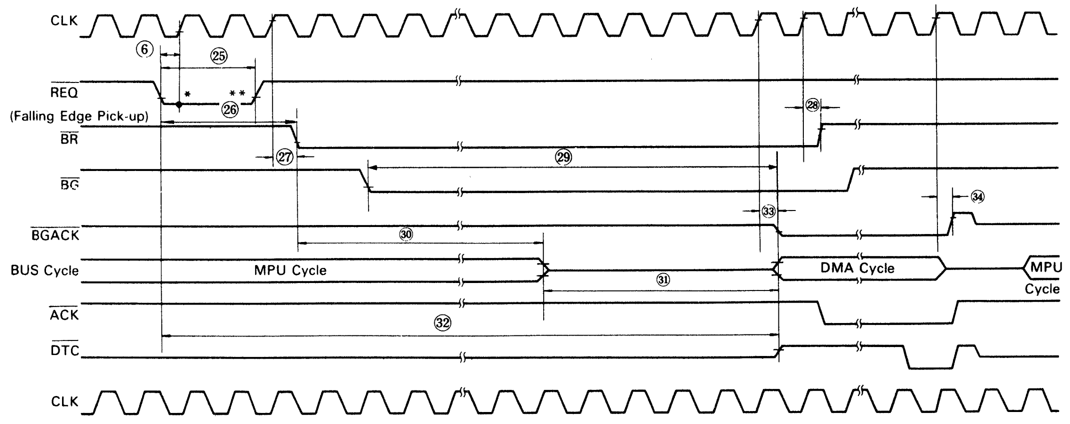


Figure 2 Input Clock Waveform



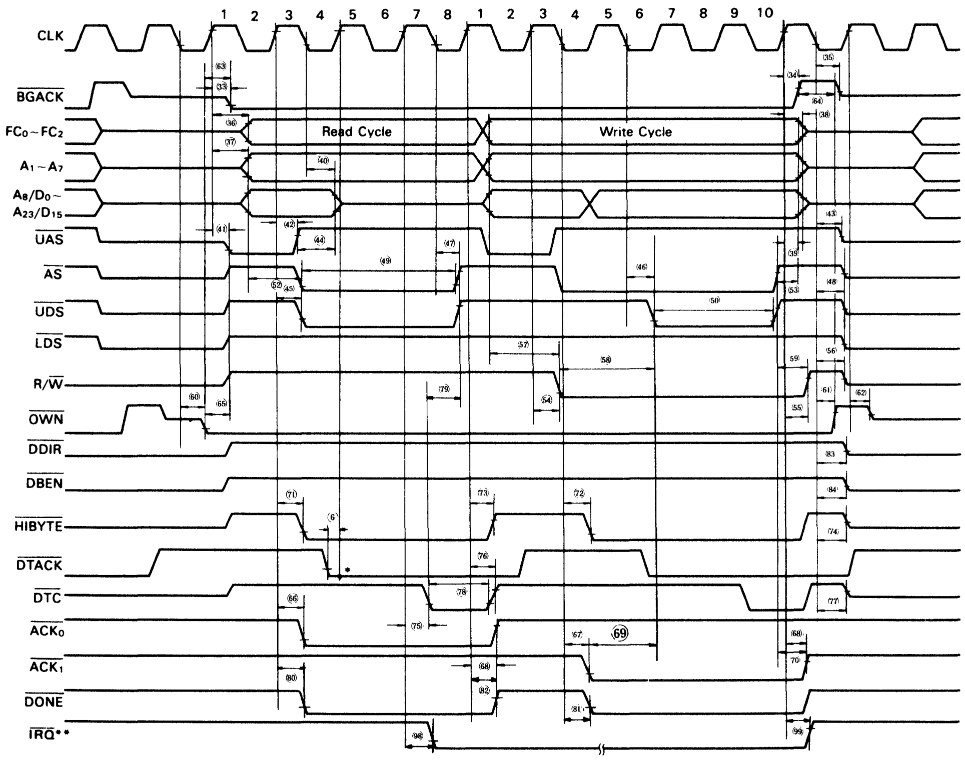
* Data are latched at the end of clock 25.

Figure 3 AC Electrical Waveforms — MPU Read/Write



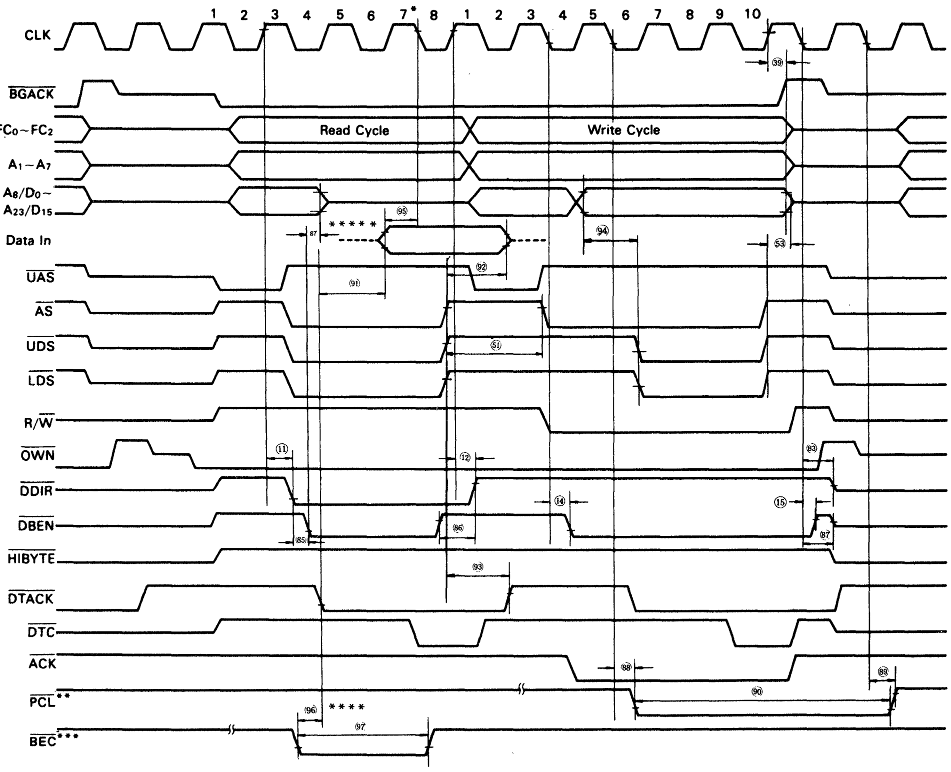
* REQ is sampled at the rising edge of CLK in cycle steal and Burst modes.
** BR isn't asserted while a BEC exception condition exists or DMAC is accessed by MPU.

Figure 4 AC Electrical Waveforms — Bus Arbitration



* $\overline{DTACK}$ is sampled at the rising edge of CLK. This is different from HD68000.
** This timing is not related to DMA Read/Write (Single Cycle) sequence.

Figure 5 AC Electrical Waveforms – DMA Read/Write (Single Cycle)



*Data is latched at the end of clock 7. This timing is the same as HD68000.

**This timing is not related to DMA Read/Write (Dual Cycle) sequence. This timing is only applicable when 1/8 clock pulse mode is selected.

***This timing is applicable when a bus exception occurs.

****If #6 is satisifed for both $\overline{\text{DTACK}}$ and $\overline{\text{BEC}}$, #96 may be Ons.

*****If the propagation delay of the external bidirectional buffer LS245 is less than 17nsec, a conflict may occur between the address output of the DMAC and the system data bus. In this case, the output of $\overline{\text{DBEN}}$ must be delayed externally.

Figure 6 AC Electrical Waveforms – DMA Read/Write (Dual Cycle)

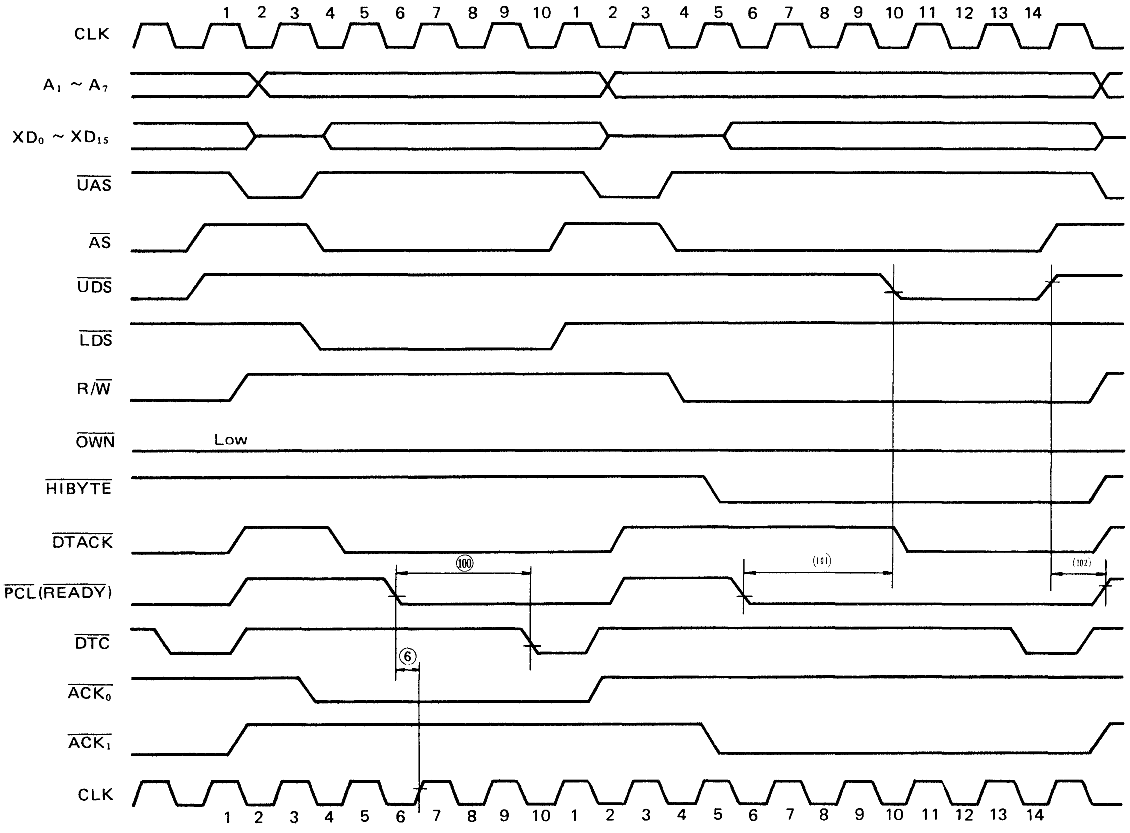
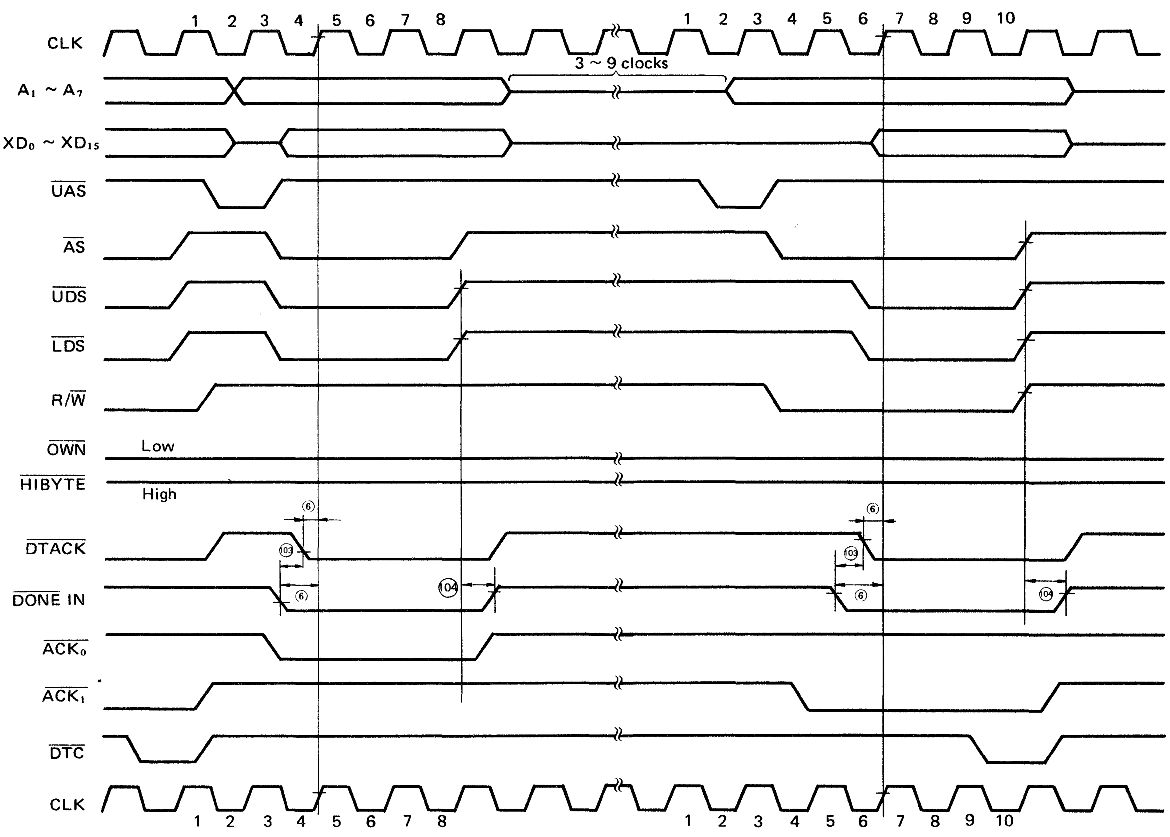


Figure 7 AC Electrical Waveforms — DMA Read/Write (Single Cycle with $\overline{PCL}$)



* If #6 is satisfied for both $\overline{DTACK}$ and $\overline{DONE}$, #103 may be 0ns.

Figure 8 AC Electrical Waveforms – $\overline{DONE}$ Input

(NOTES for Figure 3 through 8)

1) Setup time for the asynchronous inputs $\overline{BG}$, $\overline{BGACK}$, $\overline{CS}$, $\overline{IACK}$, $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, and $R/\overline{W}$ guarantees their recognition at the next falling edge of the clock. Setup time for $\overline{BEC_0} \sim \overline{BEC_2}$, $\overline{REQ_0} \sim \overline{REQ_3}$, $\overline{PCL_0} \sim \overline{PCL_3}$, $\overline{DTACK}$, and $\overline{DONE}$ guarantees their recognition at the next rising edge of the clock.
2) Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts.
3) These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.

■ SIGNAL DESCRIPTION

The following section identifies the signals used in the DMAC. In the definitions, "MPU mode" refers to the state when the DMAC is chip selected by MPU. The term "DMA mode" refers to the state when the DMAC assumes ownership of the bus. The DMAC is in the "IDLE mode" at all other times. Moreover, the DMA bus cycle refers to the bus cycle that is executed by the DMAC in the "DMA mode".

NOTE) In this data sheet, the state of the signals is described with these words: active or assert, inactive or negate.

This is done to avoid confusion when dealing with a mixture of "active-low" and "active-high" signals. The term assert or assertion is used to indicate that a signal is active or true independent of whether that voltage is low or high. The term negate or negation is used to indicate that a signal is inactive or false.

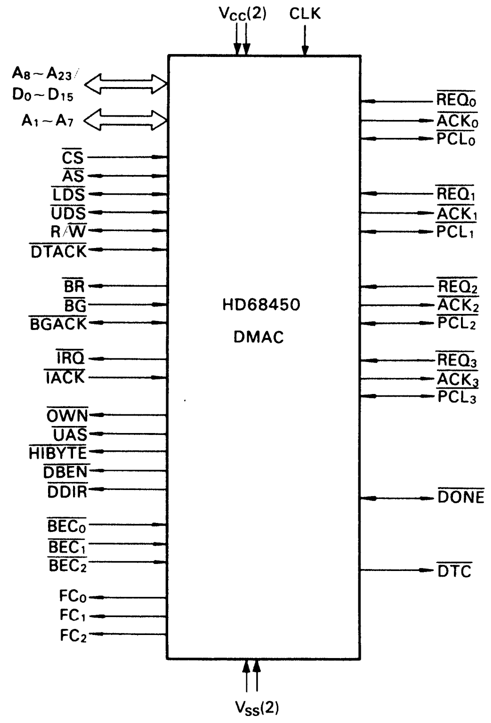


Figure 9 Input and Output Signals

● Address/Data Bus (A_8/D_0 through A_{23}/D_{15})

Input/Output	Three-statable
Active-high	

These lines are time multiplexed for the address and data bus. The lines $\overline{DDIR}$, $\overline{DBEN}$, $\overline{UAS}$ and $\overline{OWN}$ are used to control the demultiplexing of the data and address lines externally. Demultiplexing is explained in a later section. The bi-directional data bus is used to transfer data between DMAC, MPU, memory and I/O devices.

Address lines are outputs to address memory and I/O devices.

● Address Bus (A_1 through A_7)

Input/Output	Three-statable
Active-high	

In the MPU mode, the DMAC internal registers are accessed with these lines and $\overline{LDS}$, $\overline{UDS}$. The address map for these registers is shown in Table 1. During a DMA bus cycle, A_1-A_7 are outputs containing the low order address bits of the location being accessed.

● Function Code (FC_0 through FC_2)

Output	Three-statable
Active-high	

These output signals provide the function codes during DMA bus cycles. They are three-stated except in the DMA bus cycles. They are used to control the HMCS68000 memories.

● Clock (CLK)

Input

This is the input clock to the HD68450, and should never be terminated at any time. This clock can be different from the MPU clock since HD68450 operates completely asynchronously.

● Chip Select ($\overline{CS}$)

Input
Active low

This input signal is used to chip select the DMAC in "MPU" mode. If the $\overline{CS}$ input is asserted during a bus cycle which is generated by the DMAC, the DMAC internally terminates the bus cycle and signals an address error. This function protects the DMAC from accessing its own register.

● Address Strobe ($\overline{AS}$)

Input/Output	Three-statable
Active low	

In the "MPU mode," this line is an input indicating valid address input, and during the DMA bus cycle it is an output indicating a valid address output from the DMAC on the address bus.

The DMAC monitors these input lines during bus arbitration to determine the completion of the bus cycle by the MPU or other bus masters.

● Upper Address Strobe ($\overline{UAS}$)

Output	Three-statable
Active low	

This line is an output to latch the upper address lines on the multiplexed data/address lines. It is three-stated except in the "DMA mode".

● Own ($\overline{OWN}$)

Output	Three-statable
Active low	

This line is asserted by the DMAC during DMA mode, and is used to control the output of the address line latch. This line may also be used to control the direction of bi-directional buffers when loads on $\overline{AS}$, $\overline{LDS}$, $\overline{UDS}$, R/$\overline{W}$ and other signals exceed the drive capability. It is three-stated in the "MPU mode" and the "IDLE mode"

- **Data Direction ($\overline{DDIR}$)**

Outputs	Three-statable
Active low	(when data direction is input to the DMAC)
Active high	(when the data direction is output from the DMAC)

This line controls the direction of data through the bidirectional buffer which used to demultiplex the data/address lines. It is three-stated during the "IDLE mode"

- **Data Bus Enable ($\overline{DBEN}$)**

Output	Three-statable
Active low	

This line controls the output enable line of bidirectional buffers on the multiplexed data/address lines. It is a three-stated during the "IDLE mode".

- **High Byte ($\overline{HIBYTE}$)**

Output	Three-statable
Active low	

This line is used when the operand size is a byte in the single addressing mode. It is asserted when data is present on the upper eight bits of the data bus. It is used to control the output of the bidirectional buffers which connect the upper eight bits of the data bus with the lower eight bits. It is three-stated during the "MPU mode" and the "IDLE mode."

- **Read/Write (R/$\overline{W}$)**

Input/Output	Three-statable
Active low (write)	
Active high (read)	

This line is an input in the "MPU mode" and an output during the "DMA mode". It is three-stated during the "IDLE mode". It is used to control the direction of data flow.

- **Upper Data Strobe ($\overline{UDS}$), Lower Data Strobe ($\overline{LDS}$)**

Input/Output	Three-statable
Active low	

These lines are extensions of the address lines indicating which byte or bytes of data of the addressed word are being addressed. These lines combined corresponds to address line A_0 in table 1.

- **Data Transfer Acknowledge ($\overline{DTACK}$)**

Input/Output	Three-statable
Active low	

In the "MPU mode", this line is an output indicating the completion of Read/Write bus cycle by the MPU.

In the "DMA mode", the DMAC monitors this line to determine when a data transfer has completed. In the event that a bus exception is requested, except for $\overline{HALT}$, prior to or concurrent with $\overline{DTACK}$, the $\overline{DTACK}$ response is ignored and the bus exception is honored. In the "IDLE mode", this signal is three-stated.

- **Bus Exception Controls ($\overline{BEC_0}$ through $\overline{BEC_2}$)**

Input
Active low

These lines provide an encoded signal input indicating an exceptional condition in the DMA bus cycle. See bus exception section for details.

- **Bus Request ($\overline{BR}$)**

Output
Active low

This output line is used to request ownership of the bus by the DMAC.

- **Bus Grant ($\overline{BG}$)**

Input
Active low

This line is used to indicate to the DMAC that it is to be the next bus master. The DMAC cannot assume bus ownership until both $\overline{AS}$ and $\overline{BGACK}$ becomes inactive. Once the DMAC acquires the bus, it does not continue to monitor the $\overline{BG}$ input.

- **Bus Grant Acknowledge ($\overline{BGACK}$)**

Input/Output	Three-statable
Active low	

Bus Grant Acknowledge ($\overline{BGACK}$) is a bidirectional control line. As an output, it is generated by the DMAC to indicate that it is the bus master.

As an input, $\overline{BGACK}$ is monitored by the DMAC, in limited rate auto-request mode, to determine whether or not the current bus master is a DMA device or not. $\overline{BGACK}$ is also monitored during bus arbitration in order to assume bus ownership.

- **Interrupt Request ($\overline{IRQ}$)**

Output	Open drain
Active low	

This line is used to request an interrupt to the MPU.

- **Interrupt Acknowledge ($\overline{IACK}$)**

Input
Active low

This line is an input to the DMAC indicating that the current bus cycle is an interrupt acknowledge cycle by the MPU. The

DMAC responds the interrupt vector of the channel with the highest priority requesting an interrupt. There are two kinds of the interrupt vectors for each channel: normal (NIV) or error (EIV). $\overline{\text{IACK}}$ is not serviced if the DMAC has not generated $\overline{\text{IRQ}}$.

- **Channel Request ($\overline{\text{REQ}_0}$ through $\overline{\text{REQ}_3}$)**

Input Active low or falling edge

These lines are the DMA transfer request inputs from the peripheral devices.

These lines are falling edge sensitive inputs when the request mode is cycle steal. They are low-level sensitive when the request mode is burst.

- **Channel Acknowledge ($\overline{\text{ACK}_0}$ through $\overline{\text{ACK}_3}$)**

Output Active low

These lines indicate to the I/O device requesting a transfer that the request is acknowledged and the transfer is to be performed. These lines may be used as a part of the enable circuit for bus interface to the peripheral.

- **Peripheral Control Line ($\overline{\text{PCL}_0}$ through $\overline{\text{PCL}_3}$)**

Input/Output Active low	Three-statable

The four lines ($\overline{\text{PCL}_0} \sim \overline{\text{PCL}_3}$) are multi-purpose lines which may be individually programmed to be a START output, an Enable Clock input, a READY input, an ABORT input, a STATUS input, or an INTERRUPT input.

- **Done ($\overline{\text{DONE}}$)**

Input/Output Active low	Open Drain

As an output, this line is asserted concurrently with the $\overline{\text{ACK}_X}$ timing to indicate the last data transfer to the peripheral device. As an input, it allows the peripheral device to request a normal termination of the DMA transfer.

- **Device Transfer Complete ($\overline{\text{DTC}}$)**

Output Active low	Three-statable

This line is asserted when the DMA bus cycle has terminated normally with no exceptions. It may be used to supply the data latch timing to the peripheral device. In this case, data is valid at the falling edge of $\overline{\text{DTC}}$.

■ INTERNAL ORGANIZATION

The DMAC has four independent DMA channels. Each channel has its own set of channel registers. These registers define and control the activity of the DMAC in processing a channel operation.

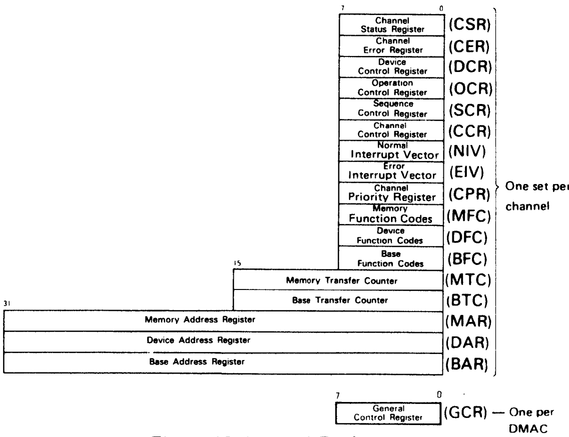

7
0
Channel Status Register (CSR)
Channel Error Register (CER)
Device Control Register (DCR)
Operation Control Register (OCR)
Sequence Control Register (SCR)
Channel Control Register (CCR)
Normal Interrupt Vector (NIV)
Error Interrupt Vector (EIV)
Channel Priority Register (CPR)
Memory Function Codes (MFC)
Device Function Codes (DFC)
Base Function Codes (BFC)
15
Memory Transfer Counter (MTC)
Base Transfer Counter (BTC)
31
Memory Address Register (MAR)
Device Address Register (DAR)
Base Address Register (BAR)
One set per channel
General Control Register (GCR) — One per DMAC

Figure 10 Internal Registers

- **Register Organization**

The internal register addresses are represented in Table 1. Address space not used within the address map is reserved for future expansion. A read from an unused location in the map results in a normal bus cycle with all ones for data. A write to one of these locations results in a normal bus cycle but no write occurs.

Unused bits of the defined registers in Table 1 read as zeros.

Table 1 Internal Register Addressing Assignments

	Address Bits									
Register	7	6	5	4	3	2	1	0	Mode	
Channel Status Register	c	c	0	0	0	0	0	0	R	W*
Channel Error Register	c	c	0	0	0	0	0	1	R	
Device Control Register	c	c	0	0	0	1	0	0	R	W
Operation Control Register	c	c	0	0	0	1	0	1	R	W
Sequence Control Register	c	c	0	0	0	1	1	0	R	W
Channel Control Register	c	c	0	0	0	1	1	1	R	W
Memory Transfer Counter	c	c	0	0	1	0	1	b	R	W
Memory Address Register	c	c	0	0	1	1	s	s	R	W
Device Address Register	c	c	0	1	0	1	s	s	R	W
Base Transfer Counter	c	c	0	1	1	0	1	b	R	W
Base Address Register	c	c	0	1	1	1	s	s	R	W
Normal Interrupt Vector	c	c	1	0	0	1	0	1	R	W
Error Interrupt Vector	c	c	1	0	0	1	1	1	R	W
Channel Priority Register	c	c	1	0	1	1	0	1	R	W
Memory Function Codes	c	c	1	0	1	0	0	1	R	W
Device Function Codes	c	c	1	1	0	0	0	1	R	W
Base Function Codes	c	c	1	1	1	0	0	1	R	W
General Control Register	1	1	1	1	1	1	1	1	R	W

cc:00-Channel #0,01-Channel #1,
10-Channel #2,11-Channel #3,
ss:00-high-order, 01-upper middle,
10-lower middle,11-low-order
b: 0-high-order, 1-low-order
* see Channel Status Register Section

- **Device Control Register (DCR)**

The DCR is a device oriented control register. The XRM bits specifies whether the channel is in burst or cycle steal request mode. The DTYP bits define what type of device is on the channel. If the DTYP bits are programmed to be a HMCS6800 device, the PCL definition is ignored and the $\overline{\text{PCL}}$ line is an Enable clock input. If the DTYP bits are programmed to be a device with $\overline{\text{READY}}$, the PCL definition is ignored and the $\overline{\text{PCL}}$ line is a $\overline{\text{READY}}$ input. The DPS bit defines the port size (eight or sixteen bits) of the peripheral device. (A port size is the largest data which the peripheral device can transfer during a DMA bus cycle.) The PCL bits define the function of the $\overline{\text{PCL}}$ line. If the DTYP bits are programmed to be HMCS6800 device, or Device with $\overline{\text{ACK}}$ and $\overline{\text{READY}}$, these definitions are ignored. The XRM bits are ignored if an auto-request mode (REQG = 00 or 01 in Operation Control Register) is selected.

7	6	5	4	3	2	1	0
XRM		DTYP		DPS	0	PCL	

XRM	(EXTERNAL REQUEST MODE)
00	Burst Transfer Mode
01	(undefined, reserved)
10	Cycle Steal Mode without Hold
11	Cycle Steal Mode with Hold
DTYP	(DEVICE TYPE)
00	HD68000 compatible device, explicitly addressed (dual addressing mode)
01	HD6800 compatible device, explicitly addressed (dual addressing mode)
10	Device with $\overline{\text{ACK}}$, implicitly addressed (single addressing mode)
11	Device with $\overline{\text{ACK}}$ and $\overline{\text{READY}}$, implicitly addressed (single addressing mode)
DPS	(DEVICE PORT SIZE)
0	8 bit port
1	16 bit port
PCL	(PERIPHERAL CONTROL LINE)
00	Status Input
01	Status Input with Interrupt
10	Start Pulse
11	Abort Input

Bit 2 Not Used

- **Operation Control Register (OCR)**

The OCR is an operation control register. The DIR bit defines the direction of the transfer. The SIZE bits define the size of the operand. The CHAIN bits define the type of the CHAIN mode. The REQG bits define how requests for transfers are generated.

7	6	5	4	3	2	1	0
DIR	0	SIZE		CHAIN		REQG	

DIR	(DIRECTION)
0	Transfer from memory to device (transfer from MAR address to DAR address)
1	Transfer from device to memory (transfer from DAR address to MAR address)
SIZE	(OPERAND SIZE)
00	Byte (8 bits)
01	Word (16 bits)
10	Long Word (32 bits)
11	(undefined, reserved)
CHAIN	(CHAINING OPERATION)
00	Chain operation is disabled
01	(undefined, reserved)
10	Array Chaining
11	Linked Array Chaining
REQG	(DMA REQUEST GENERATION METHOD)
00	Auto-request at transfer rate limited by General Control Register (Limited Rate Auto-Request)
01	Auto-request at maximum rate
10	$\overline{\text{REQ}}$ line requests an operand transfer
11	Auto-request the first operand, external request for subsequent operands

Bit 6 Not Used

- **Sequence Control Register (SCR)**

The SCR is used to define the sequencing of memory and device addresses.

7	6	5	4	3	2	1	0
0	0	0	0	MAC		DAC	

MAC	(MEMORY ADDRESS COUNT)
00	Memory address register does not count
01	Memory address register counts up
10	Memory address register counts down
11	(undefined, reserved)
DAC	(DEVICE ADDRESS COUNT)
00	Device address register does not count
01	Device address register counts up
10	Device address register counts down
11	(undefined, reserved)

Bits 7, 6, 5, 4 Not Used

- **Channel Control Register (CCR)**

The CCR is used to start or terminate the operation of a channel. This register also determines if an interrupt request is to be generated. Setting the STR bit causes immediate activation of the channel; the channel will be ready to accept request immediately. The STR and CNT bits of the register cannot be reset by a write to the register. The SAB bit is used to terminate the operation forcedly. Setting the SAB bit will reset STR and CNT. Setting the HLT bit will halt the channel operation, and clearing the HLT bit wil resume the operation. Setting the start bit must be done by a byte access. Otherwise, a timing error occurs.

7	6	5	4	3	2	1	0
STR	CNT	HLT	SAB	INT	0	0	0

STR	(START OPERATION)
0	No operation is pending
1	Start operation
CNT	(CONTINUE OPERATION)
0	No continuation is pending
1	Continue operation
HLT	(HALT OPERATION)
0	Operation not halted
1	Operation halted
SAB	(SOFTWARE ABORT)
0	Channel operation not aborted
1	Abort channel operation
INT	(INTERRUPT ENABLE)
0	No interrupts enabled
1	Interrupts enabled

Bits 2, 1, 0 Not Used

- **Channel Status Register (CSR)**

The CSR is a register containing the status of the channel.

7	6	5	4	3	2	1	0
COC	BTC	NDT	ERR	ACT	0	PCT	PCS

COC	(CHANNEL OPERATION COMPLETE)
0	Channel operation incomplete
1	Channel operation complete
BTC	(BLOCK TRANSFER COMPLETE)
0	Block transfer incomplete
1	Block transfer complete
NDT	(NORMAL DEVICE TERMINATION)
0	No normal device termination by $\overline{\text{DONE}}$ input
1	Device terminated operation normally by $\overline{\text{DONE}}$ input
ERR	(ERROR BIT)
0	No errors
1	Error as coded in CER
ACT	(CHANNEL ACTIVE)
0	Channel not active
1	Channel active
PCT	($\overline{\text{PCL}}$ TRANSITION)
0	No $\overline{\text{PCL}}$ transition occurred
1	$\overline{\text{PCL}}$ transition occurred
PCS	(THE STATE OF THE $\overline{\text{PCL}}$ INPUT LINE)
0	$\overline{\text{PCL}}$ "Low"
1	$\overline{\text{PCL}}$ "High"

Bit 2 Not Used

- **Channel Error Register (CER)**

The CER is an error condition status register. The ERR bit of CSR indicates if there is an error or not. Bits 0-4 indicate what type of error occurred.

7	6	5	4	3	2	1	0
0	0	0	ERROR CODE				

Error Code

00000	No error
00001	Configuration error
00010	Operation timing error
00101	Address error in MAR
00110	Address error in DAR
00111	Address error in BAR
01001	Bus error in MAR
01010	Bus error in DAR
01011	Bus error in BAR
01101	Count error in MTC
01111	Count error in BTC
10000	External abort
10001	Software abort

Bits 7, 6, 5 Not Used

- **Channel Priority Register (CPR)**

The CPR is used to define the priority level of the channel. Priority level 0 is the highest and priority level 3 is the lowest priority.

7	6	5	4	3	2	1	0
0	0	0	0	0	0	CP	

CP (CHANNEL PRIORITY)

00	Priority level 0
01	Priority level 1
10	Priority level 2
11	Priority level 3

Bit 7 through 2 Not Used

- **General Control Register (GCR)**

The GCR is used to define what portion of the bus cycles is available to the DMAC for limited rate auto-request generation. GCR is also used to specify the hold time for cycle steal mode with hold.

7	6	5	4	3	2	1	0
0	0	0	0	BT		BR	

BT (BURST TIME)

The number of DMA clock cycles per burst that the DMAC allows in the auto-request at a limited rate of transfer is controlled by these two bits. The number is $2^{(BT+4)}$ (two to the BT+4 power).

BR (BANDWIDTH RATIO)

The amount of the bandwidth utilized by the auto-request at a limited rate transfer is controlled by these two bits. The ratio is $2^{(BR+1)}$ (two to the BR+1 power).

The hold time for cycle steal mode with hold is defined to be minimum of 1 sample interval and maximum of 2 sample intervals. A sample interval is defined to be $2^{(BT+BR+5)}$ (two to the BT+BR+5 power) clock cycles.

Bits 7 through 4 Not Used

- **Address Registers (MAR, DAR, BAR)**

Three 32-bit registers are utilized to implement the Memory Address Register, Device Address Register, and the Base Address Register. Only the least significant twenty-four bits are connected to the address output pins. The content of the MAR is outputted when the memory is accessed in single or dual adressing mode. The content of the DAR is outputted when the peripheral device is accessed. The contents of the BAR is outputted when reading chain information from memory in the Array Chaining Mode or the Linked Array Chaining Mode. It is also used to set the top address of the next block transfer in Continue mode.

- **Function Code Registers (MFC, DFC, BFC)**

The DMAC has three function code registers per channel: the Memory Function Code Register (MFC), Device Function Code Register (DFC), and the Base Function Code Register (BFC). The contents of these registers are outputted from FC_0 through FC_2 lines when an address is outputted from MAR, DAR, or BAR, respectively. The BFC is also used to set the MFC for the transfer of the next data block in the Continue mode.

7	6	5	4	3	2	1	0
0	0	0	0	0	FC2	FC1	FC0

Bits 3 through 7 Not Used

- **Transfer Count Registers (MTC, BTC)**

Each channel has two 16-bit counters: the Memory Transfer Counter (MTC) and the Base Transfer Counter (BTC). The MTC counts the number of transfer words in one block, and is decreased by one for every operand transfer.

The BTC is used to count the number of data blocks in the Array Chaining Mode. BTC is also used to set the number of operands to transfer for the next data block in the Continue Mode.

- **Interrupt Vector Registers (NIV, EIV)**

Each channel has a Normal Interrupt Vector register and an Error Interrupt Vector register.

When an interrupt acknowledge cycle occurs, an interrupt vector is outputted from one of these registers. If the error bit (CSR) is set for the channel with interrupt pending, then content of EIV is outputted, otherwise content of NIV is outputted.

■ OPERATION DESCRIPTION

A DMAC channel operation proceeds in three principal phases. During the initialization phase, the MPU sets the channel control registers, supply the initial address and the number of transfer words, and starts the channel. During the transfer phase, the DMAC accepts requests for data operand transfers, and provides addressing and bus controls for the transfers. The termination phase occurs after the operation is completed.

This section describes DMAC operations. A description of the MPU/DMAC communication is given first. Next, the transfer phase is covered, including how the DMAC recognizes requests and how the DMAC arranges for data transfer. Following this, the initialization phase is described. The termination phase is covered, introducing chaining, error signaling, and bus exceptions. A description of the channel priority scheme rounds out the section.

- **Read/Write of the DMAC Registers by MPU**

The MPU reads and writes the DMAC internal registers and controls the DMA transfer. Figure 11 indicates the timing diagram when the MPU reads the contents of the DMAC register. The MPU outputs A_1-A_{23}, FC_0-FC_2, $\overline{AS}$, R/$\overline{W}$, $\overline{UDS}$, and $\overline{LDS}$, and accesses the DMAC internal register. The specific internal register is selected by A_1-A_7, $\overline{LDS}$ and $\overline{UDS}$. The $\overline{CS}$ and $\overline{IACK}$ lines are generated by the external circuit with A_8-A_{23} and FC_0-FC_2. The DMAC outputs data on the data bus, together with $\overline{DDIR}$, $\overline{DBEN}$ and $\overline{DTACK}$. The $\overline{DDIR}$ and $\overline{DBEN}$ control the bidirectional buffer on the bus and the $\overline{DTACK}$ indicates that the data has been sent or received by the DMAC. Read Cycle is eighteen CLKs. Figure 12 shows the MPU write cycle. Write cycle is fifteen CLKs.

Note the following points.

(1) The clock reference shown in this figure is the DMAC input clock.
(2) The $\overline{DDIR}$ and the $\overline{DBEN}$ are three-stated at the beginning which detects $\overline{CS}$ and the ending of the cycle.
(3) During the MPU read cycle, the $\overline{DTACK}$ is asserted after the data is valid on the system bus.
(4) During the MPU write cycle, the $\overline{DDIR}$ line will be driven low to direct the data buffers toward to DMAC before the buffers are enabled.
(5) During the MPU write cycle, the DMAC will latch the data before asserting $\overline{DTACK}$. Then it will negate $\overline{DBEN}$ and $\overline{DDIR}$ in the proper order.
(6) After the MPU cycle and the $\overline{LDS}$ and the $\overline{UDS}$ are negated by the MPU, the DMAC will put $\overline{DBEN}$, $\overline{DDIR}$ and the address data lines to a high impedance state.
(7) $\overline{DTACK}$ will once go "High" and then to a high impedance state after negating $\overline{LDS}$ and $\overline{UDS}$.

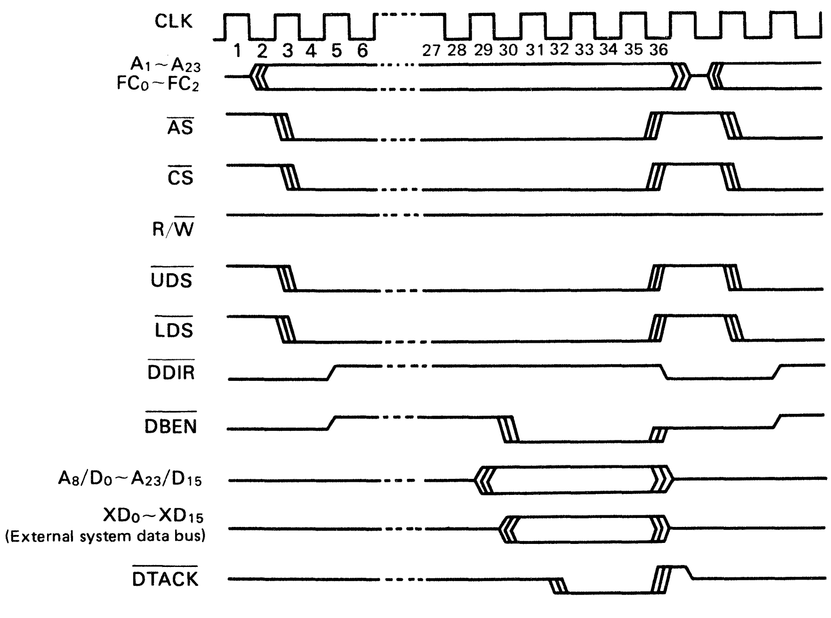


Figure 11 MPU Read from DMAC – Word

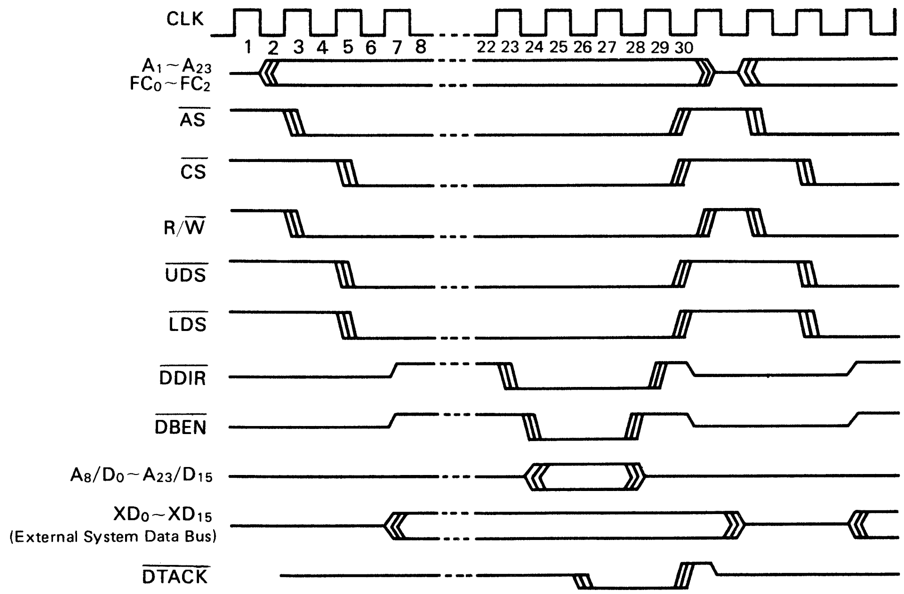
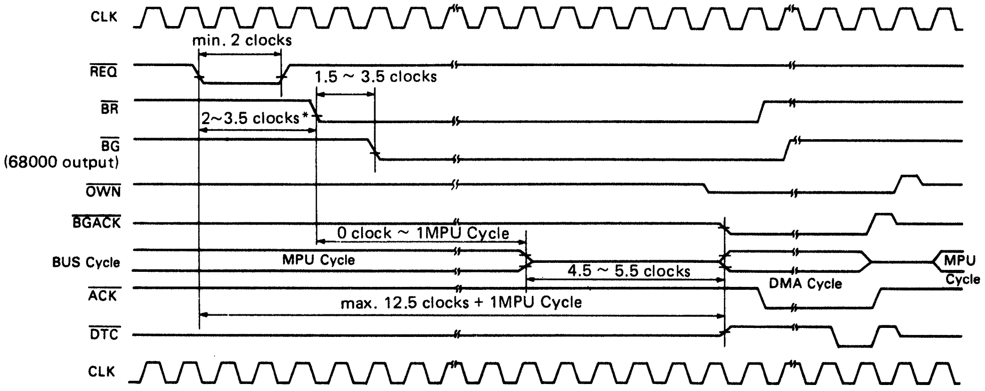


Figure 12 MPU Write to DMAC — Word

- **Bus Arbitration**

The DMAC must obtain ownership of the bus in order to transfer data. Figure 13 indicates the DMAC bus arbitration timing. It is completely compatible with that of HD68000 MPU. The DMAC asserts the Bus Request ($\overline{BR}$) to request the bus mastership. The MPU recognizes the request and asserts BG, then it grants the ownership in the next bus cycle. After the end of the current cycle ($\overline{AS}$ is negated), the MPU relinquishes the bus to the DMAC. The DMAC asserts the bus grant acknowledge ($\overline{BGACK}$) to indicate that it has the bus ownership. A half clock before $\overline{BGACK}$ is asserted, the DMAC asserts $\overline{OWN}$. $\overline{OWN}$ is kept asserted for a half clock after $\overline{BGACK}$ is negated at the end of the DMA cycle. $\overline{BR}$ is negated one clock after $\overline{BGACK}$ is asserted.



* This case assumes that no exception condition exists and DMAC isn't accessed by MPU.

Figure 13 DMAC Bus Arbitration Timing

- **Device/DMAC Communication**

Communication between peripheral devices and the DMAC is accomodated by five signal lines. Each channel has $\overline{REQ}$, $\overline{ACK}$ and $\overline{PCL}$, and the last two lines the $\overline{DONE}$ and $\overline{DTC}$ lines, are shared among the four channels.

(1) Request ($\overline{REQ}$)

The peripheral devices assert $\overline{REQ}$ to request data transfers. See the "Requests" section for details.

(2) Acknowledge ($\overline{ACK}$)

This line is used to implicitly address the device which is transferring the data (This device is not selected by address lines.) It is also asserted when the content of DAR is outputted during memory-to-memory transfer except for the auto-request mode at a limited rate or at the maximum rate.

(3) Peripheral Control Line ($\overline{PCL}$)

The function of this line is quite flexible and is determined by the DCR (Device Control Register).

The DTYP bits of the DCR define what type of device is on the channel. If the DTYP bits are programmed to be a HMCS6800 device, the PCL definition is ignored and the $\overline{PCL}$ line is an Enable clock (E clock) input. If the DTYP bits are programmed to be a device with $\overline{READY}$, the PCL definition as ignored and the $\overline{PCL}$ line is a ready input.

$\overline{PCL}$ As a Status Input

The $\overline{PCL}$ line may be programmed as a status input. The status level of this line can be determined by the PCS bit in the CSR, regardless of the PCL function determined by the DCR. If a negative transition occurs and remains stable for a minimum of two clocks, the PCT bit of the CSR is set. This PCT bit is cleared by resetting the DMAC or the writing "1" to the PCT bit.

$\overline{PCL}$ As an Interrupt

The $\overline{PCL}$ line may be programmed to generate an interrupt on a negative transition. This enables an interrupt which is requested if the PCT bit of the CSR is set. When using this function, it is necessary to reset the PCT bit in the CSR before the PCL bit in the DCR is set to interrupt, in order to avoid assertion of IRQ line at this time.

$\overline{PCL}$ As a Starting Pulse

The $\overline{PCL}$ line may be programmed to output a starting pulse. This active low starting pulse is outputted when a channel is activated, and is "Low" for a period of four clock cycles.

$\overline{PCL}$ As an Abort Input

The $\overline{PCL}$ line may be programmed to be a negative transition above input which terminates an operation by setting the external abort error in CER. It is necessary to reset the PCT bit in the CSR before activating the channel (Setting the ACT bit of CCR) so that the channel operation is not immediately aborted.

$\overline{PCL}$ As an Enable Clock (E Clock) Input

If the DTYP bits are programmed to be a HMCS6800 device, the PCL definition is ignored and the $\overline{PCL}$ line is an Enable Clock input. The Enable clock downtime must be as long as five clock cycles, and must be high for a minimum of three DMAC clock cycles, but need not be synchronous with the DMAC's clock.

$\overline{PCL}$ As a $\overline{READY}$ Input

If the DTYP bits are programmed to be a device with $\overline{READY}$, the PCL definition is ignored and the $\overline{PCL}$ line is a $\overline{READY}$ input. The $\overline{READY}$ is an active low input.

(4) DONE ($\overline{DONE}$)

This line is an active low Input/Output signal with an open drain. It is asserted when the memory transfer count is exhausted in a single block transfer. In the chaining operation, $\overline{DONE}$ is asserted only at the last transfer to the peripheral device of the last data block. In the continue mode, $\overline{DONE}$ is asserted for each data block. It is asserted and negated in coincident with the $\overline{ACK}$ line for the last data transfer to the peripheral device. It is also outputted in coincident with the $\overline{ACK}$ line of the last bus cycle, in which the address is outputted from the DAR, in the memory-to-memory transfer (dual addressing mode) that uses the $\overline{ACK}$ line.

The DMAC also monitors the state of the $\overline{DONE}$ line during the DMA bus cycle. If the device asserts $\overline{DONE}$ during $\overline{ACK}$ active, the DMAC will terminate the operation after the transfer of the current operand. If $\overline{DONE}$ is asserted on the first byte of 2 byte operation or the first word of long word operation, the DMAC does not terminate the operation until the whole operand transfer is completed. If $\overline{DONE}$ is inserted, then the DMAC terminates the operation by clearing the ACT bit of the CSR, and setting the COC and NDT bits of the CSR. If both the DMAC and the device assert $\overline{DONE}$, the device termination is not recognized, but the channel operation does terminate. $\overline{DONE}$ is outputted again for the retry exceptions bus cycles.

(5) Data Transfer Complete ($\overline{DTC}$)

$\overline{DTC}$ is an active low signal which is asserted when the actual data transfer is accomplished. It is also asserted in the bus cycle when a chain information is read from memory in the Chaining mode. However, if exceptions are generated and the DMA bus cycle terminates, $\overline{DTC}$ is not asserted. $\overline{DTC}$ is asserted one half clock before $\overline{LDS}$ and $\overline{UDS}$ are negated, and negated one half clock after $\overline{LDS}$ and $\overline{UDS}$ are negated.

- **Requests**

Requests may be externally generated by circuitry in the peripheral device, or internally generated by the auto-request mechanism. The REQG bits of the OCR determine these modes. The DMAC also supports an operation in which the DMAC auto-requests the first transfer and then waits for the peripheral device to request the following transfers.

(1) Auto-request Transfers

The auto-request mechanism provides generation of requests within the DMAC. These requests can be generated at either of two rates: maximum-rate and limited-rate. In the former case, the channel always has a request pending.

The limited rate auto-request functions by monitoring the bus utilization.

Limited-rate Auto-request

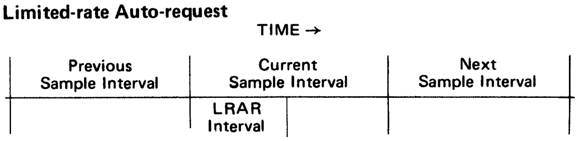


Figure 14 DMAC Sample Intervals

In the limited-rate auto-request the DMAC devides time into equal length sample intervals by counting clock cycles. The end of one sample interval makes the beginning of the next. During a sample interval, the DMAC monitors by means of $\overline{BGACK}$ pin the system bus activity of the DMAC and other bus master devices. At the end of the sample interval, decision is made whether or not to perform the channel's data transfer during the next sample interval. Namely, based on the activity of the DMAC or other bus master devices during the current sample interval, the DMAC allows limited-rate auto-requests for some initial portion of the next sample interval.

The length of the sample interval, and the portion of the sample interval during which limited-rate auto-requests can be made (the limited-rate auto-request interval) are controlled by the BT and BR bits in the GCR. The length in clock cycles of the limited-rate auto-request interval is $2^{(BT+4)}$ (2 raised to the BT+4 power). For example, if BT equals 2 and the DMA utilization of the bus was low during the previous sample interval, then the DMAC generates the auto-request transfers during the first 64 clock cycles.

The ratio of the length of the sample interval to the length of the limited-rate auto-request interval is controlled by the BR bits. The ratio of the system bus utilization of the MPU to other bus master devices including he DMAC is $2^{(BR+1)}$ (2 raised to the BR+1 power). If the fraction of DMA clock cycles during the sample interval exceeds the programmed utilization level, the DMAC will not allow limited-rate auto-requests during the next sample interval.

For example, if BR equals 3, then at most one out of 16 clock cycles during a sample interval can be used by the DMAC and other bus master devices, and still the DMAC would allow limited rate auto-request during the next sample interval. Therefore, from the viewpoint of long period, the ratio of the system bus utilization of the MPU to I/O devices including the DMAC is about 16:1. The sample interval length is not a direct parameter, but it is equal to $2^{(BT+BR+5)}$ clock cycles. Thus, the sample interval can be programmed between 32 and 2048 clock cycles.

The DMAC uses the $\overline{\text{BGACK}}$ to differentiate between the MPU bus cycle and DMAC or other bus master devices. If $\overline{\text{BGACK}}$ is active, then the DMAC assumes that the bus is used by a DMAC or other bus master devices. If it is inactive, then the DMAC assumes that it is used by the MPU.

Maximum-rate Auto-request

If the REQG bits in the OCR indicate auto-request at the maximum rate, the DMAC acquires the bus after the start bit is set and keeps it until the data transfer is completed.

If a request is made by another channel of higher priority, the DMAC services that channel and then resumes the auto-request sequence. If two or more channels are set to equal priority level and maximum rate auto-request, then the channels will rotate in a "round robbin" fashion.

If the HMCS68000 compatible device is connected to a channel, the $\overline{\text{ACK}}$ line is held inactive during an auto-request operation. Consequently, any channel may be used for the memory-to-memory transfer with the auto-request function in addition to the operation of data transfer between memory and peripheral device with using the $\overline{\text{REQ}}$ pin. Refer to Figure 15 for the timing of the memory-to-memory transfer. In this mode, the $\overline{\text{ACK}}$, $\overline{\text{HIBYTE}}$ and $\overline{\text{DONE}}$ outputs are always inactive.

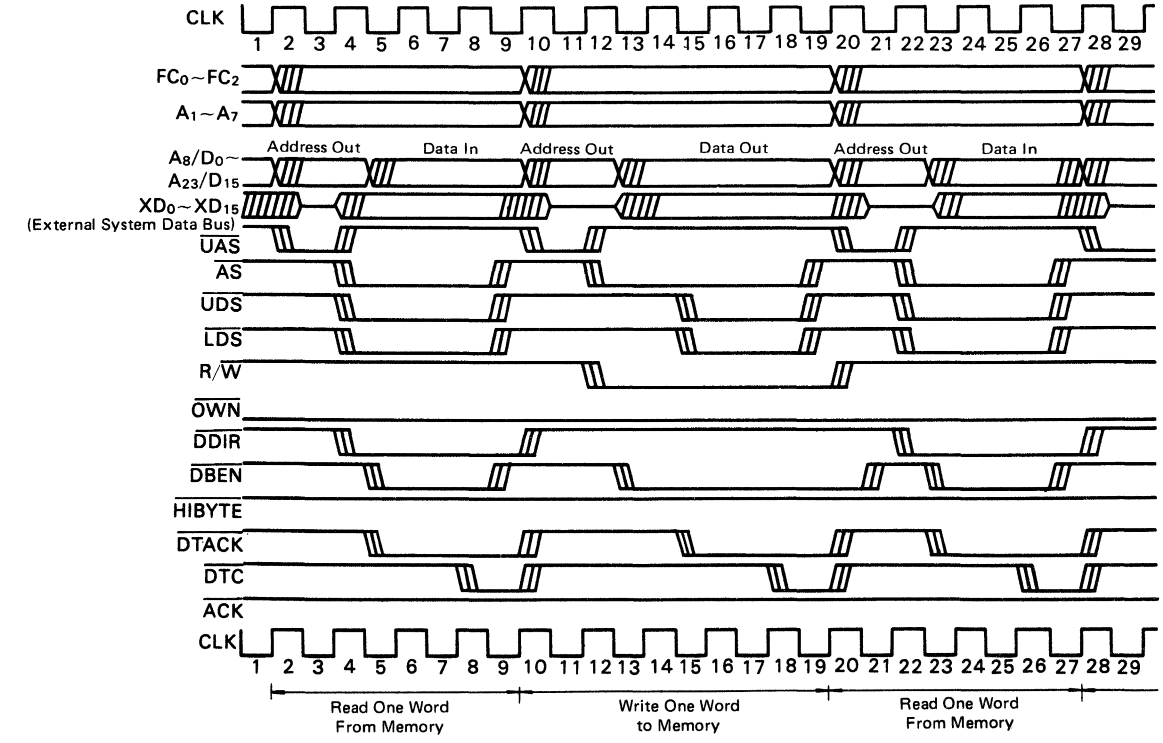


Figure 15 Memory-to-Memory Transfer Read-Write-Read Cycles

(2) External Requests

If the REQG bits of the OCR indicate that the $\overline{REQ}$ line generates requests, the transfer requests are generated externally. The request line associated with each channel allows the device to externally generate requests for DMA transfers. When the device wants an operand transferred, it makes a request by asserting the request line. The external request mode is determined by the XRM bits of the DCR, which allows both burst and cycle steal request modes. The burst request mode allows a channel to request the transfer of multiple operands using consecutive bus cycles. The cycle steal request mode allows a channel to request the transfer of a single operand. The following is the description of the burst and the cycle steal modes.

Burst Request Recognition

In the burst request mode, the $\overline{REQ}$ line is an active low input. The level sampled at the rising edge of the clock. Once the burst request is asserted, it needs to be held low until the first DMA bus cycle starts in order to insure at least one data transfer operation. In order to stop the burst mode transfer after the current bus cycle, the $\overline{REQ}$ line has to be negated one clock before the $\overline{DTC}$ output clock of this cycle. Refer to Figure 16 or the burst mode timing.

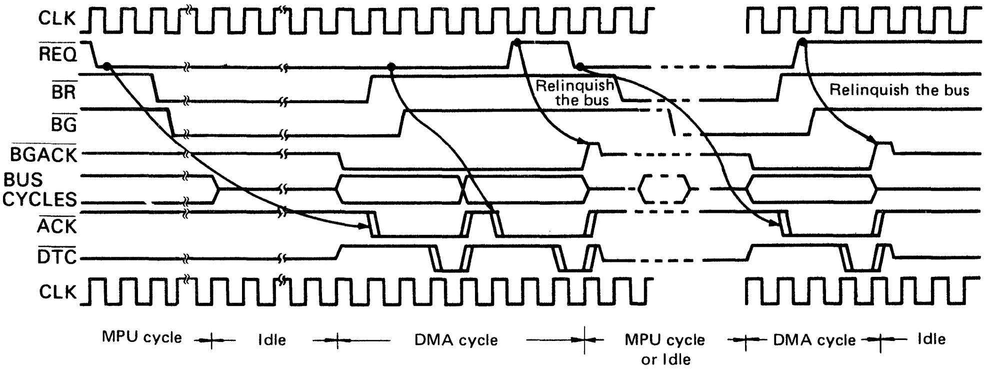


Figure 16 Burst Mode Request Timing
(Only one channel is active)

Cycle Steal Request Recognition

In the cycle steal request mode, the peripheral device requests the DMA transfer by generating an falling edge at the $\overline{REQ}$ line. The $\overline{REQ}$ line needs to be held "low" for at least 2 clock cycles. In the cycle steal mode, if the $\overline{REQ}$ line changes from "High" to "Low" between $\overline{ACK}$ output and one clock before the clock that outputs $\overline{DTC}$, then the next DMA transfer is performed without relinquishing the bus. If the bus is not relinquished, then maximum of 5 idle clocks is inserted between bus cycles. Refer to Figure 17 for the request timing of the cycle steal mode. If the XRM bits specify cycle steal without hold, the DMAC will relinquish the bus. If the XRM bits specify cycle steal with hold, the DMAC will retain ownership. The bus is not given up for arbitration until the channel operation terminates or until the device pauses. The device is determined to have paused if it does not make any requests during the next full sample interval. The sample interval counter is free running and is not reset or modified by this mode of operation. The sample interval counter is the same counter that is used for Limited Rate Auto Request and is programmed via the GCR. Figure 18 shows the request timing in the cycle steal bus hold. If the $\overline{REQ}$ is inputted during the hold time, the $\overline{ACK}$ is outputted after a maximum of 7.5 clock cycles from the picked-up clock. On the cycle steal with hold mode, the DMAC will hold the bus even when the transfer count is exhausted and the last data has been transferred. If DMA transfer is requested from other channels during this period, they are executed normally.

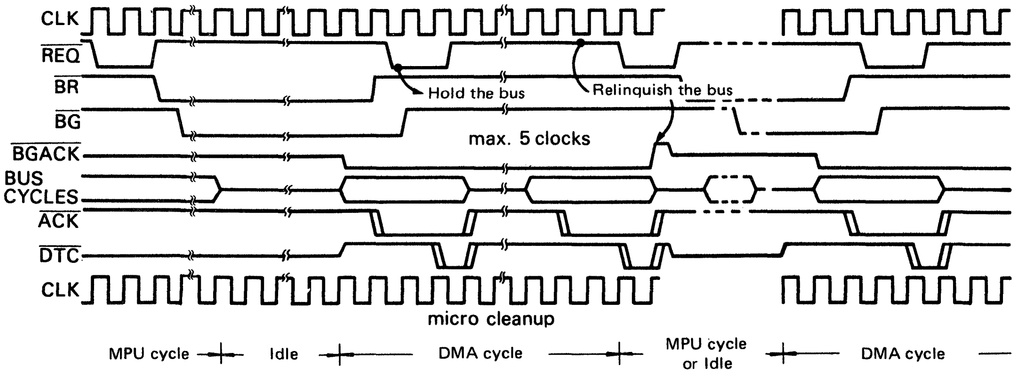


Figure 17 Cycle Steal Mode Request Timing

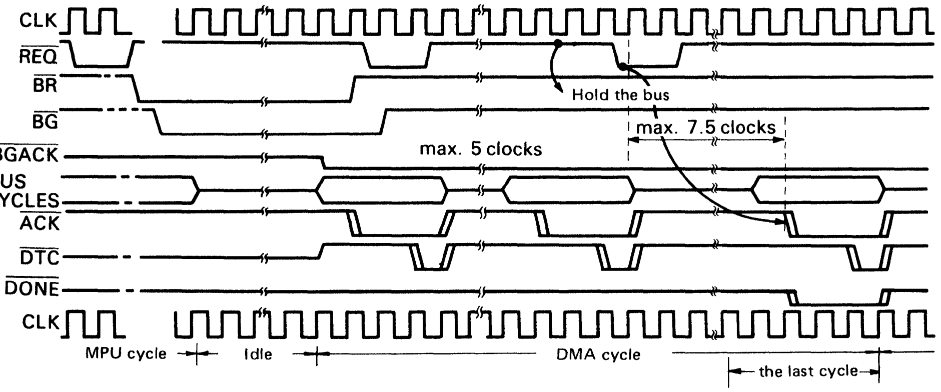


Figure 18 Cycle Steal Bus Hold Mode Request Timing

Request Recognition in Dual-address Transfers

In the following section dual-address transfers is defined. Dual address transfer is an exception to the request recognition rules in the previous paragraphs. In the cycle steal request mode, when there are two or more than transfers between the DMAC and the peripheral device during one operand transfer, the request is not recognized until the last transfer between the DMAC and the I/O device starts.

(3) Mixed Request Generation

A single channel can mix the two request generation methods. By programming the REQG bits of the OCR to "11", when the channel is started, the DMAC auto-requests the first transfer. Subsequent requests are then generated externally by the device. The $\overline{\mathrm{ACK}}$ and $\overline{\mathrm{PCL}}$ lines perform their normal functions in this operation.

- **Data Transfers**

All DMAC data transfers are assumed to be between memory and the peripheral device. The word "memory" means a 16-bit HMCS68000 bus compatible device. By programming the DCR, the characteristics of the peripheral device may be assigned. Each channel can communicate using any of the following protocols.

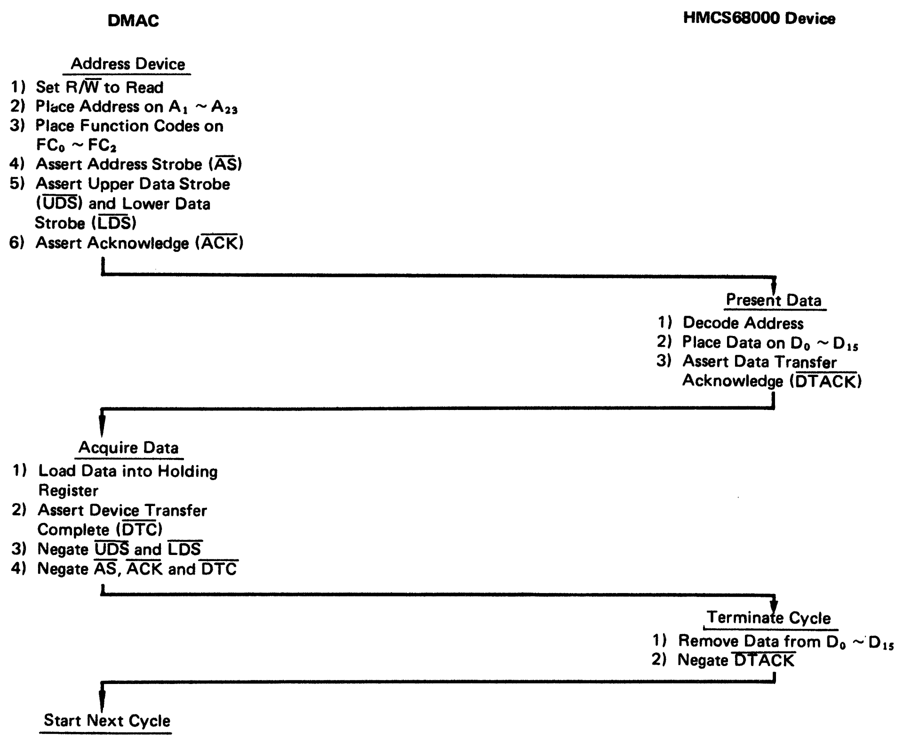
DTYP	Device Type	
00	HMCS68000 compatible device	Dual Addressing
01	HMCS6800 compatible device	Dual Addressing
10	Device with $\overline{\mathrm{ACK}}$	Single Addressing
11	Device with $\overline{\mathrm{ACK}}$ and $\overline{\mathrm{READY}}$	Single Addressing

(1) Dual Addressing

HMCS68000 and HMCS6800 compatible devices may be explicitly addressed. This means that before the peripheral transfers data, a data register within the device must be addressed. Because the address bus is used to address the peripheral, the data cannot be directly transferred to/from the memory because the memory also requires addressing. Instead, the data is transferred from the source to the DMAC and held in an internal DMAC holding register. A second bus transfer between the DMAC and the destination is then required to complete the operation. Because both the source and destination of the transfer are explicitly addressed, this protocol is called dual-addressed.

HMCS68000 Compatible Device Transfers

In this operation, when a request is received, the bus is obtained and the transfer is completed using the protocol as shown in Figures 19 and 20. Figures 21 through 24 show the transfer timings. Figure 21 and 24 show the operation when the memory is the source and the peripheral device is the destination. Figures 22 and 23 show the transfer in the opposite direction. The peripheral device is a 16-bit device in Figures 21 and 22, and a 8-bit device in Figures 23 and 24.



DMAC

HMCS68000 Device

Address Device
1) Set $R/\overline{W}$ to Read
2) Place Address on $A_1 \sim A_{23}$
3) Place Function Codes on $FC_0 \sim FC_2$
4) Assert Address Strobe ($\overline{AS}$)
5) Assert Upper Data Strobe ($\overline{UDS}$) and Lower Data Strobe ($\overline{LDS}$)
6) Assert Acknowledge ($\overline{ACK}$)

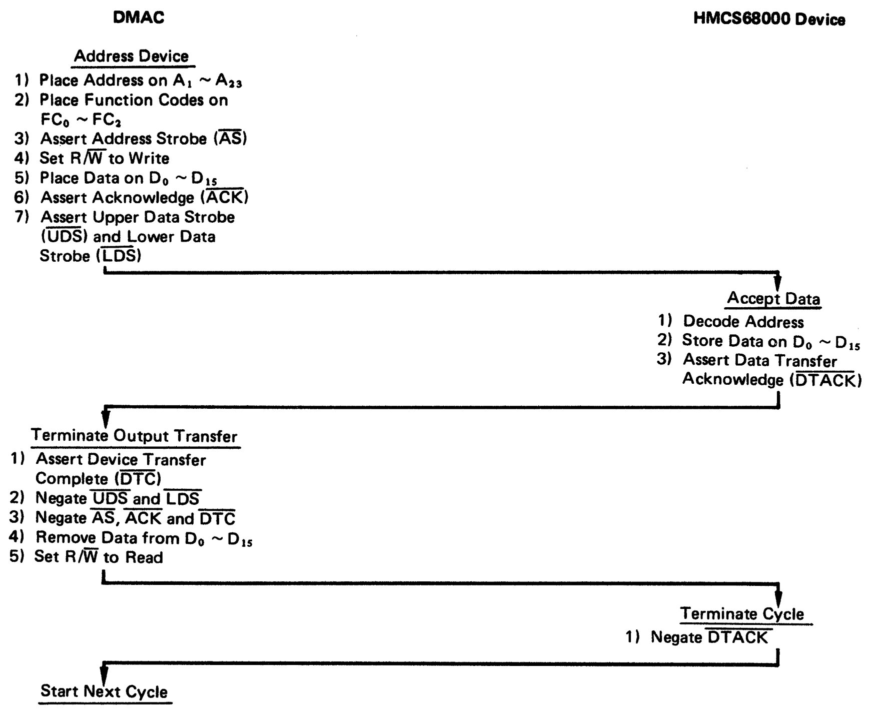
Present Data
1) Decode Address
2) Place Data on $D_0 \sim D_{15}$
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)

Acquire Data
1) Load Data into Holding Register
2) Assert Device Transfer Complete ($\overline{DTC}$)
3) Negate $\overline{UDS}$ and $\overline{LDS}$
4) Negate $\overline{AS}$, $\overline{ACK}$ and $\overline{DTC}$

Terminate Cycle
1) Remove Data from $D_0 \sim D_{15}$
2) Negate $\overline{DTACK}$

Start Next Cycle

Figure 19 Word Read Cycle Flowchart HMCS68000 Type Device



DMAC

HMCS68000 Device

Address Device
1) Place Address on $A_1 \sim A_{23}$
2) Place Function Codes on $FC_0 \sim FC_2$
3) Assert Address Strobe ($\overline{AS}$)
4) Set $R/\overline{W}$ to Write
5) Place Data on $D_0 \sim D_{15}$
6) Assert Acknowledge ($\overline{ACK}$)
7) Assert Upper Data Strobe ($\overline{UDS}$) and Lower Data Strobe ($\overline{LDS}$)

Accept Data
1) Decode Address
2) Store Data on $D_0 \sim D_{15}$
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)

Terminate Output Transfer
1) Assert Device Transfer Complete ($\overline{DTC}$)
2) Negate $\overline{UDS}$ and $\overline{LDS}$
3) Negate $\overline{AS}$, $\overline{ACK}$ and $\overline{DTC}$
4) Remove Data from $D_0 \sim D_{15}$
5) Set $R/\overline{W}$ to Read

Terminate Cycle
1) Negate $\overline{DTACK}$

Start Next Cycle

Figure 20 Word Write Cycle Flowchart HMCS68000 Type Device

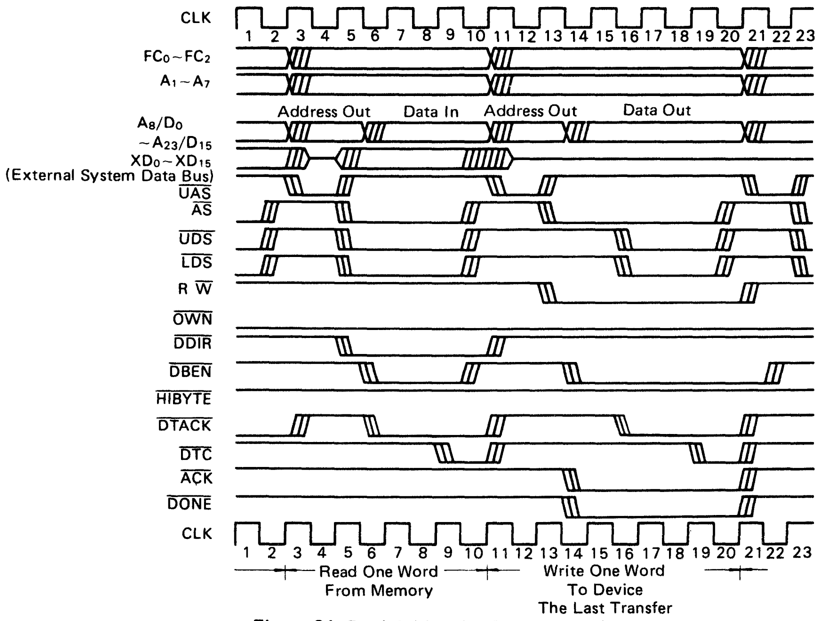


Figure 21 Dual Addressing Mode, Read/Write Cycle, Destination = 16-bit Device, Word Operand

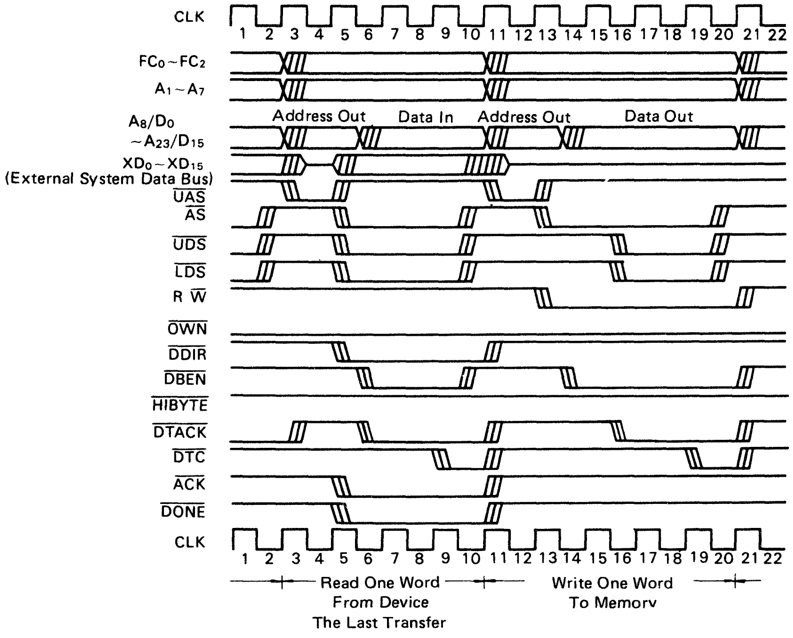


Figure 22 Dual Addressing Mode, Read/Write Cycle, Source = 16-bit Device, Word Operand

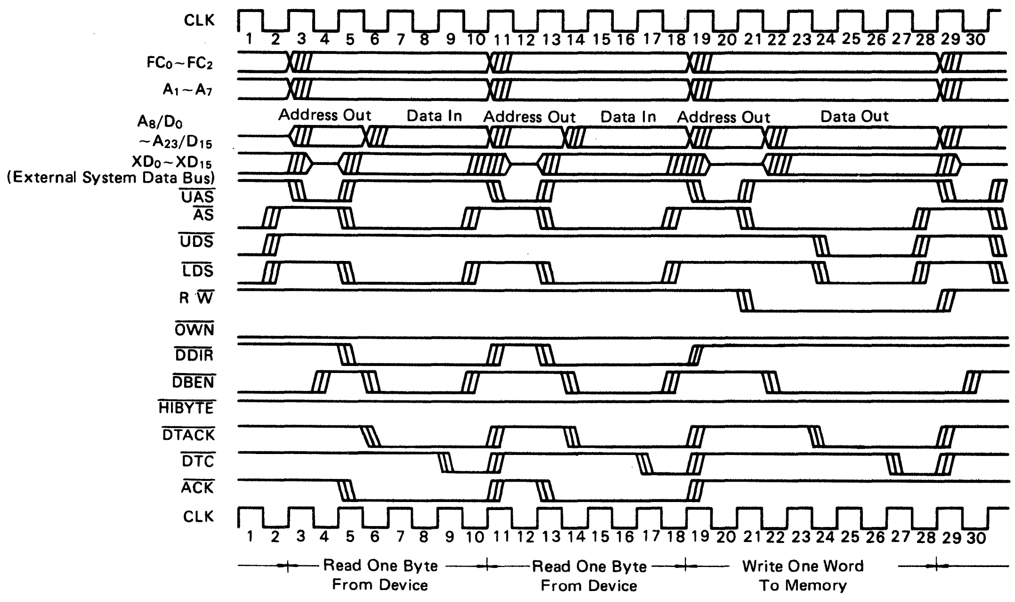


Figure 23 Dual Addressing Mode, Read/Write Cycle
Source = 8-bit Device, Word Operand

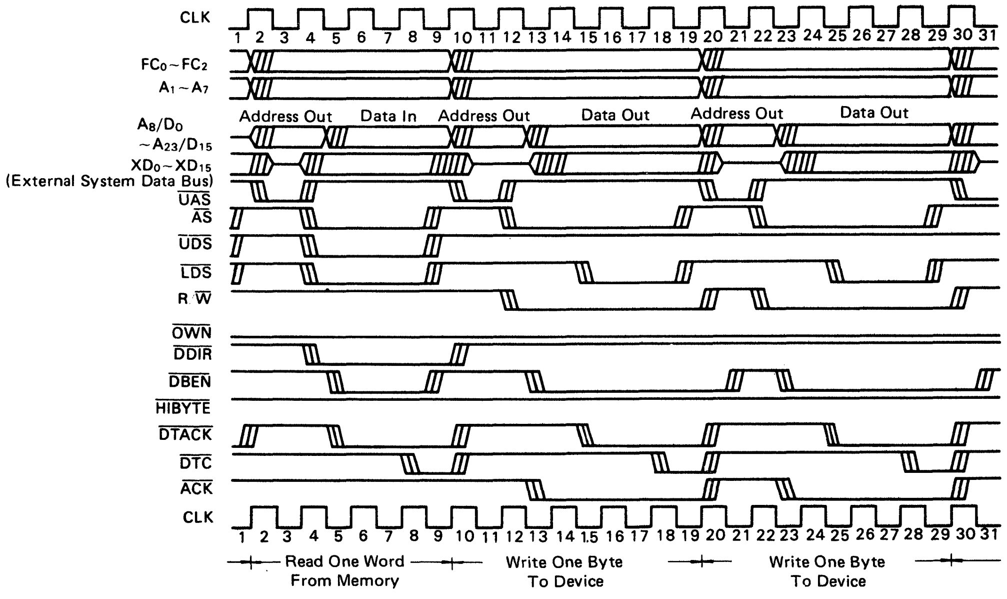


Figure 24 Dual Addressing Mode, Read/Write Cycle,
Destination = 8-bit Device, Word Operand

HMCS6800 Compatible Device Transfers

When a channel is programmed to perform HMCS6800 compatible transfers, the $\overline{\text{PCL}}$ line for that channel is defined as an Enable clock input. The DMAC performs data transfers between itself and the peripheral device using the HMCS6800 bus protocol, with the $\overline{\text{ACK}}$ output providing the $\overline{\text{VMA}}$ (valid memory address) signal. Figure 25 illustrates this protocol. Refer to Figure 26 for the read cycle timing and Figure 27 for the write cycle timing. In Figure 26, the DMAC latches the data at the falling edge of clock 19, so a latch to hold the data is necessary as shown in Figure 47.

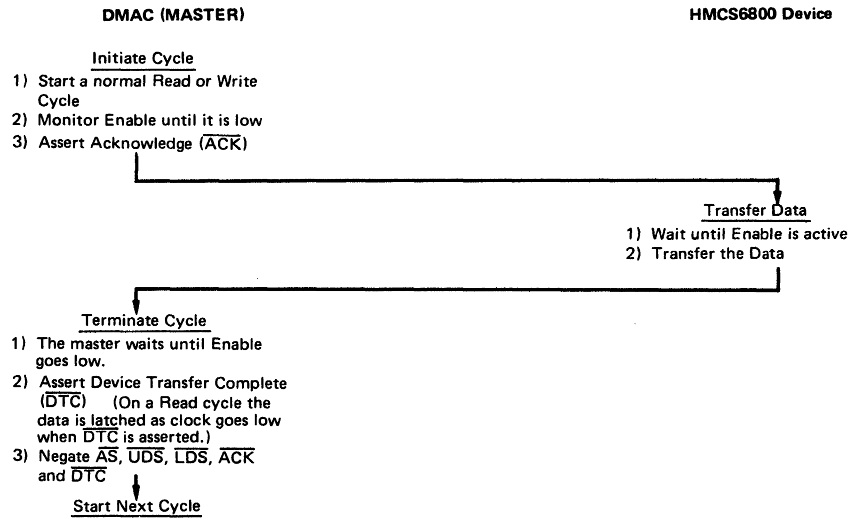

DMAC (MASTER)
HMCS6800 Device
Initiate Cycle
1) Start a normal Read or Write Cycle
2) Monitor Enable until it is low
3) Assert Acknowledge (ACK)
Transfer Data
1) Wait until Enable is active
2) Transfer the Data
Terminate Cycle
1) The master waits until Enable goes low.
2) Assert Device Transfer Complete (DTC) (On a Read cycle the data is latched as clock goes low when DTC is asserted.)
3) Negate AS, UDS, LDS, ACK and DTC
Start Next Cycle

Figure 25 HMCS6800 Cycle Flowchart

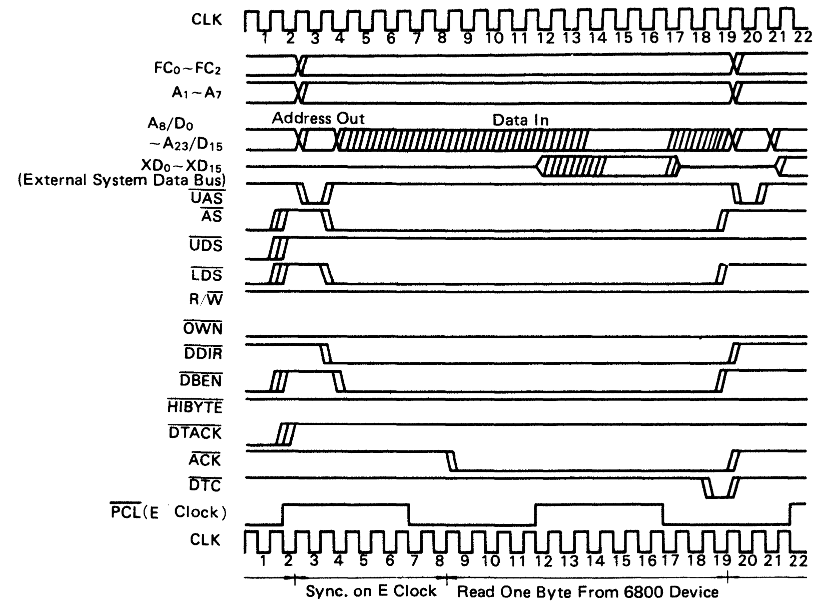


Figure 26 Dual Addressing Mode, HMCS6800 Compatible Device, Read Cycle

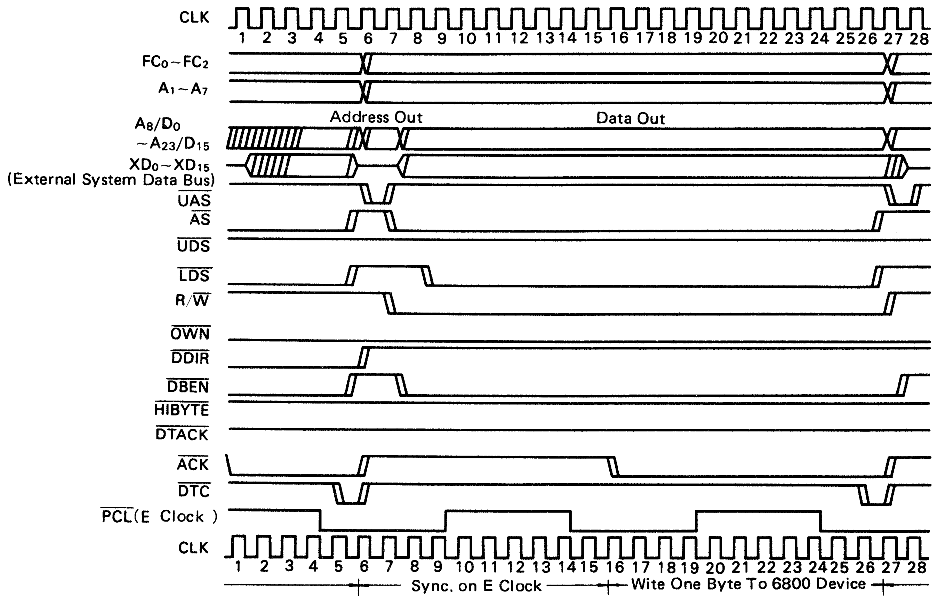


Figure 27 Dual Addressing Mode, HMCS6800 Compatible Device, Write Cycle

(2) Single Addressing Mode

Implicitly addressed devices are peripheral devices selected not by address but by $\overline{ACK}$. They do not require addressing of data register during data transfer. Transfers between memory and these devices are controlled by the request/acknowledge protocol. Such peripherals require only one bus cycle to transfer data, and the DMAC internal holding register is not used. Because only the memory is addressed during a data transfer and a transfer done in only on bus cycle, this protocol is called single-address.

Device with $\overline{ACK}$ Transfers

Under this protocol, the communication between peripheral device and the DMAC is performed with a two signal $\overline{REQ}/\overline{ACK}$ handshake. When a request is generated using the request method programmed in the DMAC's internal control registers, the DMAC obtains the bus and responds with $\overline{ACK}$. The DMAC asserts all the bus control signals required for the memory access. Refer to Figure 28 for the flowchart of the data transfer from memory to the device with $\overline{ACK}$. Figure 29 shows the flowchart of the data transfer from the device with $\overline{ACK}$ to memory. When a request is generated using the request method programmed in the control registers, the DMAC obtains the bus and responds with acknowledge. The DMAC asserts all HMCS68000 bus control signals needed for the transfer. When the DMAC accepts $\overline{DTACK}$ from memory, it asserts $\overline{DTC}$ and informs the peripheral device of the transfer termination. Figure 30 and 31 show the transfer timings of the device with $\overline{ACK}$: the port size for the former figure is 8-bit and the latter is 16-bit respectively.

When the transfer is from memory to a device, data is valid when $\overline{DTACK}$ is asserted and remains valid until the data strobes are negated. The assertion of $\overline{DTC}$ from the DMAC may be used to latch the data.

When the transfer is from device to memory, data must be valid on the HMCS68000 bus before the DMAC asserts the data strobes. The data strobes are asserted one clock period after $\overline{ACK}$ is asserted. When the DMAC obtains the bus and starts a DMA cycle, the tri-state of the $\overline{OWN}$ line is cancelled a half clock earlier than other control lines. If the DMA Cycle terminates and the DMAC relinquishes the bus, all the control signals get tri-stated a half clock before $\overline{OWN}$. The $\overline{DDIR}$ and $\overline{DBEN}$ lines are not asserted in the single addressing mode. Four clocks cycle is the smallest bus cycle for the transfer from memory to device. Five clocks cycle is the smallest bus cycle for the transfer from device to memory. If the device port size is 8-bit, either $\overline{LDS}$ or $\overline{UDS}$ is asserted. In the single adressing mode, A_8-A_{23} are outputted for only one and a half clock from the beginning of the DMA bus cycle. Therefore, A_8 through A_{23} needs to be latched externally just like in the dual addressing mode.

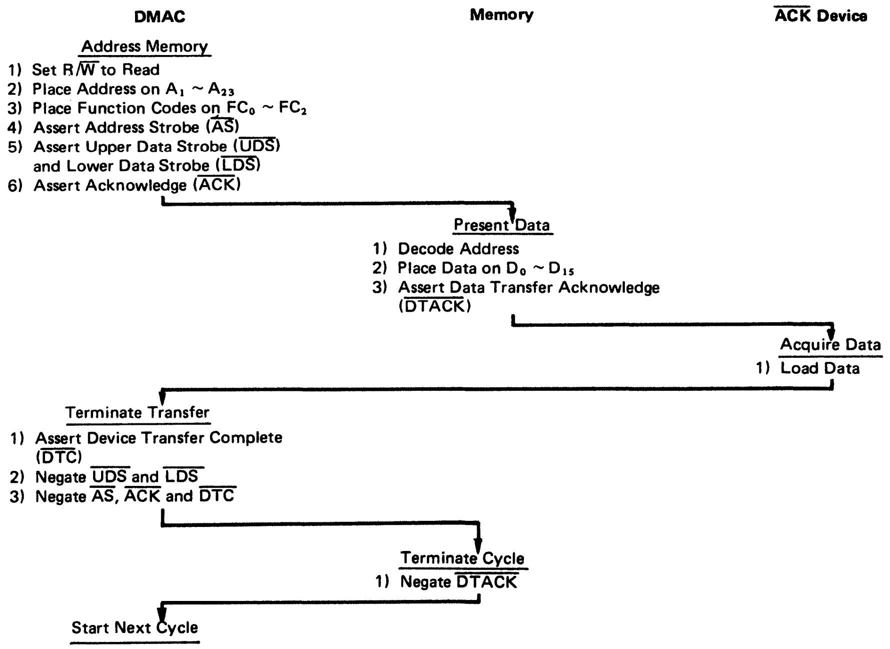

DMAC
Memory
$\overline{ACK}$ Device
Address Memory
1) Set R/$\overline{W}$ to Read
2) Place Address on $A_1 \sim A_{23}$
3) Place Function Codes on $FC_0 \sim FC_2$
4) Assert Address Strobe ($\overline{AS}$)
5) Assert Upper Data Strobe ($\overline{UDS}$) and Lower Data Strobe ($\overline{LDS}$)
6) Assert Acknowledge ($\overline{ACK}$)
Present Data
1) Decode Address
2) Place Data on $D_0 \sim D_{15}$
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)
Acquire Data
1) Load Data
Terminate Transfer
1) Assert Device Transfer Complete ($\overline{DTC}$)
2) Negate $\overline{UDS}$ and $\overline{LDS}$
3) Negate $\overline{AS}$, $\overline{ACK}$ and $\overline{DTC}$
Terminate Cycle
1) Negate $\overline{DTACK}$
Start Next Cycle

Figure 28 Word from Memory to Device with $\overline{ACK}$

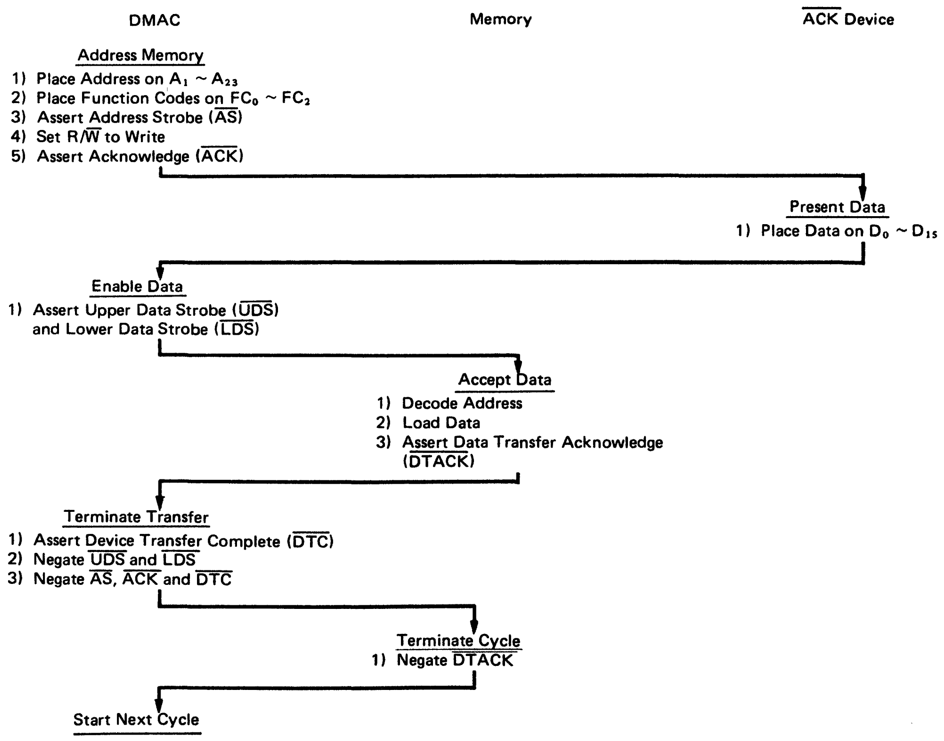

DMAC
Memory
$\overline{ACK}$ Device
Address Memory
1) Place Address on $A_1 \sim A_{23}$
2) Place Function Codes on $FC_0 \sim FC_2$
3) Assert Address Strobe ($\overline{AS}$)
4) Set R/$\overline{W}$ to Write
5) Assert Acknowledge ($\overline{ACK}$)
Present Data
1) Place Data on $D_0 \sim D_{15}$
Enable Data
1) Assert Upper Data Strobe ($\overline{UDS}$) and Lower Data Strobe ($\overline{LDS}$)
Accept Data
1) Decode Address
2) Load Data
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)
Terminate Transfer
1) Assert Device Transfer Complete ($\overline{DTC}$)
2) Negate $\overline{UDS}$ and $\overline{LDS}$
3) Negate $\overline{AS}$, $\overline{ACK}$ and $\overline{DTC}$
Terminate Cycle
1) Negate $\overline{DTACK}$
Start Next Cycle

Figure 29 Word from Device with $\overline{ACK}$ to Memory

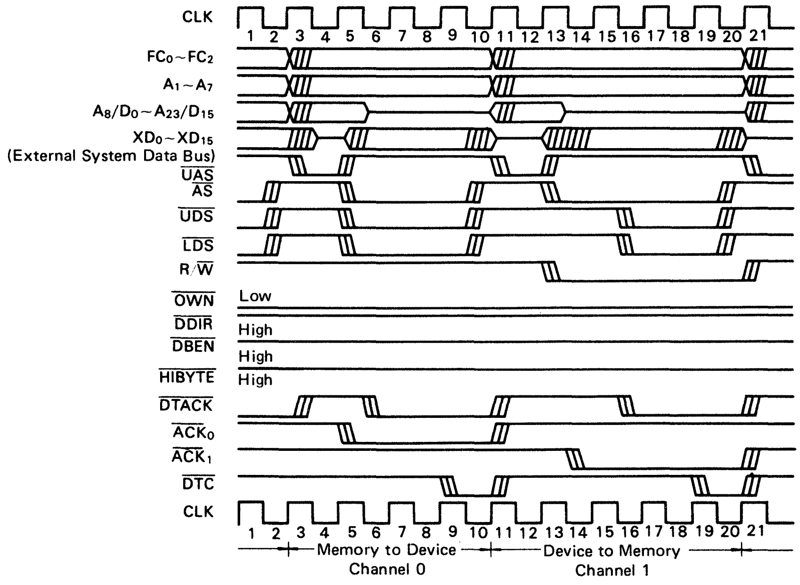


Figure 30 Single Addressing Mode with 16-Bit Devices as Source and Destination (Read-Write Cycles)

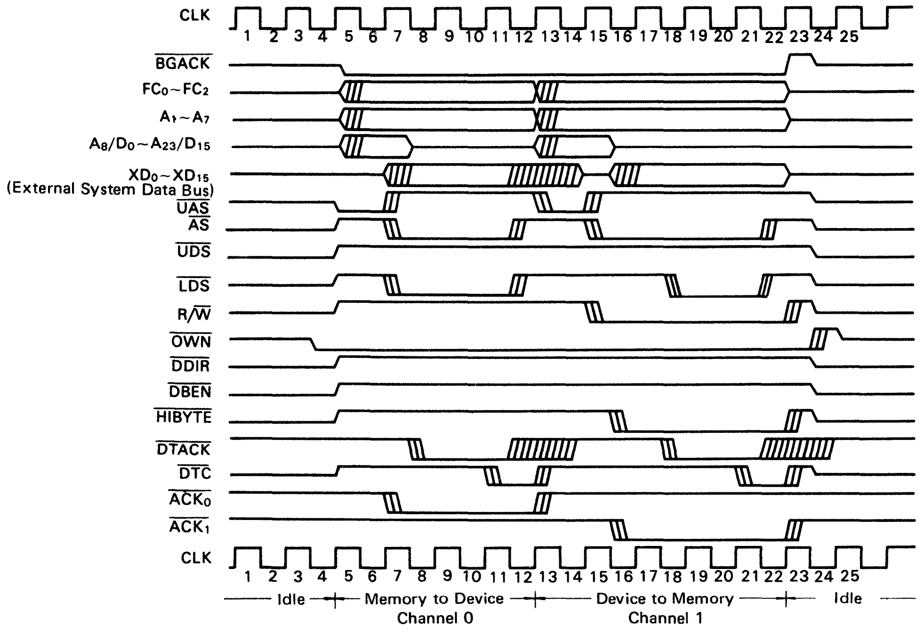


Figure 31 Single Addressing Mode with 8-Bit Device as Source and Destination (Read-Write Cycles)

Device with $\overline{ACK}$ and $\overline{READY}$ Transfers

Under this protocol, the communication between peripheral device and the DMAC is performed using a three signal $\overline{REQ}/\overline{ACK}/\overline{READY}$ handshake. The $\overline{READY}$ input to the DMAC is provided by the $\overline{PCL}$ line. The $\overline{READY}$ line is active low. When a request is generated using the request method programmed in the control registers, the DMAC obtains the bus and asserts $\overline{ACK}$ to notify the device that the transfer is to take place. The DMAC waits for $\overline{READY}$ ($\overline{PCL}$ input), which is a response from the device, in addition to $\overline{DTACK}$ which is a response from memory.

When the DMAC accepts both signals, it terminates the transfer. Refer to Figures 33 and 34 for the flowcharts of the data transfer between memory and the device with $\overline{ACK}$ and $\overline{READY}$. Refer to Figure 35 for the transfer timing of the 8-bit device. When the data transfer is from memory to a device, data is valid from the assertion of $\overline{DTACK}$ to the negation of $\overline{LDS}$ and $\overline{UDS}$. $\overline{DTC}$ is asserted a half clock before $\overline{LDS}$ and $\overline{UDS}$ are negated, so this line may be used for latching the data by the peripheral device. In this case, $\overline{READY}$ ($\overline{PCL}$ input) indicates that the device has received the data. Both $\overline{DTACK}$ and $\overline{READY}$ ($\overline{PCL}$ input) signals are needed for terminating the DMA cycle.

When the data transfer is from the device to memory, data must be valid on the bus before the DMAC asserts $\overline{LDS}$ and $\overline{UDS}$. Therefore, $\overline{READY}$ ($\overline{PCL}$ input) is used as the signal to indicate that the peripheral device has outputted the data on the bus. When the DMAC detects PCL (READY input), then it asserts $\overline{LDS}$ and $\overline{UDS}$. After asserting $\overline{LDS}$ and $\overline{UDS}$, the DMAC terminates the cycle when $\overline{DTACK}$ signal from the memory is detected.

When Array Chain or Link Array Chain is set in Device with $\overline{ACK}$ and $\overline{READY}$ Transfer mode, $\overline{READY}$ input is also necessary during DMA bus cycles for reading the chain information from memory. The circuit as shown in Figure 32 may be used in order to generate $\overline{READY}$ input when reading the chain information from memory.

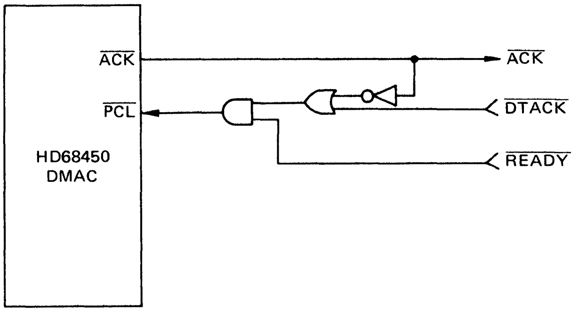
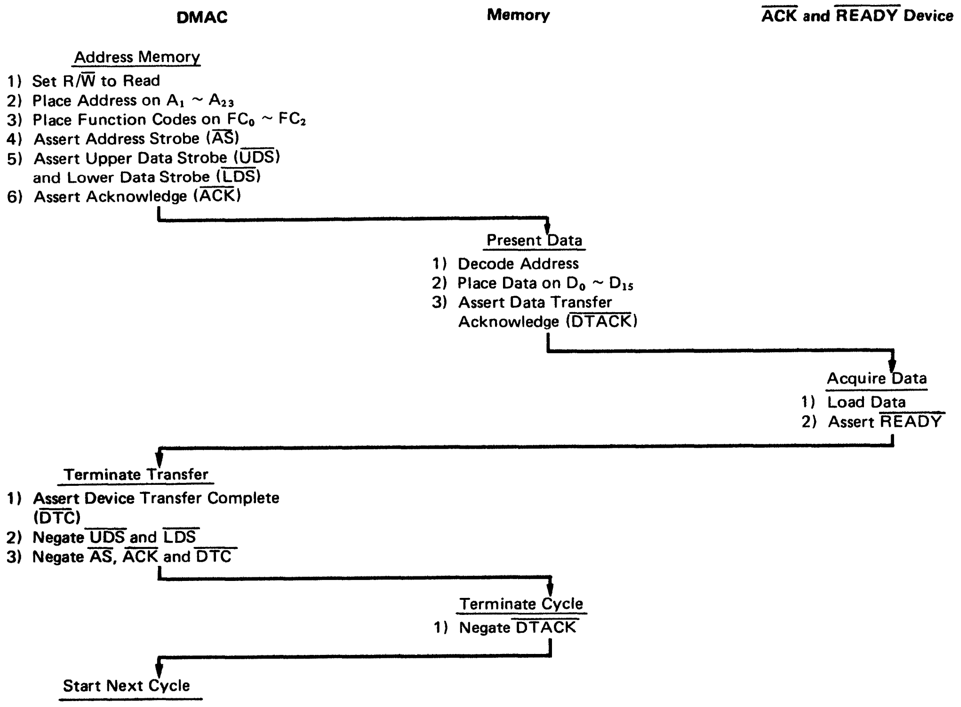


Figure 32 $\overline{READY}$ Circuit When Array or Link Array Chain is set for Device with $\overline{ACK}$ and $\overline{READY}$



DMAC
Memory
ACK and READY Device
Address Memory
1) Set R/W to Read
2) Place Address on A1 ~ A23
3) Place Function Codes on FC0 ~ FC2
4) Assert Address Strobe (AS)
5) Assert Upper Data Strobe (UDS) and Lower Data Strobe (LDS)
6) Assert Acknowledge (ACK)
Present Data
1) Decode Address
2) Place Data on D0 ~ D15
3) Assert Data Transfer Acknowledge (DTACK)
Acquire Data
1) Load Data
2) Assert READY
Terminate Transfer
1) Assert Device Transfer Complete (DTC)
2) Negate UDS and LDS
3) Negate AS, ACK and DTC
Terminate Cycle
1) Negate DTACK
Start Next Cycle

Figure 33 Word from Memory to Device with $\overline{ACK}$ and $\overline{READY}$

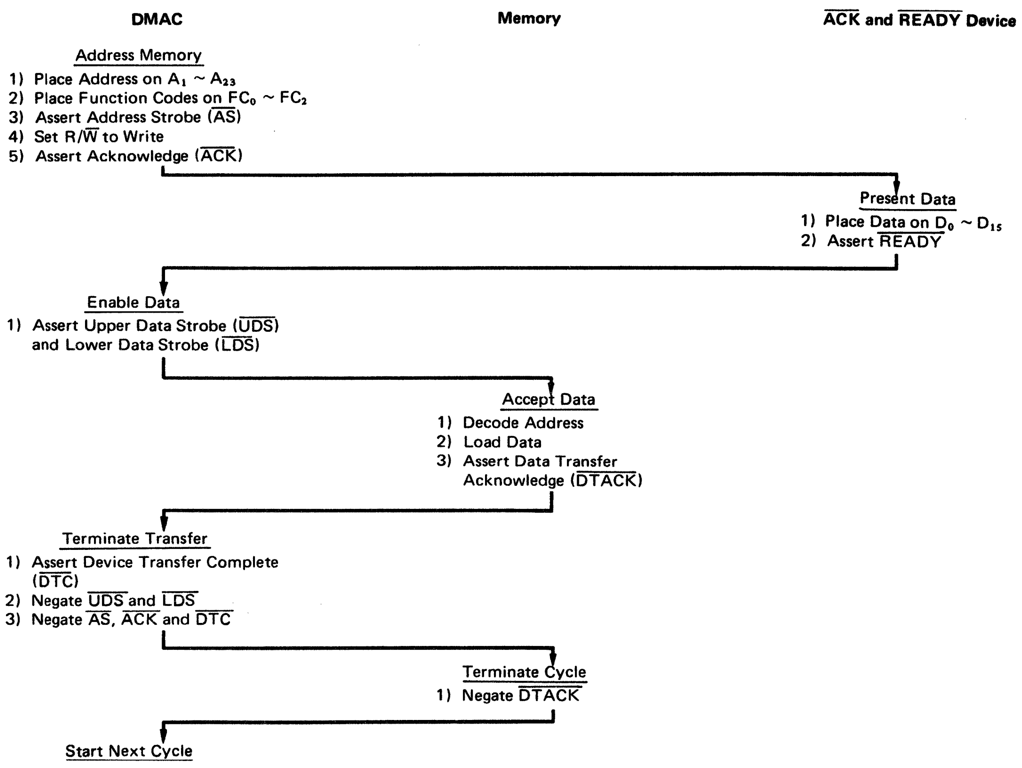

DMAC
Memory
$\overline{ACK}$ and $\overline{READY}$ Device
Address Memory
1) Place Address on $A_1 \sim A_{23}$
2) Place Function Codes on $FC_0 \sim FC_2$
3) Assert Address Strobe ($\overline{AS}$)
4) Set R/$\overline{W}$ to Write
5) Assert Acknowledge ($\overline{ACK}$)
Present Data
1) Place Data on $D_0 \sim D_{15}$
2) Assert $\overline{READY}$
Enable Data
1) Assert Upper Data Strobe ($\overline{UDS}$) and Lower Data Strobe ($\overline{LDS}$)
Accept Data
1) Decode Address
2) Load Data
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)
Terminate Transfer
1) Assert Device Transfer Complete ($\overline{DTC}$)
2) Negate $\overline{UDS}$ and $\overline{LDS}$
3) Negate $\overline{AS}$, $\overline{ACK}$ and $\overline{DTC}$
Terminate Cycle
1) Negate $\overline{DTACK}$
Start Next Cycle

Figure 34 Word from Device with $\overline{ACK}$ and $\overline{READY}$ to Memory

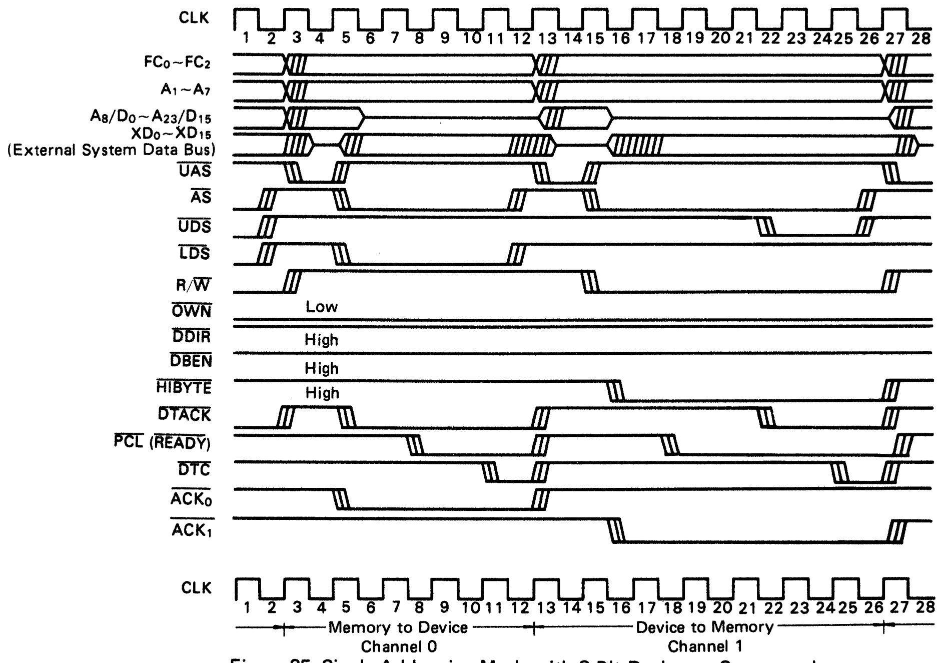


Figure 35 Single Addressing Mode with 8-Bit Devices as Source and Destination with $\overline{PCL}$ Used as a $\overline{READY}$ Input (Read-Write Cycles)

Operands and Addressing

Three factors enter into how the actual data is handled: port size, operand size and address sequencing.

Port Size

The DCR is used to program the device port size.

DPS	Device Port Size
0	8 bit port
1	16 bit port

The port size is the number of bits of data which the device can transfer in a single bus cycle. During a DMAC bus cycle, a 16-bit port transfers 16 bits of data on $D_0 \sim D_{15}$, while an 8-bit port transfers 8 bits of data, either on $D_0 \sim D_7$ or on $D_8 \sim D_{15}$. The memory is always assumed to have a port size of 16.

Operand Size

OCR is used to program the operand size.

SIZE	Operand Size
00	Byte
01	Word
10	Long word
11	(undefined, reserved)

The operand size is the number of bits of data to be transferred to honor a single request. Multiple bus cycles may be required to transfer the operand through the device port. A byte operand consists of 8 bits of data, a word operand consists of 16 bits of data, a long word operand consists of 32 bits of data. The transfer counter counts the number of operands transferred.

Table 2 indicates the combinations supported by the DMAC about the peripheral devices with different port size and operand sizes in the single and dual addressing mode. In the single addressing mode, port size and operand size must be the same. In the dual addressing mode, byte operand cannot be used when the port size is sixteen and the REQG bit is 10 or 11.

Table 2 Operation Combinations

Addressing	Device Type	Port	Operand			REQG bits of OCR
			Byte	Word	Long Word	
Dual	68000, 6800	8	O	O	O	00, 01, 10, 11
Dual	68000, 6800	16	O	O	O	00, 01
Dual	68000, 6800	16	X	O	O	10, 11
Single	with $\overline{ACK}$ or	8	O	X	X	00, 01, 10, 11
	$\overline{ACK}$ & $\overline{READY}$	16	X	O	X	00, 01, 10, 11

O ; enable X ; disable

(3) Address Sequencing

The sequence of addresses generated depends upon the port size, operand size, whether the addresses are to count up, down or not change and whether the transfer is executed in the single addressing mode or the dual addressing mode. The memory address count method and the peripheral device address count method is programmed using the Memory address count (MAC) bit and the Device address count (DAC) bit in the Sequence Control Register (SCR).

(i) Single addressing mode

In the single addressing mode, memory address sequencing is shown in Table 3. If the operand size is byte, the memory address increment is one (1). If the operand size is word, the memory address increment is two (2). If the memory address register does not count, the memory address is unchanged after the transfer.

If the memory address counts up, the increment is added to the memory address; if the memory address counts down, the increment is subtracted from the memory address. The memory address is changed after the operand is transferred.

Table 3 Single Address Sequencing

Port Size	Operand Size	Memory Address Increment		
		+ (increment)	= (unchanged)	- (decrement)
8	Byte	+1	0	-1
16	Word	+2	0	-2

(ii) Dual addressing mode
In the dual addressing mode, the operand size need not match the port size. Thus the transfer of an operand may require several DMA bus cycles. Each DMA bus cycle, between memory and DMAC and between DMAC and the device, is called the operand part and transfers a portion or all of the operand. The addresses of the operand parts are in a linear increasing sequence. The step between the addresses of the operand is two. The size of the operand parts is the minimum of the port size and the operand size. The number of the operand part is the operand size divided by the port size. In the dual addressing mode, memory is regarded as a device whose port size is 16-bits.
If the port size is 16 bits, the operand size is byte, and the request generation method is auto request or auto request at a limited rate, the DMAC packs consecutive transfers. This means that word transfers are made from the associated address with an address increment of two (2). If the initial source address location contains a single byte, the first transfer is a byte transfer to the internal DMAC holding register, and subsequent transfers from the source are word transfers. If the initial destination location contains a single byte, the first transfer is a byte transfer from the internal DMAC holding register, and any remaining byte remains in the holding register. Likewise, if either the final source or destination location contains a single byte, only a byte transfer is done. Packing is not performed if the address does not count; each byte is transferred by a separate access to the same location. The dual address sequencing is shown in Table 4.

Table 4 Dual Address Sequencing

Port Size	Operand Size	Part Size	Operand Part Address	Address Increment		
				+	=	-
8	Byte	Byte	A	+2	0	-2
8	Word	Byte	A, A+2	+4	0	-4
8	Long	Byte	A, A+2, A+4, A+6	+8	0	-8
16	Byte	Pack	A	+P	0	-P
16	Word	Word	A	+2	0	-2
16	Long	Word	A, A+2	+4	0	-4

P = 1 if packing is not done
= 2 if packing is done

Pack = byte if packing is not done
= word if packing is done

An Example of a Dual Address Transfer

This section contains an example of a dual address transfer using Table 4 of Dual-Address Sequencing. The table is reproduced here as Table 5. The transfer mode of this example is the following:

1. Device Port size = 8 bits
2. Operand size = Long Word (32 bits)
3. Memory to Device Transfer
4. Source (Memory) Counts up, Destination (Device) Counts Down
5. Memory Transfer Counter = 2

In this mode, a data transfer from the source (memory) is done according to the 6th row of Table 5, since the port size of the memory is always 16 bits. A data transfer to the destination (device) is done according to the 3rd row of Table 5.
Table 6 shows the data transfer sequence.

The memory map of this example is shown in Table 7. The operand consists of BYTE A through BYTE D in memory of Table 7. Prior to the transfer, MAR and DAR are set to 00000012 and 00000108 respectively. The operand is transferred to the 8 bit port device according to the order of transfer number in Table 6.

Table 5 Dual-Address Sequencing (Table 4)

Row No.	Port Size	Operand Size	Operand Part Size	Operand Part Addresses	Address Increment		
					+	=	-
1	8	BYTE	BYTE	A	+2	0	-2
2	8	WORD	BYTE	A, A+2	+4	0	-4
③	8	LONG	BYTE *4	A, A+2, A+4, A+6 *3 *5 *7 *8	+8	0	-8 *10
4	16	BYTE	PACK (BYTE or WORD)**	A	+P	0	-P
5	16	WORD	WORD	A	+2	0	-2
⑥	16	LONG	WORD *2	A, A+2 *1 *6	+4 *9	0	-4

* Numbers in Table 5 correspond to ones in Table 6 and 7.
** Refer to Address Sequencing on Operand Part Size and PACK.

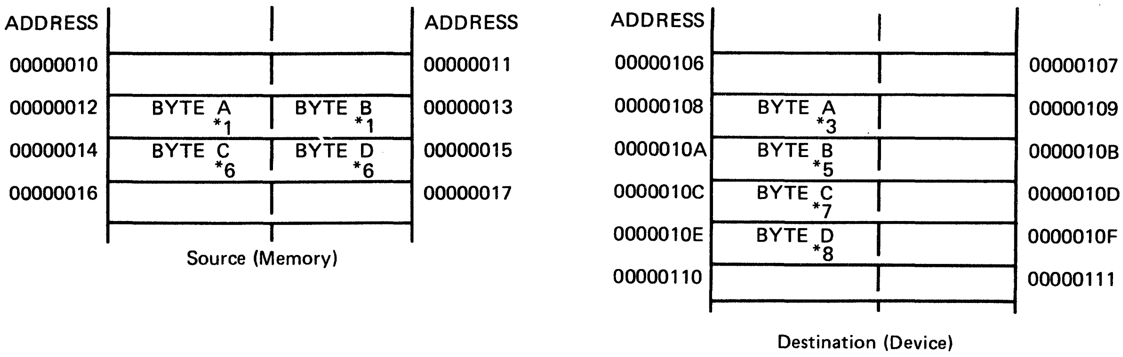
Table 6 An Example of a Data Transfer for One Operand

SRC: Source (Memory), DST Destination (Device), HR: Holding Register (DMAC Internal Reg.)

Transfer No.	Data Transfer	Address Output	Data Size on Bus	DMAC Registers after Transfer		Comment
				MAR	DAR	
0	–	–	–	00000012	00000108	Initial Register Setting
1	SRC→HR	00000012 *1	WORD *2	00000014	00000108	Higher order 16 bits of operand is fetched.
2	HR →DST	00000108 *3	BYTE *4	00000014	0000010A	Higher order 16 bits of operand is transferred.
3	HR →DST	0000010A *5	BYTE *4	00000014	0000010C *10	
4	SRC→HR	00000014 *6	WORD *2	00000016 *9	0000010C	Lower order 16 bits of operand is fetched
5	HR →DST	0000010C *7	BYTE *4	00000016	0000010E	Lower order 16 bits of operand is transferred.
6	HR →DST	0000010E *8	BYTE *4	00000016	00000110 *10	
6′	–	–	–	00000016	00000110	MAR, DAR are pointing the next operand addresses when the transfer is complete.

Mode: Port size = 8, Operand size = Long Word, Memory to Device, Source (Memory) Counts Up, Destination (Device) Counts Down

Table 7 Memory Map for the Example of the Data Transfer



- **Initiation and Control of Channel Operation**

(1) Operation Initiation

To initiate the operation of a channel the STR bit of the CCR is set to start the operation. Setting the STR bit causes the immediate activation of the channel, the channel will be ready to accept requests immediately. The channel initiates the operation by resetting the STR bit and setting the channel active bit in the CSR. Any pending requests are cleared, and the channel is then ready to receive requests for the new operation. If the channel is configured for an illegal operation, the configuration error is signaled, and no channel operation is run. The illegal operations include the selection of any of the options marked "(undefined, reserved)". If the MTC is set to zero in any operation or BTC is set to zero in the array chaining mode, then the count error is signaled and the channel is not activated. The channel cannot be started if any of the ACT, COC, BTC, NDT or ERR bits is set in the CSR. In this case, the channel signals the operation timing error.

(2) Operation Continuation (Continue Mode)

The continue bit (CNT) allows multiple blocks to be transferred in unchained operations. The CNT bit is set in order to continue the current channel operation. If an attempt is made to continue a chained operation, a configuration error is signaled. The base address register and base transfer counter should have been previously initialized.

The continue bit may be set as the channel is started or while the channel is still active. The operation timing error bit is signaled if a continuation is otherwise attempted.

When the memory transfer counter is exhausted and the continue bit of the CCR is set, the DMAC performs a continuation of the channel operation. The base address, base function code, and base transfer count registers are copied into the memory address, memory function code, and memory transfer count registers. The block transfer complete (BTC) bit of the CSR is set, the continue bit is reset, and the channel begins a new block transfer. If the memory transfer counter is loaded with a terminal count, the count error is signaled.

(3) Operation Halting (Halt)

The CCR has a halt bit which allows suspension of the operation of the channel. If this bit is set, a request may still be generated and recognized, but the DMAC does not attempt to acquire the bus or to make transfers for the halted channel. When this bit is reset, the channel resumes operation and services any request that may have been received while the channel was halted. However, in the burst request mode, the transfer request should be kept asserted until the initiation of the first transfer after clearing the halt bit.

(4) Operation Abort by Software (Software Abort)

Setting the software abort bit (SAB) in the CCR allows the current operation of the channel to be aborted. In this case, the ERR bit and the COC bit in the CSR are set and the ACT bit is reset. The error code for the software abort is set in the CER. The SAB bit is designed to be reset if the ERR bit is reset. When the CCR is read, the SAB always reads as zero(0).

(5) Interrupt Enable

The CCR has an interrupt enable bit (INT) which allows the channel to request interrupts. If INT is set, the channel can request interrupts. If it is clear, the channel will not request interrupts.

- **Channel Operation Termination**

As part of the transfer of an operand, the DMAC decrements the memory transfer counter (MTC). If the chaining mode is not used and the CNT bit is not set or the last block is transferred in the chaining mode, the operation of the channel is complete when the last operand transfer is completed and the MTC is zero. The DMAC notifies the peripheral device of the channel completion via the $\overline{\text{DONE}}$ output.

However, in the continue mode, $\overline{\text{DONE}}$ is outputted at the termination of every data block transfer. When the channel operation has been completed, the ACT bit of the CSR is cleared, and the COC bit of the CSR is set.

The occurrence of errors, such as the bus error, during the DMA bus cycle also terminates the channel operation. In this case, the ACT bit in the CSR is cleared, the ERR and the COC bits are set, and at the same time the code corresponding to the error that occurred is set in the CER.

(1) Channel Status Register (CSR)

The channel status register contains the status of the channel. The register, except for ACT and PCS bits, is cleared by writing a one (1) into each bit of the register to be cleared. Those bits positions which contain a zero (0) in the write data remain unaffected. ACT and PCS bits are unaffected by the write operation.

COC

The channel operation complete (COC) bit is set if the channel operation has completed. The COC bit must be cleared in order to start another channel operation. The COC bit is cleared only by writing a one to this bit or resetting the DMAC.

PCS

The peripheral status (PCS) bit reflects the level of the $\overline{\text{PCL}}$ line regardless of its programmed function. If $\overline{\text{PCL}}$ is at "High" level, the PCB bit reads as one. If $\overline{\text{PCL}}$ is at "Low" level, the PCS bit reads as zero. The PCS bit is unaffected by writing to the CSR.

PCT

The peripheral control transition (PCT) bit is set, if a falling edge transition has occurred on the $\overline{\text{PCL}}$ line. (The $\overline{\text{PCL}}$ line must remain at "low" level for at least two clock cycles.) The PCT bit is cleared by writing a one to this bit or resetting the DMAC.

BTC

Block transfer complete (BTC) bit is set when the continue (CNT) bit of CCR is set and the memory transfer counter (MTC) is exhausted. The BTC bit must be cleared before the another continuation is attempted (namely, setting the CNT bit again), otherwise an operation timing error occurs. The BTC bit is cleared by writing a one to this bit or resetting the DMAC.

NDT

Normal device termination (NDT) bit is set when the peripheral device terminates the channel operation by asserting the $\overline{\text{DONE}}$ line while the peripheral device was being acknowledged. The NDT bit is cleared by writing a one to this bit or resetting the DMAC.

ERR

Error (ERR) bit is set if any errors have been signaled. When the ERR bit is set, the code corresponding to the kind of the error that occurred is set in the CER. The ERR bit is cleared by writing a one to this bit or resetting the DMAC.

ACT

The active (ACT) bit is asserted after the STR bit has been set and the channel operation has started. This bit is remains set until the channel operation is terminated. The ACT bit is unaffected by write operations. This bit is cleared by the termination of the channel or resetting the DMAC.

(2) Interrupts

The DMAC can signal the termination of the channel operation by generating an interrupt request. The INT bit of the CCR determines if an interrupt can be generated. The interrupt request is generated by the following condition.

① INT = 1
and
② COC = 1 or BTC = 1 or ERR = 1 or NDT = 1 or PCT = 1
(the $\overline{\text{PCL}}$ line is an interrupt input)

This may be represented as

$$\overline{\text{IRQ}} = \overline{\text{INT} \cdot (\text{COC} + \text{BTC} + \text{ERR} + \text{NDT} + \text{PCT}^*)}$$

(*$\overline{\text{PCL}}$ line is programmed as an interrupt input.)

When the $\overline{\text{IRQ}}$ line is asserted, changing the INT bit from one to zero to one will cause the $\overline{\text{IRQ}}$ output to change from "low" to "high" to "low" again. The $\overline{\text{IRQ}}$ should be negated by clearing the COC, the BTC, the ERR, the NDT and the PCT bits.

If the DMAC receives $\overline{\text{IACK}}$ from the MPU during asserting the $\overline{\text{IRQ}}$, the DMAC provides an interrupt vector. If multiple channels have interrupt requests, the determination of which channel presents its interrupt vector is made using the same priority scheme defined for the channel operations.

The bus cycle in which the DMAC provides the interrupt vector when receiving an $\overline{\text{IACK}}$ from the MPU is called the interrupt acknowledge cycle. The interrupt vector returned to the MPU comes from either the normal or the error interrupt vector register. The normal interrupt register is used unless the ERR bit of CSR is set, in which case the error interrupt vector register is used. The content of the interrupt vector register is placed on $D_0 \sim D_7$, and $\overline{\text{DTACK}}$ is asserted to indicate that the vector is on the data bus. If a reset occurs, all interrupt vector registers are set to $0F (binary 00001111), the value of the uninitialized interrupt vector. The timing of the interrupt acknowledge cycle is shown in Figure 36. The HD68000 MPU outputs the interrupt level into A_1-A_3 and A_4-A_7 is held "high" during the interrupt acknowledge cycle, but the HD68450 DMAC ignores these signals.

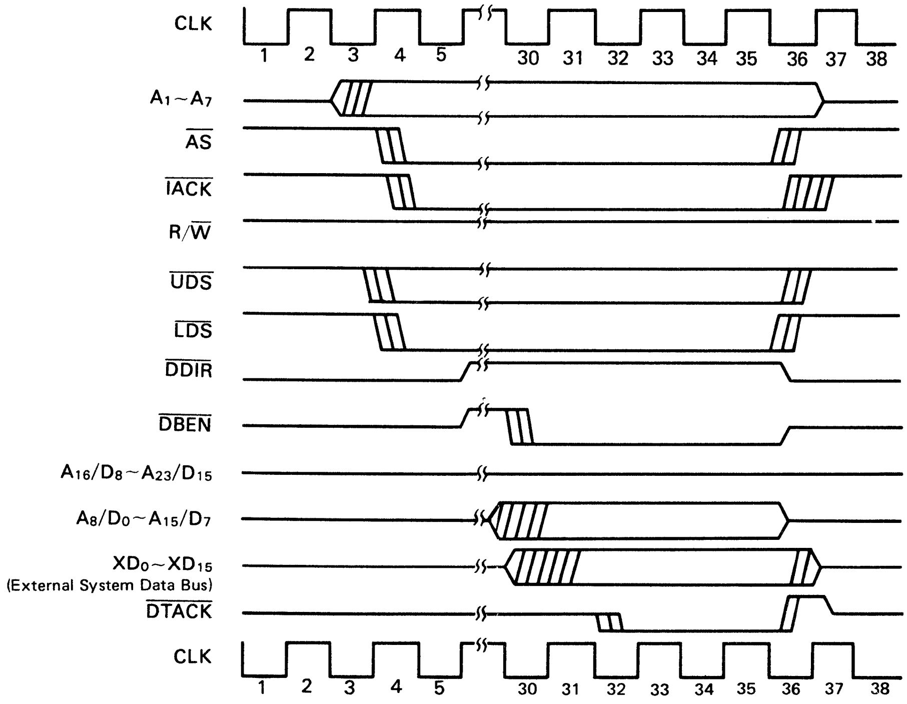


Figure 36 MPU $\overline{\text{IACK}}$ Cycle to DMAC

(3) Multiple Data Block Transfer Operation

When the memory transfer counter (MTC) is exhausted, the channel operation still continues if the channel is set to the array chaining mode or the linked array chaining mode and the chain is not exhausted. The channel operation also continues if the continue bit (CNT) of the CCR is set. The DMAC provides the initialization of the memory address register and the memory transfer counter in these cases so that the DMAC can transfer the multiple blocks.

Continued Operation

The continued operation is described in the Initiation and the Control of the Channel Operation section.

Array Chaining

This type of chaining uses an array in memory consisting of memory addresses and transfer counts. Each entry in the array is six bytes long and, consists of four bytes of address followed by two bytes of transfer count. The beginning address of this array is in the base address register, and the number of entries in the array is in the base transfer counter. Before starting any block transfers, the DMAC fetches the entry currently pointed to by the base address register. The address information is placed in the memory address register, and the count information is placed in the memory transfer counter. As each chaining entry is fetched, the base transfer counter is decremented by one. After the chaining entry is fetched, the base address register is incremented to point the next entry. When the base transfer counter reaches a terminal count of zero, the chain is exhausted, and the entry just fetched determines the last block of the channel operation.

An example of the array chaining mode operation and the memory format for supporting for array chaining is shown in Figure 37. The array must start at an even address, or the entry fetch results is an address error. If a terminal count is loaded into the memory transfer counter or the base transfer counter, the count error is signaled. Since the base registers may be read by the MPU, appropriate error recovery information is available should the DMAC encounter an error anywhere in the chain. Contents of the BFC is outputted as the function code when the DMAC is accessing the memory using the base address register. The value of the function code registers are unchanged in the array chaining operation.

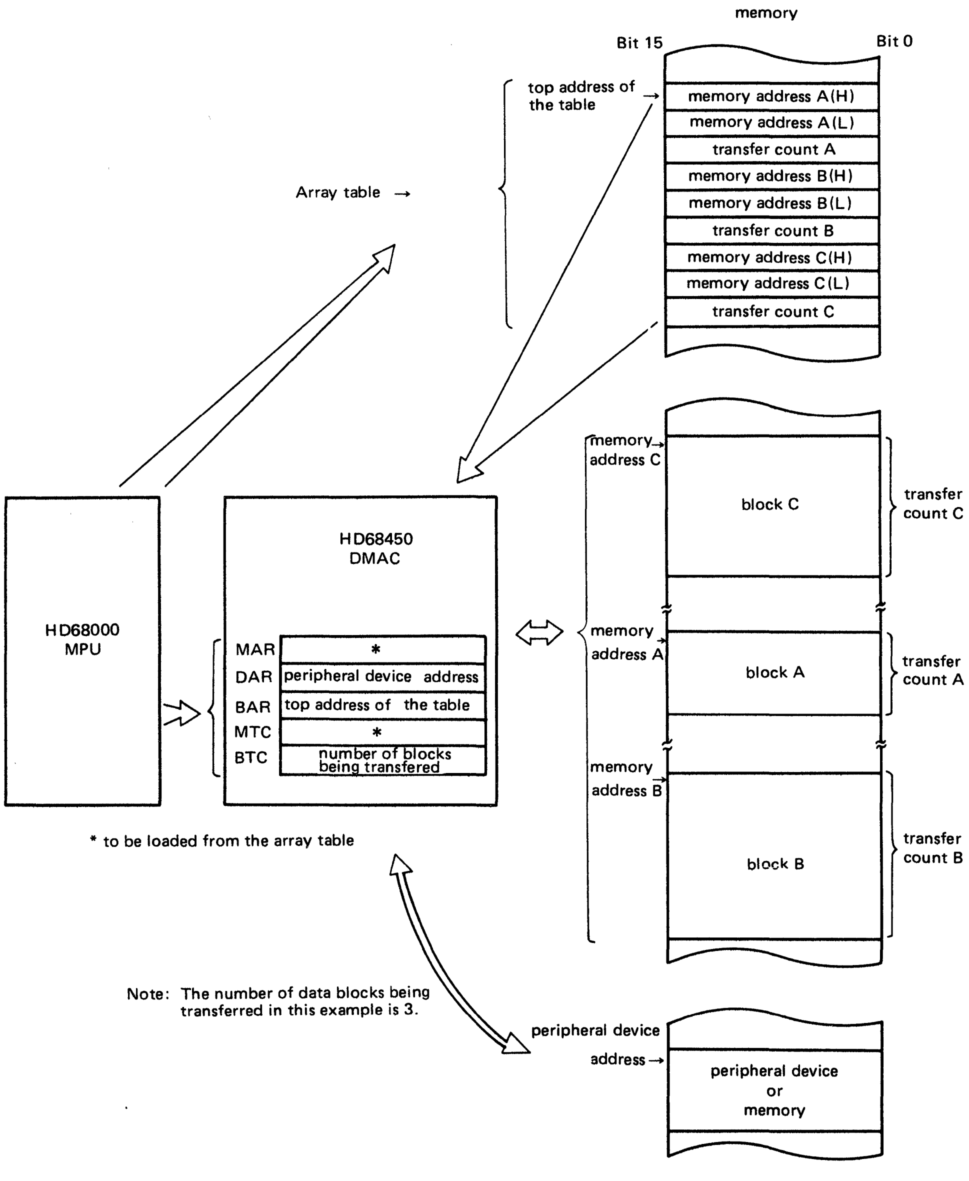


Figure 37 Transfer Example of the Array Chaining Mode

Linked Array Chaining

This type of chaining uses a list in memory consisting of memory address, transfer counts, and link addresses. Each entry in the chain list is ten bytes long, and consists of four bytes of memory address, two bytes of transfer count and four bytes of link address. The address of the first entry in the list is in the base address register, and the base transfer counter is unused. Before starting any block transfers, the DMAC fetches the entry currently pointed to by the base address register. The address information is placed in the memory address register, the count information is placed in the memory transfer counter, and the link address replaces the current contents of the base address register. The channel then begins a new block transfer. As each chaining entry is fetched, the update base address register is examined for the terminal link which has all 32 bits equal to zero. When the new base address is the terminal address, the chain is exhausted, and the entry just fetched determines the last block of the channel operation.

An example of the linked array chaining mode operation and the memory format for supporting it is shown is Figure 38.

In Figure 38, the DMAC transfers data blocks in the order of Block A, Block B, and Block C. In the linked array chaining

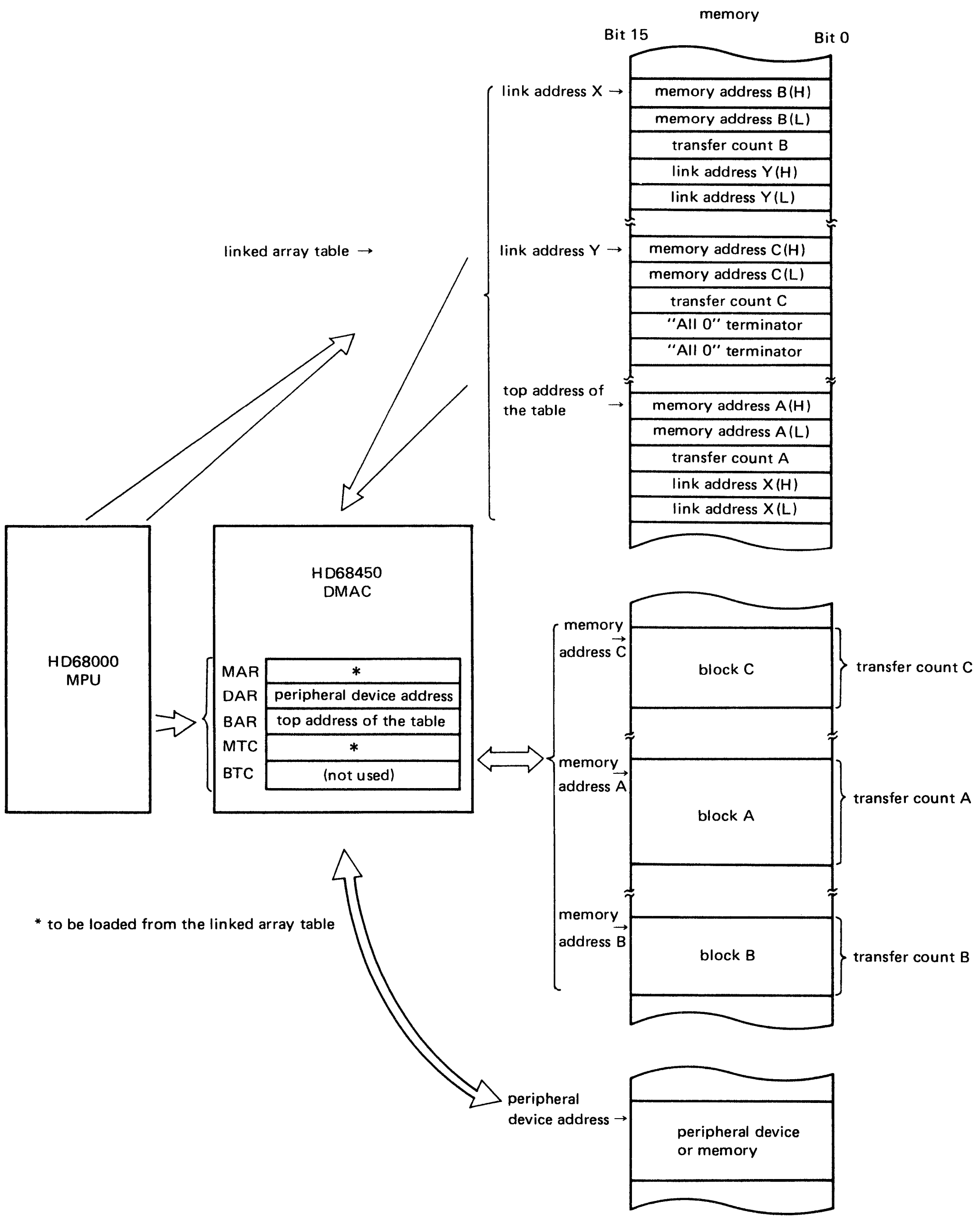


Figure 38 Transfer Example of the Linked Array Chaining Mode

mode, the BTC is not used. When the DMAC refers to the linked array table, the value of the BFC is outputted as the function code. The values of the function code registers are unchanged by the linked array chaining operation.

This type of chaining allows entries to be easily removed or inserted without having to reorganize data within the chain. Since the end of the chain is indicated by a terminal link, the number of entries in the array need not be specified to the DMAC.

The linked array table must start at an even address in the linked array chaining mode. Starting the table at an odd address results in an address error. If "0" is initially loaded to the MTC, the count error is signaled. Because the MPU can read all of the DMAC registers, all necessary error recovery information is available to the operating system.

The comparision of both chaining modes is shown in Table 8.

Table 8 Chaining Mode Address/Count Information

Chaining Mode	Base Address Register	Base Transfer Counter	Completed When
Array Chaining	address of the array table	number of data blocks being transferred	Base Transfer Count = 0
Linked Array Chaining	address of the linked array table	(unused)	Linked Address = 0

(4) Bus Exception Conditions

The DMAC has three lines for inputting bus exception conditions called $\overline{BEC_0}$, $\overline{BEC_1}$, and $\overline{BEC_2}$. The priority encoder can be used to generate these signals externally. These lines are encoded as shown in Table 9.

Table 9

$\overline{BEC_2}$	$\overline{BEC_1}$	$\overline{BEC_0}$	Exception Condition
1	1	1	No exception condition
1	1	0	Halt
1	0	1	Bus error
1	0	0	Retry
0	1	1	Relinquish bus and retry
0	1	0	(undefined, reserved)
0	0	1	(undefined, reserved)
0	0	0	Reset

In order to guarantee, reliable decoding, the DMAC verifies that the incoming code has been statable for two DMAC clock cycles before acting on it. The DMAC picks up $\overline{BEC_0}$-$\overline{BEC_2}$ at the rising edge of the clock. If $\overline{BEC_0}$-$\overline{BEC_2}$ is asserted to the undefined code, the operation of the DMAC does not proceed. For example, when the DMAC is waiting for $\overline{DTACK}$, inputting $\overline{DTACK}$ does not result in the termination of the cycle if $\overline{BEC_0}$-$\overline{BEC_2}$ is asserted to the undefined code. In addition, when the transfer request is received, $\overline{BR}$ is not asserted if the $\overline{BEC_0}$-$\overline{BEC_2}$ is not set to no exception condition.

If exception condition, except for HALT, is inputted during the DMA bus cycle prior to, or in coincidence with $\overline{DTACK}$, the DMAC terminates the current channel operation immediately. Here coincident means meeting the same set up requirements for the same sampling edge of the clock. If a bus exception condition exists, the DMAC does not generate any bus cycles until it is removed. However, the DMAC still recognizes requests.

Halt

The timing diagram of halt is shown in Figure 39. This diagram shows halt being generated during a read cycle from the 68000 compatible device in the dual addressing mode. If the halt exception is asserted during a DMA bus cycle, the DMAC does not terminate the bus cycle immediately. The DMAC waits for the assertion of $\overline{DTACK}$ before terminating the bus cycle so that the bus cycle is completed normally. In the halted state, the DMAC puts all the control signals to high impedance and relinquishes the bus to the MPU. The DMAC does not output the $\overline{BR}$ until halt exception is negated. When halt exception is negated, the DMAC acquires the bus again and proceeds the DMA operation. In order to insure a halt exception operation, the $\overline{BEC}$ lines must be set to halt at least until the assertion of $\overline{DTC}$.

If the DMAC has the bus, but is not executing any bus cycle, the DMAC relinquishes the bus as soon as halt exception is asserted.

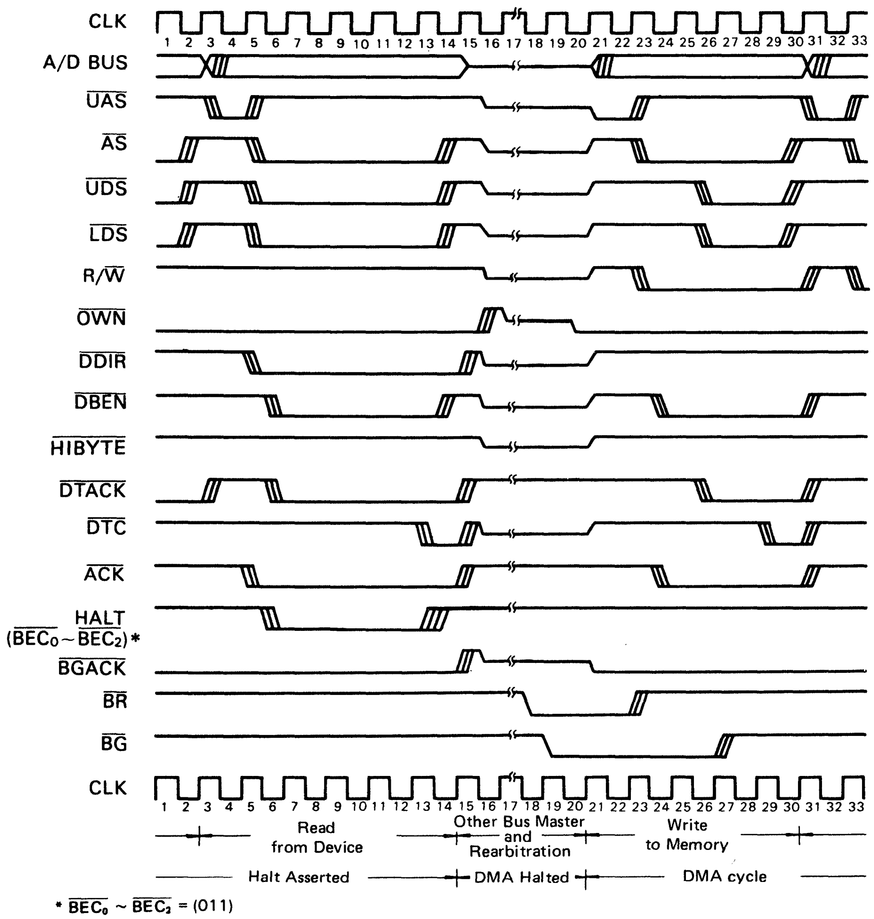
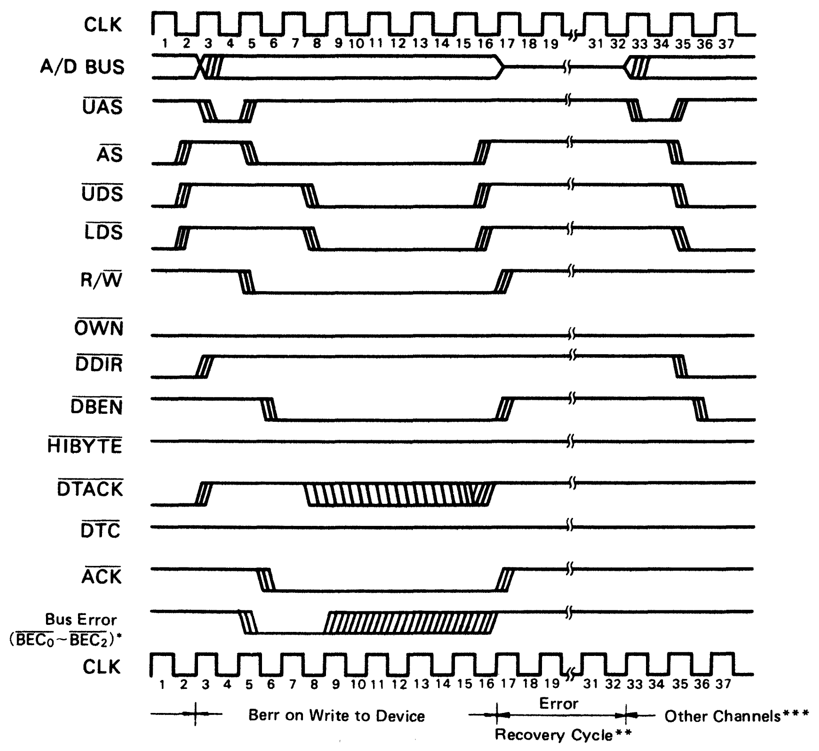


Figure 39 Halt Operation

Bus Error

The bus error exception is generated by external circuitry to indicate the current transfer cannot be successfully completed and is to be aborted. The recognition of this exception during a DMAC bus cycle signals the internal bus error condition for the channel for which the current bus cycle is being run. As soon as the DMAC recognizes the bus error exception, the DMAC immediately terminates the bus cycle and proceeds to the error recovery cycle. In this cycle, the DMAC adjusts the values of the MAR, the DAR, the MTC and the BTC to the values when the bus error exception occurred. 25 clocks are required for the error recovery cycle in the single addressing mode and in the read cycle of the dual addressing mode. 29 clocks are required in the write cycle of the dual addressing mode. If the DMAC does not have any transfer request in the other channels after the error recovery cycle, the DMAC relinquishes the bus.

The diagram of the bus error timing is shown in Figure 40.



* $\overline{BEC_0}$-$\overline{BEC_2}$ = (101)

** In the single addressing mode and in the read cycle of the dual addressing mode: 25 clocks
In the write cycle of the dual addressing mode: 29 clocks

*** The DMAC keeps the bus because the other channels have requests pending. If other channels do not have requests, the DMAC relinquishes the bus after the error recovery cycle.

Figure 40 Bus Error Operation

Retry

The retry exception causes the DMAC to terminate the present operation and retry that operation when retry is removed, and thus will not honor any requests until it is removed. However, the DMAC still recognizes requests. The retry timing is shown in Figure 41.

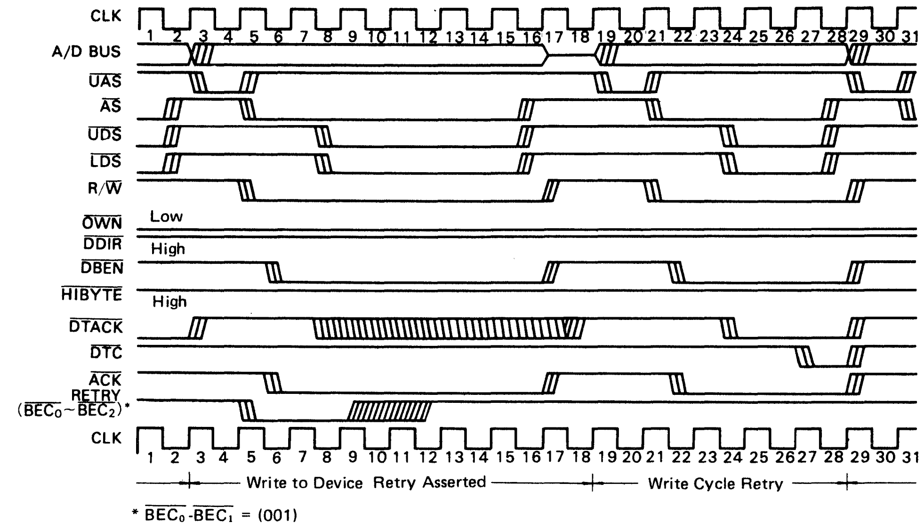


Figure 41 Retry Operation

Relinquish and Retry (R&R)

The relinquish and retry exception causes the DMAC to relinquish the bus and three-state all bus master controls and when the exception is removed, rearbitrate for the bus to retry the previous operation.

The diagram of the relinquish and retry timing is shown in Figure 42.

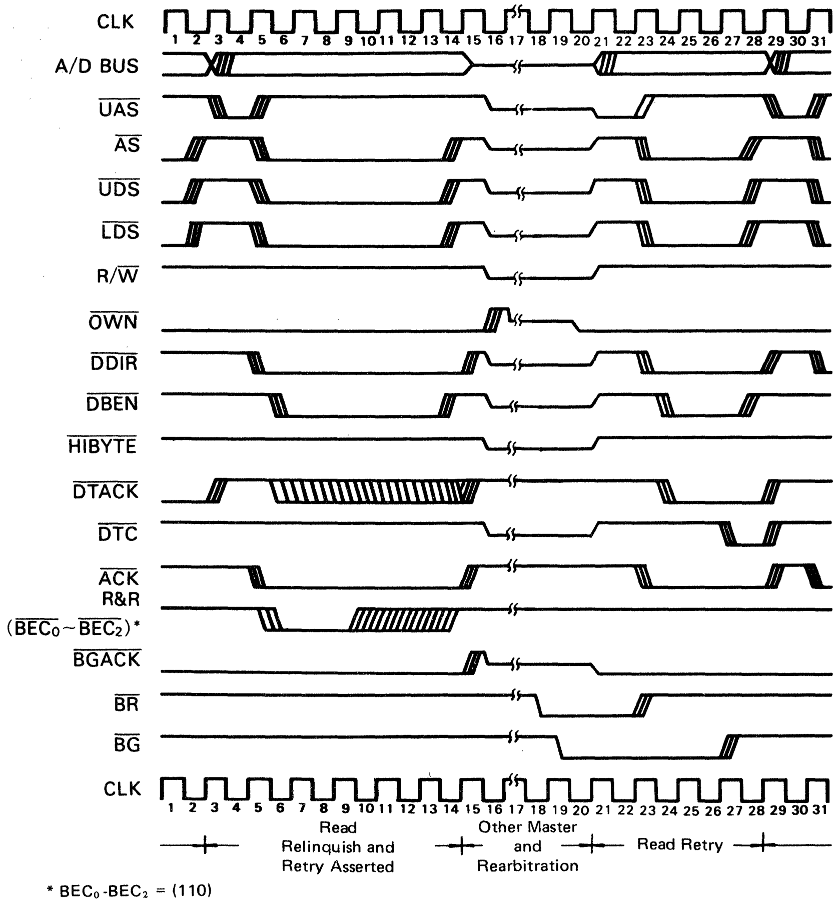


Figure 42 Relinquish and Retry Operation

Reset

The reset provides a means of resetting and initializing the DMAC. If the DMAC is bus master when the reset is asserted, the DMAC relinquishes the bus. Reset clears GCR, DCR, OCR, SCR, CCR, CSR, CPR, and CER for all channels. The NIV and the EIV are all set to $(0F)_{16}$, which is the uninitialized interrupt vector number for the HD68000 MPU. MTC, MAR, DAR, BTC, BAR, MFC, DFC, and BFC are not affected. In order to insure a reset, $\overline{BEC_0}$ ~ $\overline{BEC_2}$ must be kept at "Low" level for at least ten clocks.

(5) Error Conditions

When an error is signaled on a channel, all activity on that channel is stopped. The ACT bit of the CSR is cleared, and the COC bit is set. The ERR bit of the CSR is set, and the error code is indicated in the CER. All pending operations are cleared, so that both the STR and CNT bits of CCR are cleared.

Enumerated below are the error signals and their sources.

(a) Configuration Error – This error occurs if the STR bit is set in the following cases.
  (i) the CNT bit is set at the same time STR bit in the chaining mode.
  (ii) DTYP specifies a single addressing mode, and the device port size is not the same as the operand size.
  (iii) DTYP specifies a dual addressing mode, DPS is 16 bits, SIZE is 8 bits and REQG is "10" or "11".
  (iv) an undefined configuration is set in the registers. The undefined configurations are: XRM = 01, MAC = 11, DAC = 11, CHAIN = 01, and SIZE = 11.

(b) Operation Timing Error – An operation timing error occurs in the following cases:
  (i) when the CNT bit is set after the ACT bit has been set by the DMAC in the chaining mode, or when the STR and the ACT bits are not set.
  (ii) the STR bit is set when ACT, COC, BTC, NDT or ERR is set.
  (iii) an attempt to write to the DCR, OCR, SCR, MAR, DAR, MTC, MFC, or DFC is made when the STR bit or the ACT bit is set.
  (iv) an attempt to set the CNT bit is made when the BTC and the ACT bits are set.

(c) Address Error – An address error occurs in the following cases:
  (i) an odd address is set for word or long word operands.
  (ii) $\overline{CS}$ or $\overline{IACK}$ is asserted during the DMA bus cycle.

(d) Bus Error – Bus error occurs when a bus error exception is signaled during a DMA bus cycle.

(e) Count Error – A count error occurs in the following cases:
   (i) The STR bit is set when zero is set in the MTC and the MTC and the chaining mode is not used.
   (ii) the STR bit is set when zero is set in BTC for the array chaining mode.
   (iii) zero is loaded from memory to the BTC or the MTC in the chaining modes or the continue mode.

(f) External Abort – External abort occurs if an abort is asserted by the external circuitry when the $\overline{\text{PCL}}$ line is configured as an abort input and the STR or the ACT bit is set.

(g) Software abort – Software abort occurs if the SAB bit is set when the STR or the ACT bit is set.

Error Recovery Procedures

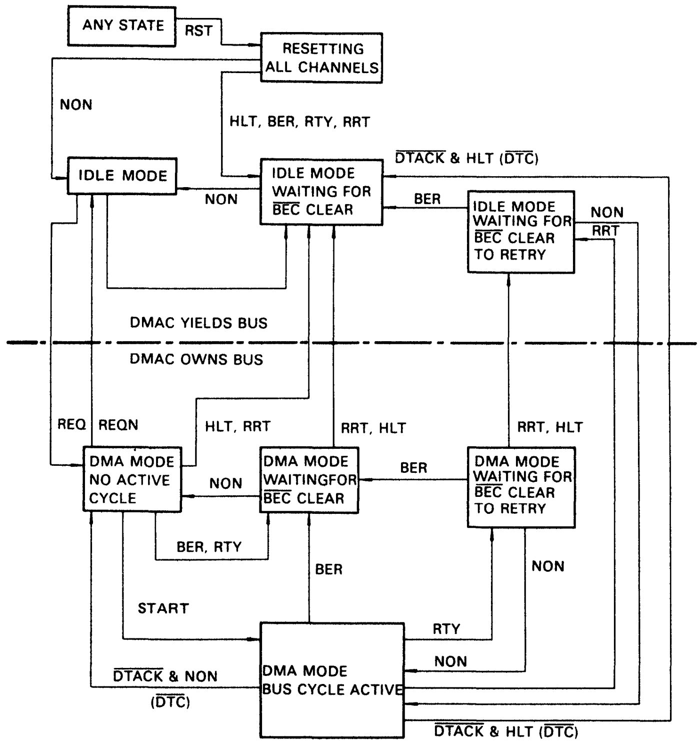
If an error occurs during a DMA transfer, appropriate information is available to the operating system (OS) to allow a software failure recovery operation. The operating system must be able to determine how much data was transferred, where the data was transferred to, an what type of error occurred.

The information available to the operating system consists of the present value of the Memory Address, Device Address and Base Address Registers, the Memory Transfer and Base Transfer Counters, the channel status register, the channel error register, and the channel control register. After the successful completion of any transfer, the memory and device address registers points to the location of the next operand to be transferred and the memory transfer counter contains the number of operands yet to be transferred. If an error occurs during a transfer, that transfer has not completed and the registers contain the values they had before the transfer was attempted. If the channel operation uses chaining, the Base Address Register points to the next chain entry to be serviced, unless the termination occurred while attempting to fetch an entry in the chain. In that case, the Base Address Register points to the entry being fetched. However, in the case of external abort, there are cases in which the previous values are not recovered.

Bus Exception Operating Flow

The bus exception operating flow in the case of multiple exception conditions occurring continuously in sequence is shown in Figure 43. Note that the DMAC can receive and execute the next exception condition. For example, if the retry exception occurs, and next the relinquish and retry exception occurs while the DMAC is waiting for the retry condition to be cleared, the DMAC relinquishes the bus and waits for the exception condition to be cleared. If a bus error occurs during this period, the DMAC executes the bus error exception operation.

The flow diagram of the normal operation without exception operation or errors is shown in Figure 44.


ANY STATE
RST
RESETTING ALL CHANNELS
NON
HLT, BER, RTY, RRT
IDLE MODE
NON
IDLE MODE WAITING FOR $\overline{\text{BEC}}$ CLEAR
$\overline{\text{DTACK}}$ & HLT ($\overline{\text{DTC}}$)
BER
IDLE MODE WAITING FOR $\overline{\text{BEC}}$ CLEAR TO RETRY
NON
RRT
DMAC YIELDS BUS
DMAC OWNS BUS
REQ
REQN
HLT, RRT
RRT, HLT
RRT, HLT
DMA MODE NO ACTIVE CYCLE
NON
DMA MODE WAITINGFOR $\overline{\text{BEC}}$ CLEAR
BER
DMA MODE WAITING FOR $\overline{\text{BEC}}$ CLEAR TO RETRY
BER, RTY
BER
NON
START
RTY
NON
$\overline{\text{DTACK}}$ & NON ($\overline{\text{DTC}}$)
DMA MODE BUS CYCLE ACTIVE
$\overline{\text{DTACK}}$ & HLT ($\overline{\text{DTC}}$)

RST	: reset
NON	: no exception
HLT	: halt
BER	: bus error
RTY	: retry
RRT	: relinquish and retry
REQ	: external request
REQN	: no external request
START	: bus cycle start
$\overline{\text{DTACK}}$	: $\overline{\text{DTACK}}$ signal asserted
$\overline{\text{DTC}}$	: $\overline{\text{DTC}}$ signal asserted

Figure 43 Bus Exception Flow Diagram

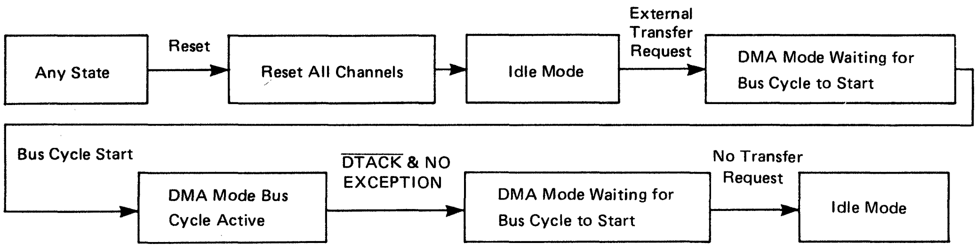

Any State
Reset
Reset All Channels
Idle Mode
External Transfer Request
DMA Mode Waiting for Bus Cycle to Start
Bus Cycle Start
DMA Mode Bus Cycle Active
DTACK & NO EXCEPTION
DMA Mode Waiting for Bus Cycle to Start
No Transfer Request
Idle Mode

Figure 44 Flow of Normal Operation Without Exception or Error Condition

- **Channel Priorities**

Each channel has a priority level, which is determined by the contents of the Channel Priority Register (CPR). The priority of a channel is a number from 0 to 3, with 0 being the highest priority level. When multiple requests are pending at the DMAC, the channel with the highest priority receives first service. The priority of a channel is independent of the device protocol or the request mechanism for that channel. If there are several requesting channels at the highest priority level, a round-robin resolution is used, that is, as long as these channels continue to have requests, the DMAC does operand transfers in rotation.

Resetting the DMAC puts the priority level of all channels to "0", the highest priority level.

■ APPLICATIONS INFORMATION

Examples of how to interface HD68450 to a HD68000 based system are shown in Figure 45 and Figure 46.

Figure 45 shows an example of how to demultiplex the address/data bus. $\overline{\text{OWN}}$ and $\overline{\text{UAS}}$ are used to control 74LS373 for latching the address. $\overline{\text{DBEN}}$ and $\overline{\text{DDIR}}$ are used to control the bi-directional buffer 74LS245.

Figure 46 shows an example of inter-device connection in the HMCS68000 system.

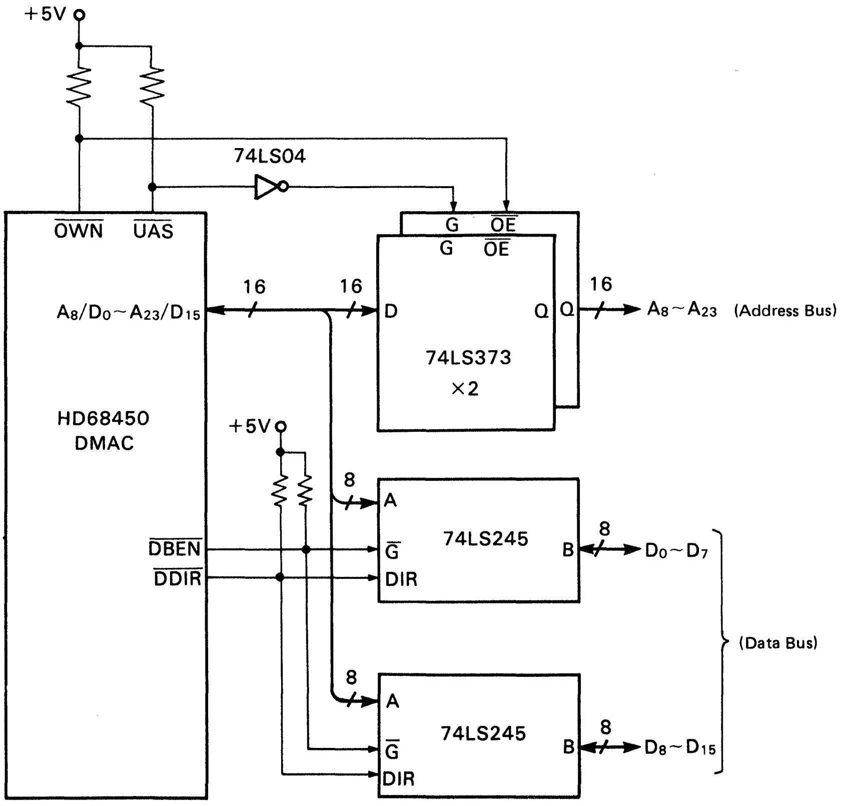
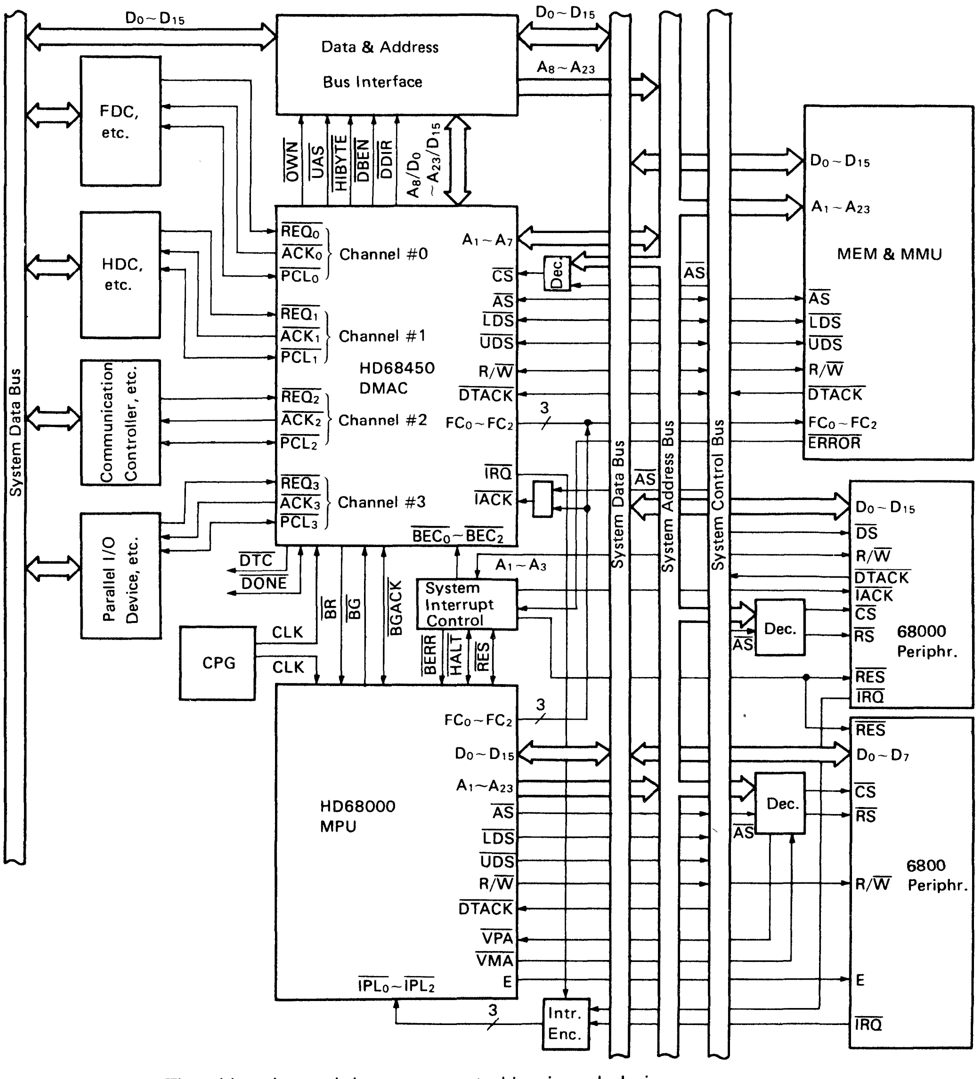


Figure 45 An Example of the Demultiplexed Address Data Bus



The address bus and the system control bus in each device are omitted in this Figure.

Figure 46 An Example of Inter-device Connection in the HMCS68000 System

■ ATTENTION ON USAGE

(1) How to interface various 6800 type peripheral devices to the DMAC based system.

When the DMAC is reading data from the 6800 device, the DMAC latches the data when $\overline{DTC}$ is asserted and not at the falling edge of E clock. The 74LS373 need to be provided externally as shown in Figure 47 so that the data from the 6800 device can be held on the bus for a large period of time until the DMAC can latch the correct data.

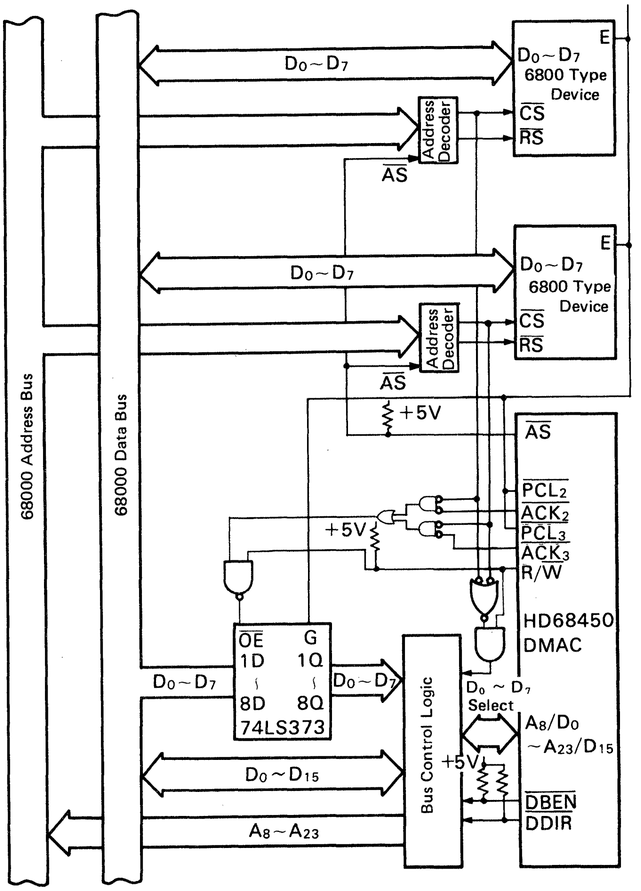


Figure 47 An Example of Connection with 6800 type Peripheral Devices (channel 2 and 3 are used)

(2) When"external abort"is inputted during the $\overline{DONE}$ input cycle

When the transfer direction is from the peripheral device to memory and $\overline{PCL}$ signal is set to the external abort input mode in the dual addressing mode, the external abort will be ignored during the subsequent write cycle from the DMAC's internal holding register to memory if $\overline{DONE}$ is inputted during the read cycle from the peripheral device to the DMAC's internal holding register.

In this case, the channel status register (CSR) and the channel error register (CER) show the normal termination caused by $\overline{DONE}$ Input. The user is able to examine the PCT bit and the ERR bit in order to detect the external abort inputted at the timing described above. If PCT = 1, ERR = 0, and NDT = 1, then an external abort has occurred.

(3) Multiple Errors

The DMAC will log the first error encountered in the channel error resister. If an error is pending in the error register and another error is encountered the second error will not be logged. Even though the second error is not logged in the CER, it will still be recognized internally and the channel will not start.

(4) The use of thick wiring is recommended between Vss of the HD68450 and the ground of the circuit board. When a socket is used to install the DMAC on the board, please make sure that the contact of the Vss pins are made well.

PRECAUTIONS:

1. Extra Data Transfer in the Burst Mode

In certain conditions when two or more channels are active and the $\overline{REQ}$ signal for the channel which is transferring in burst mode has negated, the transfer operation will terminate one data transfer later than specified in the data sheet. The condition on which this occurs is shown in Figure 2. Problems may occur in applications that need to control exact data transfer count using the $\overline{REQ}$ line in the burst mode.

(Countermeasure)

When switching the channel of operation using the burst request signals, negate the $\overline{REQ}$ signal within the period bounded by (3) and (4) in Figure 48. ($\overline{DTC}$ falling edge may be used for obtaining the timing for the negation of $\overline{REQ}$.)

Caution must be taken when this countermeasure is used since this external circuit will not be compatible with the next mask version which will have this anomaly fixed.

NOTE 1: If transfer request is asserted in channel 1, before (1) which is 1 clock before $\overline{DTC}$ assertion of channel 0, the next bus cycle should be the bus cycle for channel 1 according to the data sheet. However, the current DMAC transfers one more data for channel 0 from 13th clock as shown above, before it changes to channel 1.

NOTE 2: If channel 1 has higher priority than channel 0, then NO extra data is transferred even if request for channel 1 is asserted before (2). In this case, data transfer for channel 1 starts from the 13th clock as specified in the data sheet.

*The timing in which one extra data is transferred in the burst mode (the case for changing from channel 0 to channel 1).

2. One Byte of Transfer Data is Left in the DMAC

When the DMAC is set to dual addressing mode, port size 8 bits, external request mode, and data transfer from peripheral device to memory, the last byte of the transfer may be left inside the DMAC's internal register without being transferred to memory if the transfer is stopped before the transfer count is exhausted. The last byte that is left inside the DMAC becomes inaccessible by the MPU.

In this mode, the DMAC transfers data repeating the following bus cycles:

(1) READ BYTE
(Byte is read from the peripheral device to DMAC)

(2) READ BYTE
(Byte is read from the peripheral device to DMAC)

(3) WRITE WORD
(Word is written to memory from DMAC)

If the transfer is terminated after (1) READ BYTE (see NOTE*), then the byte data that was ready by (1) READ BYTE bus cycle is not written to memory and is left inside the DMAC's internal holding register. The DMAC's internal holding register cannot be accessed by the MPU, so that it becomes "lost."

This will not occur when single addressing mode is used. So, please use the single addressing mode when the transfer needs to be terminated before the transfer is exhausted.

Note:*The methods to terminate the transfer operation before the transfer counter becomes zero are (1) assert external abort using the PCL, (2) set the SAB bit to cause software abort.

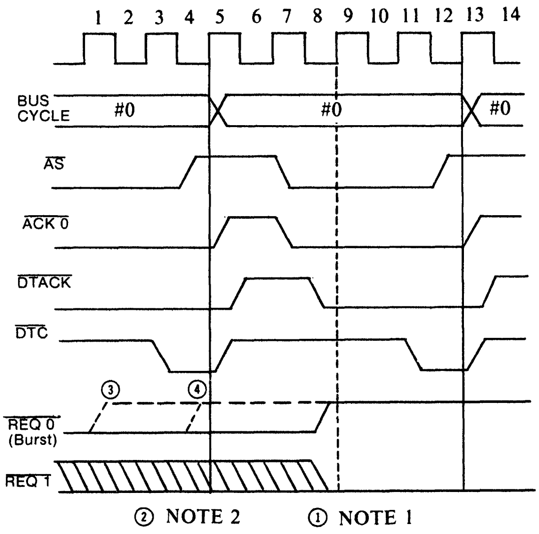


Figure 48. Extra Data Transfer in the Burst Mode*

HD68000 (HD68000-4, HD68000-6, HD68000-8, HD68000-10, HD68000-12) HD68000Y (HD68000Y4, HD68000Y6, HD68000Y8, HD68000Y10, HD68000Y12) MPU (Micro Processing Unit)

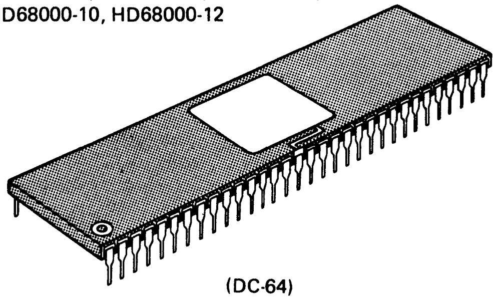
Advances in semiconductor technology have provided the capability to place on a single silicon chip a microprocessor at least an order of magnitude higher in performance and circuit complexity than has been previously available. The HD68000 is one of such VLSI microprocessors. It combines rate-of-the-art technology and advanced circuit design techniques with computer sciences to achieve an architecturally advanced 16-bit microprocessor.

The resources available to the HD68000 user consist of the following.

As shown in the programming model, the HD68000 offers seventeen 32-bit registers in addition to the 32-bit program counter and a 16-bit status register. The first eight registers (D0-D7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) data operations. The second set of seven registers (A0-A6) and the system stack pointer may be used as software stack pointers and base address registers. In addition, these registers may be used for word and long word address operations. All 17 registers may be used as index registers.

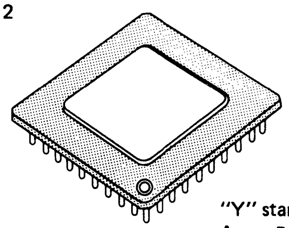
– The specification for HD68000-10/-12 and HD68000 Y4/Y6/Y8/Y10/Y12 are preliminary. –

HD68000-4, HD68000-6, HD68000-8, HD68000-10, HD68000-12



(DC-64)

HD68000Y4, HD68000Y6, HD68000Y8, HD68000Y10, HD68000Y12



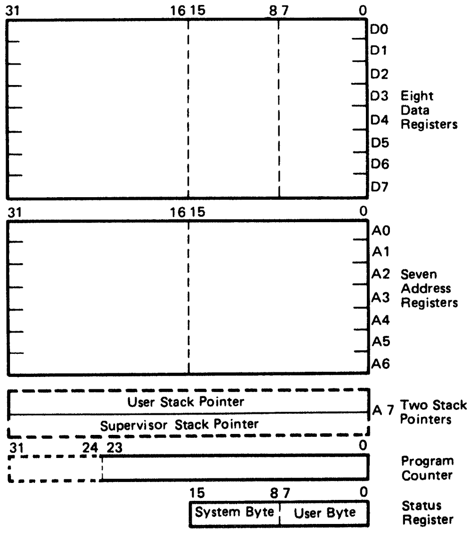
"Y" stands for Pin Grid Array Package.

(PGA-68)

■ FEATURES

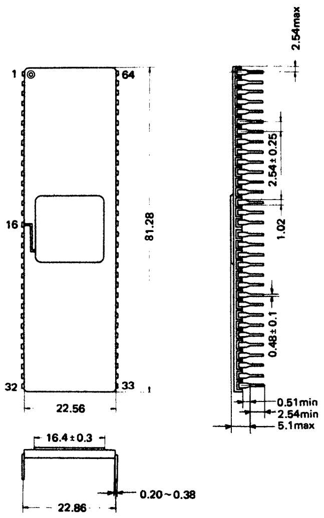
- 32-Bit Data and Address Registers
- 16 Megabyte Direct Addressing Range
- 56 Powerful Instruction Types
- Operations of Five Main Data Types
- Memory Mapped I/O
- 14 Addressing Modes
- Compatible with MC68000L4, MC68000L6, MC68000L8, MC68000L10 and MC68000L12

■ PROGRAMMING MODEL

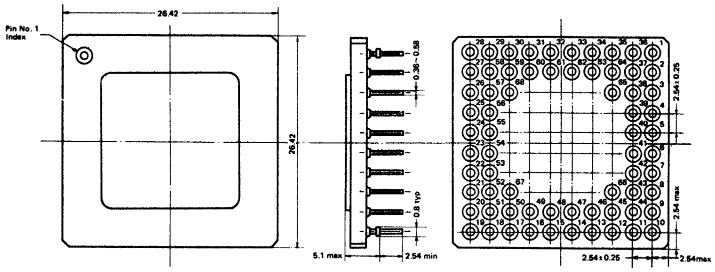


■ PACKAGE DIMENSIONS (Unit: mm)

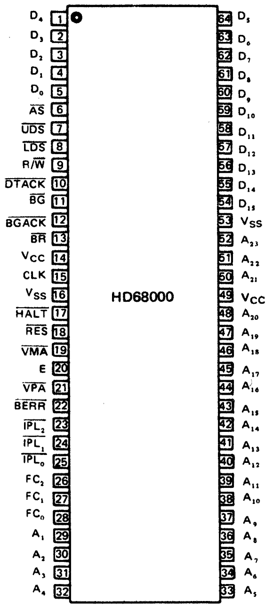
● DC-64 (Side-brazed Ceramic DIP)



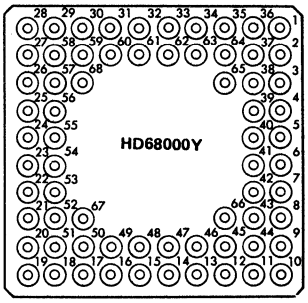
● PGA-68 (Pin Grid Array)



■ PIN ARRANGEMENT



(Top View)



(Bottom View)

Pin No.	Function	Pin No.	Function	Pin No.	Function	Pin No.	Function
1	N/C	18	A_9	35	D_1	52	A_{12}
2	$\overline{DTACK}$	19	N/C	36	$\overline{AS}$	53	A_{15}
3	$\overline{BGACK}$	20	A_{14}	37	$\overline{LDS}$	54	A_{16}
4	$\overline{BR}$	21	A_{16}	38	$\overline{BG}$	55	V_{CC}
5	CLK	22	A_{17}	39	V_{CC}	56	V_{SS}
6	$\overline{HALT}$	23	A_{19}	40	V_{SS}	57	A_{23}
7	$\overline{VMA}$	24	A_{20}	41	$\overline{RES}$	58	D_{14}
8	E	25	A_{21}	42	$\overline{VPA}$	59	D_{11}
9	$\overline{BERR}$	26	A_{22}	43	$\overline{IPL_2}$	60	D_9
10	N/C	27	D_{15}	44	$\overline{IPL_0}$	61	D_6
11	FC_2	28	D_{12}	45	FC_1	62	D_3
12	FC_0	29	D_{10}	46	N/C	63	D_0
13	A_1	30	D_8	47	A_2	64	$\overline{UDS}$
14	A_3	31	D_7	48	A_5	65	R/$\overline{W}$
15	A_4	32	D_5	49	A_8	66	$\overline{IPL_1}$
16	A_6	33	D_4	50	A_{10}	67	A_{13}
17	A_7	34	D_2	51	A_{11}	68	D_{13}

■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Value	Unit
Supply Voltage	V_{CC}*	-0.3 ~ +7.0	V
Input Voltage	V_{in}*	-0.3 ~ +7.0	V
Operating Temperature Range	T_{opr}	0 ~ +70	°C
Storage Temperature	T_{stg}	-55 ~ +150	°C

* With respect to V_{SS} (SYSTEM GND)

(NOTE) Permanent LSI damage may occur if maximum ratings are exceeded. Normal operation should be under recommended operating conditions. If these conditions are exceeded, it could affect reliability of LSI.

■ RECOMMENDED OPERATING CONDITIONS

Item	Symbol	min	typ	max	Unit
Supply Voltage	V_{CC}*	4.75	5.0	5.25	V
Input Voltage	V_{IH}*	2.0	–	V_{CC}	V
	V_{IL}*	-0.3	–	0.8	V
Operating Temperature	T_{opr}	0	25	70	°C

* With respect to V_{SS} (SYSTEM GND)

■ ELECTRICAL CHARACTERISTICS

● DC CHARACTERISTICS (V_{CC} = 5V ±5%, V_{SS} = 0V, Ta = 0 ~ +70°C, Fig. 1, 2, 3, unless otherwise noted.)

Item		Symbol	Test Condition	min	typ	max	Unit
Input "High" Voltage		V_{IH}		2.0	–	V_{CC}	V
Input "Low" Voltage		V_{IL}		V_{SS}-0.3	–	0.8	V
Input Leakage Current	$\overline{\text{BERR}}$, $\overline{\text{BGACK}}$, $\overline{\text{BR}}$, $\overline{\text{DTACK}}$, $\overline{\text{IPL}_0}$ ~ $\overline{\text{IPL}_2}$, $\overline{\text{VPA}}$, CLK	I_{in}	@5.25V	–	–	2.5	μA
	$\overline{\text{HALT}}$, $\overline{\text{RES}}$			–	–	20	
Three-State (Off State) Input Current	$\overline{\text{AS}}$, A_1 ~ A_{23}, D_0 ~ D_{15}, FC_0 ~ FC_2, $\overline{\text{LDS}}$, R/$\overline{\text{W}}$, $\overline{\text{UDS}}$, $\overline{\text{VMA}}$	I_{TSI}	@2.4V/0.4V	–	–	20	μA
Output "High" Voltage	$\overline{\text{AS}}$, A_1 ~ A_{23}, $\overline{\text{BG}}$, D_0 ~ D_{15}, FC_0 ~ FC_2, $\overline{\text{LDS}}$, R/$\overline{\text{W}}$, $\overline{\text{UDS}}$, $\overline{\text{VMA}}$	V_{OH}	I_{OH} = -400μA	2.4	–	–	V
	E*			V_{CC}-0.75	–	–	
Output "Low" Voltage	$\overline{\text{HALT}}$	V_{OL}	I_{OL} = 1.6 mA	–	–	0.5	V
	A_1 ~ A_{23}, $\overline{\text{BG}}$, FC_0 ~ FC_2		I_{OL} = 3.2 mA	–	–	0.5	
	$\overline{\text{RES}}$		I_{OL} = 5.3 mA	–	–	0.5	
	$\overline{\text{AS}}$, D_0 ~ D_{15}, $\overline{\text{LDS}}$, R/$\overline{\text{W}}$, E, $\overline{\text{UDS}}$, $\overline{\text{VMA}}$		I_{OL} = 5.3 mA	–	–	0.5	
Power Dissipation		P_D	f = 8 MHz	–	–	1.5	W
Capacitance (Package Type Dependent)		C_{in}	V_{in} = 0V, Ta = 25°C, f = 1 MHz	–	10.0	20.0	pF

* With external pull up register of 470 Ω

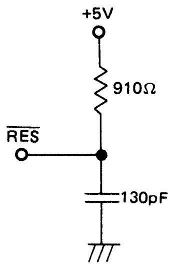


Figure 1 $\overline{\text{RES}}$ Test Load

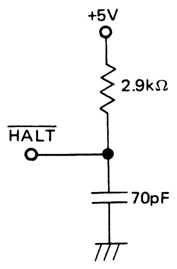
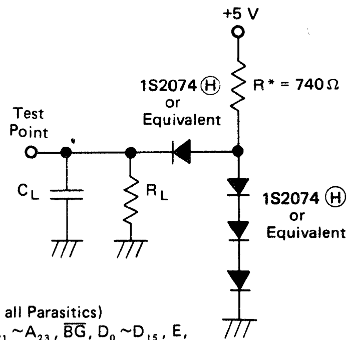


Figure 2 $\overline{\text{HALT}}$ Test Load



C_L = 130 pF (Includes all Parasitics)
R_L = 6.0 kΩ for $\overline{\text{AS}}$, A_1 ~ A_{23}, $\overline{\text{BG}}$, D_0 ~ D_{15}, E, FC_0 ~ FC_2, $\overline{\text{LDS}}$, R/$\overline{\text{W}}$, $\overline{\text{UDS}}$, $\overline{\text{VMA}}$
*R = 1.22 kΩ for A_1 ~ A_{23}, $\overline{\text{BG}}$, E, FC_0 ~ FC_2

Figure 3 Test Loads

• AC CHARACTERISTICS (V_{CC} = 5.0V ±5%, V_{SS} = 0V, Ta = 0 ~ +70°C, unless otherwise noted.)

Number	Item	Symbol	Test Condition	4MHz Version HD68000-4 HD68000Y4*		6MHz Version HD68000-6 HD68000Y6*		8MHz Version HD68000-8 HD68000Y8*		10MHz Version HD68000-10* HD68000Y10*		12.5MHz Version HD68000-12* HD68000Y12*		Unit
				min	max	min	max	min	max	min	max	min	max	
	Frequency of Operation	f		2.0	4.0	2.0	6.0	2.0	8.0	2.0	10.0	2.0	12.5	MHz
①	Clock Period	t_{cyc}		250	500	167	500	125	500	100	500	80	500	ns
②	Clock Width "Low"	t_{CL}		115	250	75	250	55	250	45	250	35	250	ns
③	Clock Width "High"	t_{CH}		115	250	75	250	55	250	45	250	35	250	ns
④	Clock Fall Time	t_{Cf}		—	10	—	10	—	10	—	10	—	5	ns
⑤	Clock Rise Time	t_{Cr}		—	10	—	10	—	10	—	10	—	5	ns
⑥	Clock "Low" to Address	t_{CLAV}		—	90	—	80	—	70	—	55	—	55	ns
6A	Clock "High" to FC Valid	t_{CHFCV}		—	90	—	80	—	80	—	60	—	55	ns
⑦	Clock "High" to Address/Data High Impedance (Maximum)	t_{CHAZx}		—	120	—	100	—	80	—	70	—	60	ns
⑧	Clock "High" to Address/FC Invalid (Minimum)	t_{CHAZn}		0	—	0	—	0	—	0	—	0	—	ns
⑨[1]	Clock "High" to $\overline{AS}$, $\overline{DS}$ "Low" (Maximum)	t_{CHSLx}		—	80	—	70	—	60	—	55	—	55	ns
⑩	Clock "High" to $\overline{AS}$, $\overline{DS}$ "Low" (Minimum)	t_{CHSLn}		0	—	0	—	0	—	0	—	0	—	ns
⑪[2]	Address to $\overline{AS}$, $\overline{DS}$ (Read) "Low" /$\overline{AS}$ Write	t_{AVSL}		55	—	35	—	30	—	20	—	0	—	ns
11A[2]	FC Valid to $\overline{AS}$, $\overline{DS}$ (Read) "Low" /$\overline{AS}$ Write	t_{FCVSL}		80	—	70	—	60	—	50	—	40	—	ns
⑫[1]	Clock "Low" to $\overline{AS}$, $\overline{DS}$ "High"	t_{CLSH}		—	90	—	80	—	70	—	55	—	50	ns
⑬[2]	$\overline{AS}$, $\overline{DS}$ "High" to Address/FC Invalid	t_{SHAZ}		60	—	40	—	30	—	20	—	10	—	ns
⑭[2,5]	$\overline{AS}$, $\overline{DS}$ Width "Low" (Read)/$\overline{AS}$ Write	t_{SL}		535	—	337	—	240	—	195	—	160	—	ns
14A[2]	$\overline{DS}$ Width "Low" (Write)	—		285	—	170	—	115	—	95	—	80	—	ns
⑮[2]	$\overline{AS}$, $\overline{DS}$ Width "High"	t_{SH}		285	—	180	—	150	—	105	—	65	—	ns
⑯	Clock "High" to $\overline{AS}$, $\overline{DS}$ High Impedance	t_{CHSZ}		—	120	—	100	—	80	—	70	—	60	ns
⑰[2]	$\overline{AS}$, $\overline{DS}$ "High" to R/$\overline{W}$ "High"	t_{SHRH}		60	—	50	—	40	—	20	—	10	—	ns
⑱[1]	Clock "High" to R/$\overline{W}$ "High" (Maximum)	t_{CHRHx}		—	90	—	80	—	70	—	60	—	60	ns
⑲	Clock "High" to R/$\overline{W}$ "High" (Minimum)	t_{CHRHn}		0	—	0	—	0	—	0	—	0	—	ns
⑳[1]	Clock "High" to R/$\overline{W}$ "Low"	t_{CHRL}		—	90	—	80	—	70	—	60	—	60	ns
㉑[2]	Address Valid to R/$\overline{W}$ "Low"	t_{AVRL}	Fig. 4 ~Fig. 7	45	—	25	—	20	—	0	—	0	—	ns
21A[2]	FC Valid to R/$\overline{W}$ "Low"	t_{FCVRL}		80	—	70	—	60	—	50	—	30	—	ns
㉒[2]	R/$\overline{W}$ "Low" to $\overline{DS}$ "Low" (Write)	t_{RLSL}		200	—	140	—	80	—	50	—	30	—	ns
㉓	Clock "Low" to Data Out Valid	t_{CLDO}		—	90	—	80	—	70	—	55	—	55	ns
㉕[2]	$\overline{DS}$ "High" to Data Out Invalid	t_{SHDO}		60	—	40	—	30	—	20	—	15	—	ns
㉖[2]	Data Out Valid to $\overline{DS}$ "Low" (Write)	t_{DOSL}		55	—	35	—	30	—	20	—	15	—	ns
㉗[6]	Data In to Clock "Low" (Setup Time)	t_{DICL}		30	—	25	—	15	—	15	—	15	—	ns
㉘[2]	$\overline{AS}$, $\overline{DS}$ "High" to $\overline{DTACK}$ "High"	t_{SHDAH}		0	240	0	160	0	120	0	90	0	70	ns
㉙	$\overline{DS}$ "High" to Data Invalid (Hold Time)	t_{SHDI}		0	—	0	—	0	—	0	—	0	—	ns
㉚	$\overline{AS}$, $\overline{DS}$ "High" to $\overline{BERR}$ "High"	t_{SHBEH}		0	—	0	—	0	—	0	—	0	—	ns
㉛[2,6]	$\overline{DTACK}$ "Low" to Data In (Setup Time)	t_{DALDI}		—	180	—	120	—	90	—	65	—	50	ns
㉜	$\overline{HALT}$ and $\overline{RES}$ Input Transition Time	t_{RHrf}		0	200	0	200	0	200	0	200	0	200	ns
㉝	Clock "High" to $\overline{BG}$ "Low"	t_{CHGL}		—	90	—	80	—	70	—	60	—	50	ns
㉞	Clock "High" to $\overline{BG}$ "High"	t_{CHGH}		—	90	—	80	—	70	—	60	—	50	ns
㉟	$\overline{BR}$ "Low" to $\overline{BG}$ "Low"	t_{BRLGL}		1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk.Per.
㊱	$\overline{BR}$ "High" to $\overline{BG}$ "High"	t_{BRHGH}		1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk.Per.
㊲	$\overline{BGACK}$ "Low" to $\overline{BG}$ "High"	t_{GALGH}		1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk.Per.
㊳	$\overline{BG}$ "Low" to Bus High Impedance (With $\overline{AS}$ "High")	t_{GLZ}		—	120	—	100	—	80	—	70	—	60	ns
㊴	$\overline{BG}$ Width "High"	t_{GH}		1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk.Per.
㊻	$\overline{BGACK}$ Width "Low"	t_{BGL}		1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk.Per.
㊼[6]	Asynchronous Input Setup Time	t_{ASI}		30	—	25	—	20	—	20	—	20	—	ns
㊽	$\overline{BERR}$ "Low" to $\overline{DTACK}$ "Low" (Note 3)	t_{BELDAL}		50	—	50	—	50	—	50	—	50	—	ns
53	Data Hold from Clock "High"	t_{CHDO}		0	—	0	—	0	—	0	—	0	—	ns
55	R/$\overline{W}$ to Data Bus Impedance Change	t_{RLDO}		55	—	35	—	30	—	20	—	10	—	ns
56	$\overline{HALT}$ /$\overline{RES}$ Pulse Width (Note 4)	t_{HRPW}		10	—	10	—	10	—	10	—	10	—	Clk.Per.

* Preliminary

(to be continued)

Number	Item	Symbol	Test Condition	4MHz Version HD68000-4 HD68000Y4*		6MHz Version HD68000-6 HD68000Y6*		8MHz Version HD68000-8 HD68000Y8*		10MHz Version HD68000-10* HD68000Y10*		12.5MHz Version HD68000-12* HD68000Y12*		Unit
				min	max	min	max	min	max	min	max	min	max	
24	Clock "High" to R/$\overline{W}$, $\overline{VMA}$ High Impedance	t_{CHRZ}	Fig. 45, Fig. 46	—	120	—	100	—	80	—	70	—	60	ns
40	Clock "Low" to $\overline{VMA}$ "Low"	t_{CLVML}		—	90	—	80	—	70	—	70	—	70	ns
41	Clock "Low" to E Transition	t_{CLE}		—	100	—	85	—	70	—	55	—	45	ns
42	E Output Rise and Fall Time	t_{Erf}		—	25	—	25	—	25	—	25	—	25	ns
43	$\overline{VMA}$ "Low" to E "High"	t_{VMLEH}		325	—	240	—	200	—	150	—	90	—	ns
44	$\overline{AS}$, $\overline{DS}$ "High" to $\overline{VPA}$ "High"	t_{SHVPH}		0	240	0	160	0	120	0	90	0	70	ns
45	E "Low" to Address/$\overline{VMA}$/FC Invalid	t_{ELAI}		55	—	35	—	30	—	10	—	10	—	ns
49	E "Low" to $\overline{AS}$, $\overline{DS}$ Invalid	t_{ELSI}		−80	—	−80	—	−80	—	−80	—	−80	—	ns
50	E Width "High"	t_{EH}		900	—	600	—	450	—	350	—	280	—	ns
51	E Width "Low"	t_{EL}		1400	—	900	—	700	—	550	—	440	—	ns
52	E Extended Rise Time	t_{CIEHX}		80	—	80	—	80	—	80	—	80	—	ns
54	Data Hold from E "Low" (Write)	t_{ELDOZ}		60	—	40	—	30	—	20	—	15	—	ns

* Preliminary

(NOTES) 1. For a loading capacitance of less than or equal to 50 picofarads, subtract 5 nanoseconds from the values given in these columns.
2. Actual value depends on clock period.
3. If #47 is satisfied for both $\overline{DTACK}$ and $\overline{BERR}$, #48 may be 0 ns.
4. After V_{CC} has been applied for 100 ms.
5. For the mask version 68000 #14 and #14A are one clock period less than the given number.
6. If the asynchronous setup time (#47) requirements are satisfied, the $\overline{DTACK}$ low-to-data setup time (#31) requirement can be ignored. The data must only satisfy the data-in to clock-low setup time (#27) for the following cycle.

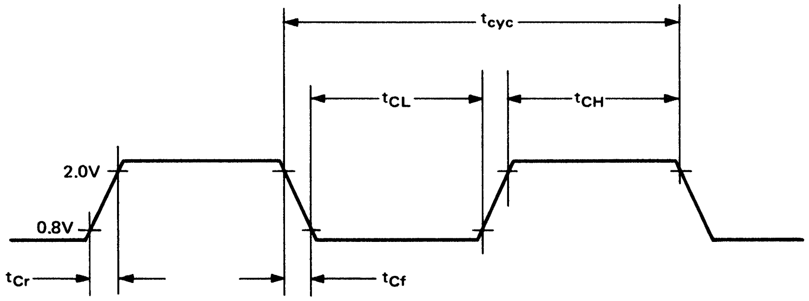
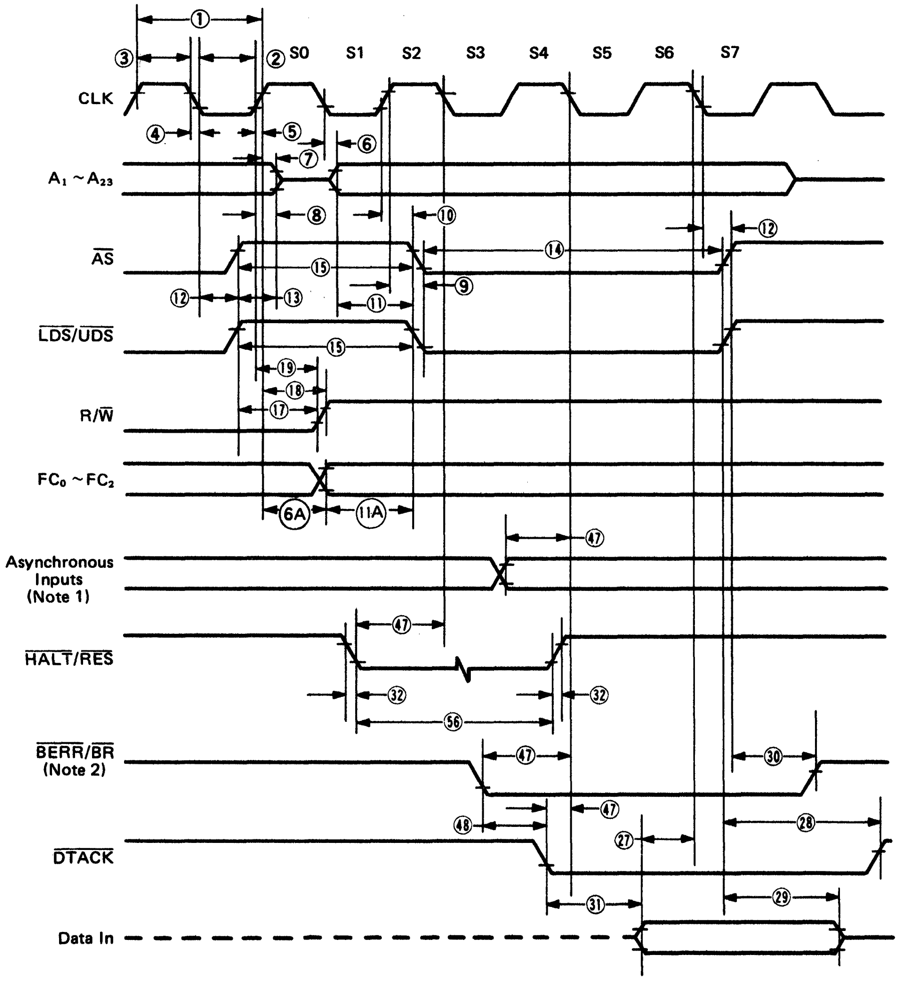


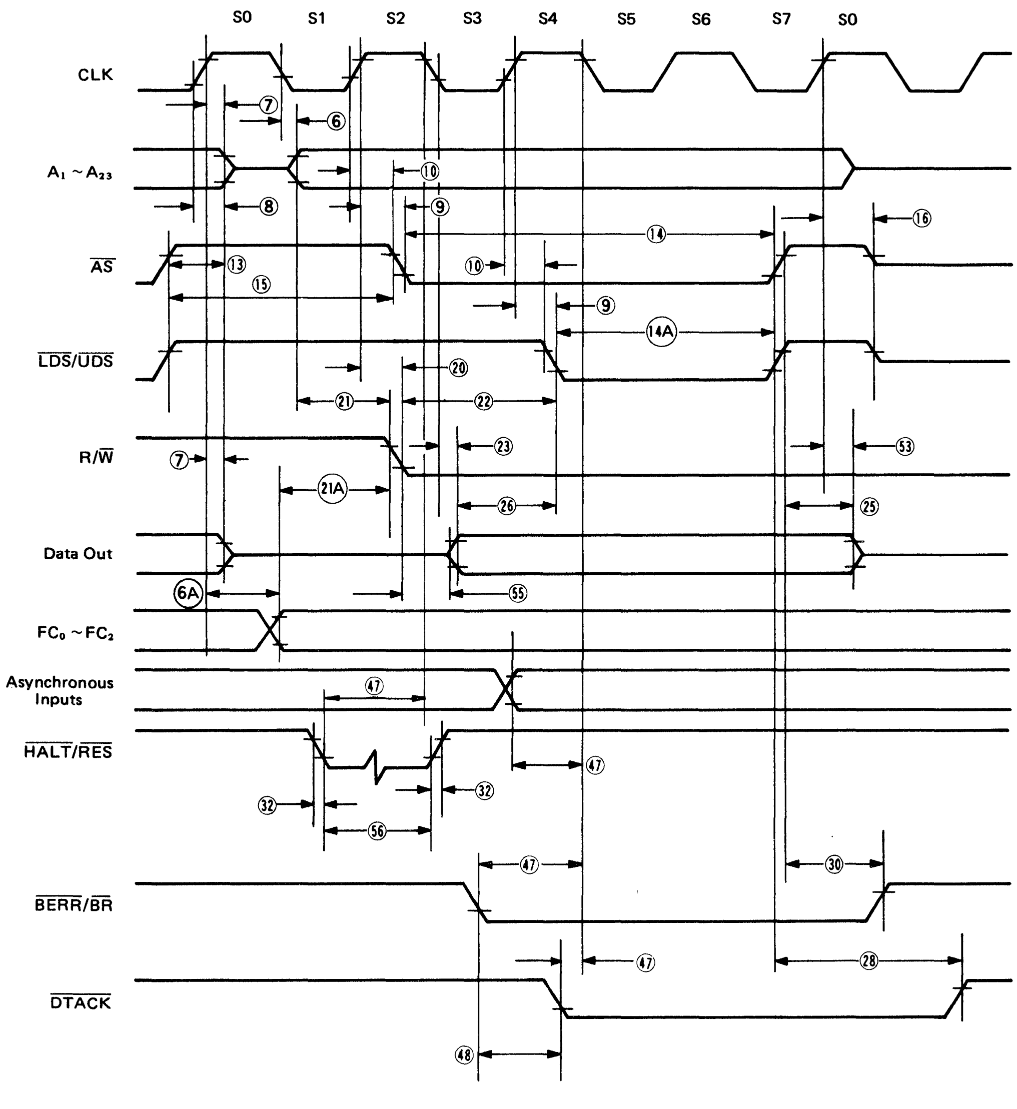
Figure 4 Input Clock Waveform



CLK
$A_1 \sim A_{23}$
$\overline{AS}$
$\overline{LDS}/\overline{UDS}$
$R/\overline{W}$
$FC_0 \sim FC_2$
Asynchronous Inputs (Note 1)
$\overline{HALT}/\overline{RES}$
$\overline{BERR}/\overline{BR}$ (Note 2)
$\overline{DTACK}$
Data In
S0
S1
S2
S3
S4
S5
S6
S7

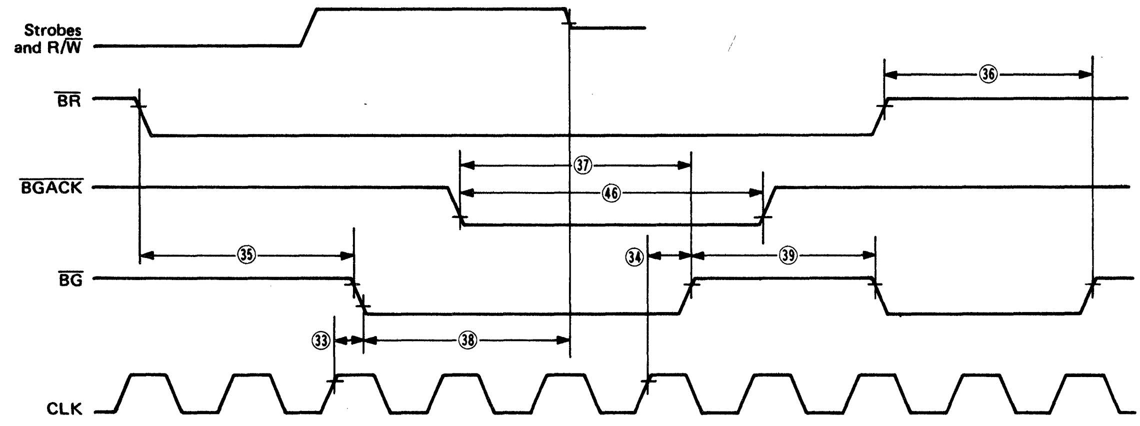
(NOTES) 1. Setup time for the asynchronous inputs $\overline{BGACK}$, $\overline{IPL_0}$ ~ $\overline{IPL_2}$ and $\overline{VPA}$ guarantees their recognition at the next falling edge of the clock.
2. $\overline{BR}$ need fall at this time only in order to insure being recognized at the end of this bus cycle.
3. Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted.

Figure 5 Read Cycle Timing



(NOTE) Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted.

Figure 6 Write Cycle Timing



(NOTES) 1. Setup time for the asynchronous inputs $\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, $\overline{IPL_0}$ ~ $\overline{IPL_2}$, and $\overline{VPA}$ guarantees their recognition at the next falling edge of the clock.
2. Waveform measurements for all inputs and outputs are specified at: logic high = 2.0 volts, logic low = 0.8 volts.
3. These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input an output signals. Refer to other functional descriptions and their related diagrams for device operation.

Figure 7 AC Electrical Waveforms — Bus Arbitration

■ SIGNAL DESCRIPTION

The following paragraphs contain a brief description of the input and output signals. A discussion of bus operation during the various machine cycles and operations is also given.

• SIGNAL DESCRIPTION

The input and output signals can be functionally organized into the groups shown in Figure 8. The following paragraphs provide a brief description of the signals and also a reference (if applicable) to other paragraphs that contain more detail about the function being performed.

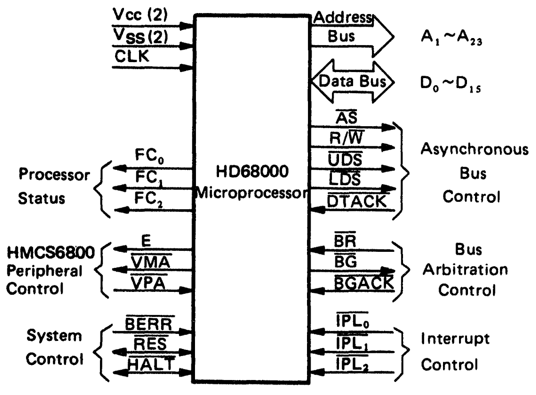


Figure 8 Input and Output Signals

ADDRESS BUS (A_1 through A_{23})

This 23-bit, unidirectional, three-state bus is capable of addressing 8 megawords of data. It provides the address for bus operation during all cycles except interrupt cycles. During interrupt cycles, address lines A_1, A_2, and A_3 Provide information about what level interrupt is being serviced while address lines A_4 through A_{23} are all set to a logic high.

DATA BUS (D_0 through D_{15})

This 16-bit, bidirectional, three-state bus is the general purpose data path. It can transfer and accept data in either word or byte length. During an interrupt acknowledge cycle, an external device supplies the vector number on data lines $D_0 \sim D_7$.

ASYNCHRONOUS BUS CONTROL

Asynchronous data transfer are handled using the following control signals: address strobe, read/write, upper and lower data strobes, and data transfer acknowledge. These signals are explained in the following paragraphs.

Address Strobe ($\overline{AS}$)

This signal indicates that there is a valid address on the address bus.

Read/Write (R/$\overline{W}$)

This signal defines the data bus transfer as a read or write cycle. The R/$\overline{W}$ signal also works in conjunction with the upper and lower data strobes as explained in the following paragraph.

Upper and Lower Data Strobes ($\overline{\text{UDS}}$, $\overline{\text{LDS}}$)

These signals control the data on the data bus, as shown in Table 1. When the R/$\overline{\text{W}}$ line is high, the processor will read from the data bus as indicated. When the R/$\overline{\text{W}}$ line is low, the processor will write to the data bus as shown.

Table 1 Data Strobe Control of Data Bus

$\overline{\text{UDS}}$	$\overline{\text{LDS}}$	R/$\overline{\text{W}}$	$D_8 \sim D_{15}$	$D_0 \sim D_7$
High	High	–	No valid data	No valid data
Low	Low	High	Valid data bits 8 ~ 15	Valid data bits 0 ~ 7
High	L ow	High	No valid data	Valid data bits 0 ~ 7
Low	High	High	Valid data bits 8 ~ 15	No valid data
Low	Low	Low	Valid data bits 8 ~ 15	Valid data bits 0 ~ 7
High	Low	Low	Valid data bits 0 ~ 7*	Valid data bits 0 ~ 7
Low	High	Low	Valid data bits 8 ~ 15	Valid data bits 8 ~ 15*

* These conditions are a result of current implementation and may not appear on future devices.

Data Transfer Acknowledge ($\overline{\text{DTACK}}$)

This input indicates that the data transfer is completed. When the processor recognizes $\overline{\text{DTACK}}$ during a read cycle, data is latched and the bus cycle terminated. When $\overline{\text{DTACK}}$ is recognized during a write cycle, the bus cycle is terminated.

An active transition of data transfer acknowledge, $\overline{\text{DTACK}}$, indicates the termination of a data transfer on the bus.

If the system must run at a maximum rate determined by RAM access times, the relationship between the times at which $\overline{\text{DTACK}}$ and DATA are sampled are important.

All control and data lines are sampled during the HD68000's clock high time. The clock is internally buffered, which results in some slight differences in the sampling and recognition of various signals. HD68000 allow $\overline{\text{BERR}}$ or $\overline{\text{DTACK}}$ to be recognized in S4, S6, etc., which terminates the cycle*. The $\overline{\text{DTACK}}$ signal, like other control signals, is internally synchronized to allow for valid operation in an asynchronous system. If the required setup time (#47) is met during S4, $\overline{\text{DTACK}}$ will be recognized during S5 and S6, and data will be captured during S6. The data must meet the required setup time (#27).

If an asynchronous control signal does not meet the required setup time, it is possible that it may not be recognized during that cycle. Because of this, asynchronous systems must not allow $\overline{\text{DTACK}}$ to precede data by more than parameter #31.

Asserting $\overline{\text{DTACK}}$ (or $\overline{\text{BERR}}$) on the rising edge of a clock (such as S4) after the assertion of address strobe will allow a HD68000 system to run at its maximum bus rate. If setup times #27 and #47 are guaranteed, #31 may be ingnored.

* The mask version 68000 allowed $\overline{\text{DTACK}}$ to be recognized as early as S2 (bus state 2).

BUS ARBITRATION CONTROL

These three signals form a bus arbitration circuit to determine which device will be the bus master device.

Bus Request ($\overline{\text{BR}}$)

This input is wire ORed with all other devices that could be bus masters. This input indicates to the processor that some other device desires to become the bus master.

Bus Grant ($\overline{\text{BG}}$)

This output indicates to all other potential bus master devices that the processor will release bus control at the end of the current bus cycle.

Bus Grant Acknowledge ($\overline{\text{BGACK}}$)

This input indicates that some other device has become the bus master. This signal cannot be asserted until the following four conditions are met:

(1) A Bus Grant has been received
(2) Address Strobe is inactive which indicates that the microprocessor is not using the bus
(3) Data Transfer Acknowledge is inactive which indicates that neither memory nor peripherals are using the bus
(4) Bus Grant Acknowledge is inactive which indicates that no other device is still claiming bus mastership.

INTERRUPT CONTROL ($\overline{\text{IPL}_0}$, $\overline{\text{IPL}_1}$, $\overline{\text{IPL}_2}$)

These input pins indicate the encoded priority level of the device requesting an interrupt. Level seven is the highest priority while level zero indicates that no interrupts are requested. The least significant bit is given in $\overline{\text{IPL}_0}$ and the most significant bit is contained in $\overline{\text{IPL}_2}$.

SYSTEM CONTROL

The system control inputs are used to either reset or halt the processor and to indicate to the processor that bus errors have occurred. The three system control inputs are explained in the following paragraphs.

Bus Error ($\overline{\text{BERR}}$)

This input informs the processor that there is a problem with the cycle currently being executed. Problems may be a result of:

(1) Nonresponding devices
(2) Interrupt vector number acquisition failure
(3) Illegal access request as determined by a memory management unit
(4) Other application dependent errors.

The bus error signal interacts with the halt signal to determine if exception processing should be performed or the current bus cycle should be retried.

Refer to **BUS ERROR AND HALT OPERATION** paragraph for additional information about the interaction of the bus error and halt signals.

Reset ($\overline{\text{RES}}$)

This bidirectional signal line acts to reset (initiate a system initialization sequence) the processor in response to an external reset signal. An internally generated reset (result of a RESET instruction) causes all external devices to be reset and the internal state of the processor is not affected. A total system reset (processor and external devices) is the result of external HALT and RESET signals applied at the same time. Refer to **RESET OPERATION** paragraph for additional information about reset operation.

Halt ($\overline{\text{HALT}}$)

When this bidirectional line is driven by an external device, it will cause the processor to stop at the completion of the current bus cycle. When the processor has been halted using this input, all control signals are inactive and all three-state lines are put in their high-impedance state. Refer to **BUS ERROR AND HALT OPERATION** paragraph for additional information about the interaction between the halt and bus error signals.

When the processor has stopped executing instructions, such as in a double bus fault condition, the halt line is driven by the processor to indicate to external devices that the processor has stopped.

HMCS6800 PERIPHERAL CONTROL

These control signals are used to allow the interfacing of synchronous HMCS6800 peripheral devices with the asynchronous HD68000. These signals are explained in the following paragraphs.

Enable (E)

This signal is the standard enable signal common to all HMCS6800 type peripheral devices. The period for this output is ten HD68000 clock periods (six clocks low; four clocks high).

Valid Peripheral Address ($\overline{\text{VPA}}$)

This input indicates that the device or region addressed is a HMCS6800 family device and that data transfer should be synchronized with the enable (E) signal. This input also indicates that the processor should use automatic vectoring for an interrupt. Refer to **INTERFACE WITH HMCS6800 PERIPHERALS.**

Valid Memory Address ($\overline{\text{VMA}}$)

This output is used to indicate to HMCS6800 peripheral devices that there is a valid address on the address bus and the processor is synchronized to enable. This signal only responds to a valid peripheral address ($\overline{\text{VPA}}$) input which indicates that the peripheral is a HMCS6800 family device.

PROCESSOR STATUS (FC_0, FC_1, FC_2)

These function code outputs indicate the state (user or supervisor) and the cycle type currently being executed, as shown in Table 2. The information indicated by the function code outputs is valid whenever address strobe ($\overline{\text{AS}}$) is active.

Table 2 Function Code Outputs

FC_2	FC_1	FC_0	Cycle Type
Low	Low	Low	(Undefined, Reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	(Undefined, Reserved)
High	Low	Low	(Undefined, Reserved)
High	Low	High	Superviser Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

CLOCK (CLK)

The clock input is a TTL-compatible signal that is internally buffered for development of the internal clocks needed by the processor. The clock input shall be a constant frequency.

SIGNAL SUMMARY

Table 3 is a summary of all the signals discussed in the previous paragraphs.

Table 3 Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	Three State
Address Bus	$A_1 \sim A_{23}$	output	high	yes
Data Bus	$D_0 \sim D_{15}$	input/output	high	yes
Address Strobe	$\overline{\text{AS}}$	output	low	yes
Read/Write	R/$\overline{\text{W}}$	output	read-high write-low	yes
Upper and Lower Data Strobes	$\overline{\text{UDS}}$, $\overline{\text{LDS}}$	output	low	yes
Data Transfer Acknowledge	$\overline{\text{DTACK}}$	input	low	no
Bus Request	$\overline{\text{BR}}$	input	low	no
Bus Grant	$\overline{\text{BG}}$	output	low	no
Bus Grant Acknowledge	$\overline{\text{BGACK}}$	input	low	no
Interrupt Priority Level	$\overline{\text{IPL}_0}$, $\overline{\text{IPL}_1}$, $\overline{\text{IPL}_2}$	input	low	no
Bus Error	$\overline{\text{BERR}}$	input	low	no
Reset	$\overline{\text{RES}}$	input/output	low	no*
Halt	$\overline{\text{HALT}}$	input/output	low	no*
Enable	E	output	high	no
Valid Memory Address	$\overline{\text{VMA}}$	output	low	yes
Valid Peripheral Address	$\overline{\text{VPA}}$	input	low	no
Function Code Output	FC_0, FC_1, FC_2	output	high	yes
Clock	CLK	input	high	no
Power Input	V_{CC}	input	–	–
Ground	V_{SS}	input	–	–

* Open drain

■ REGISTER DESCRIPTION AND DATA ORGANIZATION

The following paragraphs describe the registers and data organization of the HD68000.

• OPERAND SIZE

Operand sizes are defined as follows: a byte equals 8 bits, a word equals 16 bits, and a long word equals 32 bits. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. All explicit instructions support byte, word or long word operands. Implicit instructions support some subset of all three sizes.

• DATA ORGANIZATION IN REGISTERS

The eight data registers support data operands of 1, 8, 16, or 32 bits. The seven address registers together with the active stack pointer support address operands of 32 bits.

DATA REGISTERS

Each data register is 32 bits wide. Byte operands occupy the low order 8 bits, word operands the low order 16 bits, and long word operands the entire 32 bits. The least significant bit is addressed as bit zero; the most significant bit is addressed as bit 31.

When a data register is used as either a source or destination operand, only the appropriate low-order portion is changed; the remaining high-order portion is neither used nor changed.

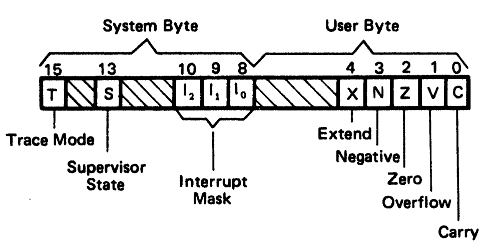
ADDRESS REGISTERS

Each address register and the stack pointer is 32 bits wide and holds a full 32 bit address. Address registers do not support byte sized operands. Therefore, when an address register is used as a source operand, either the low order word or the entire long word operand is used depending upon the operation size. When an address register is used as the destination operand, the entire register is affected regardless of the operation size. If the operation size is word, any other operands are sign extended to 32 bits before the operation is performed.

• STATUS REGISTER

The status register contains the interrupt mask (eitht levels available) as well as the condition codes; extend (X), negative (N), zero (Z), overflow (V), and carry (C). Additional status bits indicate that the processor is in a trace (T) mode and/or in a supervisor (S) state.

Status Register



Unused, read as zero.

• DATA ORGANIZATION IN MEMORY

Bytes are individually addressable with the high order byte having an even address the same as the word, as shown in Figure 9. The low order byte has an odd address that is one count higher than the word address. Instructions and multibyte data are accessed only on word (even byte) boundaries. If a long word datum is located at address n (n even), then the second word of that datum is located at address n + 2.

The data types supported by the HD68000 are: bit data, integer data of 8, 16, or 32 bits, 32-bit addresses and binary coded decimal data. Each of these data types is put in memory, as shown in Figure 10.

■ BUS OPERATION

The following paragraphs explain control signal and bus operation during data transfer operations, bus arbitration, bus error and halt conditions, and reset operation.

• DATA TRANSFER OPERATIONS

Transfer of data between devices involve the following leads:

(1) Address Bus A_1 through A_{23}
(2) Data Bus D_0 through D_{15}
(3) Control Signals

The address and data buses are separate parallel buses used to transfer data using an asynchronous bus structure. In all cycles, the bus master assumes responsibility for deskewing all signals it issues at both the start and end of a cycle. In addition, the bus master is responsible for deskewing the acknowledge and data signals from the slave device.

The following paragraphs explain the read, write, and read-modify-write cycles. The indivisible read-modify-write cycle is the method used by the HD68000 for interlocked multiprocessor communications.

(NOTE) The terms **assertion** and **negation** will be used extensively. This is done to avoid confusion when dealing with a mixture of "active-low" and "active-high" signals. The term assert or assertion is used to indicate that a signal is active or true independent of whether that voltage is low or high. The term negate or negation is used to indicate that a signal is inactive or false.

Read Cycle

During a read cycle, the processor receives data from memory or a peripheral device. The processor reads bytes of data in all cases. If the instruction specifies a word (or double word) operation, the processor reads both bytes. When the instruction specifies byte operation, the processor uses an internal A_0 bit to determine which byte to read and then issues the data strobe required for that byte. For bytes operations, when the A_0 bit equals zero, the upper data strobe is issued. When the A_0 bit equals one, the lower data strobe is issued. When the data is received, the processor correctly positions it internally.

A word read cycle flow chart is given in Figure 11. A byte read cycle flow chart is given in Figure 12. Read cycle timing is given in Figure 13. Figure 14 details word and byte read cycle operations. Refer to these illustrations during the following detailed.

At state zero (S0) in the read cycle, the address bus (A_1 through A_{23}) is in the high impedance state. A function code is asserted on the function code output line (FC_0 through FC_2). The read/write (R/W) signal is switched high to indicate a read cycle. One half clock cycle later, at state 1, the address bus is released from the high impedance state. The function code outputs indicate which address space that this cycle will operate on.

In state 2, the address strobe ($\overline{AS}$) is asserted to indicate that there is a valid address on the address bus and the upper and lower data strobe ($\overline{UDS}$, $\overline{LDS}$) is asserted as required. The memory or peripheral device uses the address bus and the address strobe to determine if it has been selected. The selected device uses the read/write signal and the data strobe to place its information on the data bus. Concurrent with placing data on the data bus, the selected device asserts data transfer acknowledge ($\overline{DTACK}$).

Data transfer acknowledge must be present at the processor at the start of state 5 or the processor will substitute wait states for states 5 and 6. State 5 starts the synchronization of the returning data transfer acknowledge. At the end of state 6 (beginning of state 7) incoming data is latched into an internal data bus holding register.

During state 7, address strobe and the upper and/or lower data strobes are negated. The address bus is held valid through state 7 to allow for static memory operation and signal skew. The read/write signal and the function code outputs also remain valid through state 7 to ensure a correct transfer operation. The slave device keeps its data asserted until it detects the negation of either the address strobe or the upper and/or lower data strobe. The slave device must remove its data and data transfer acknowledge within one clock period of recognizing the negation of the address or data strobes. Note that the data bus might not become free and data transfer acknowledge might not be removed until state 0 or 1.

When address strobe is negated, the slave device is released. Note that a slave device must remain selected as long as address strobe is asserted to ensure the correct functioning of the read-modify-write cycle.

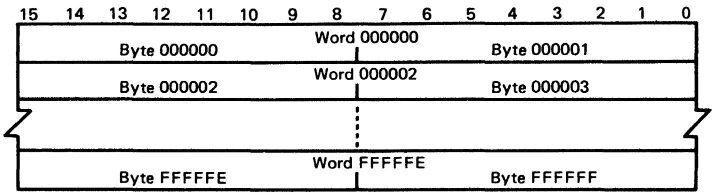


Figure 9 Word Organization in Memory

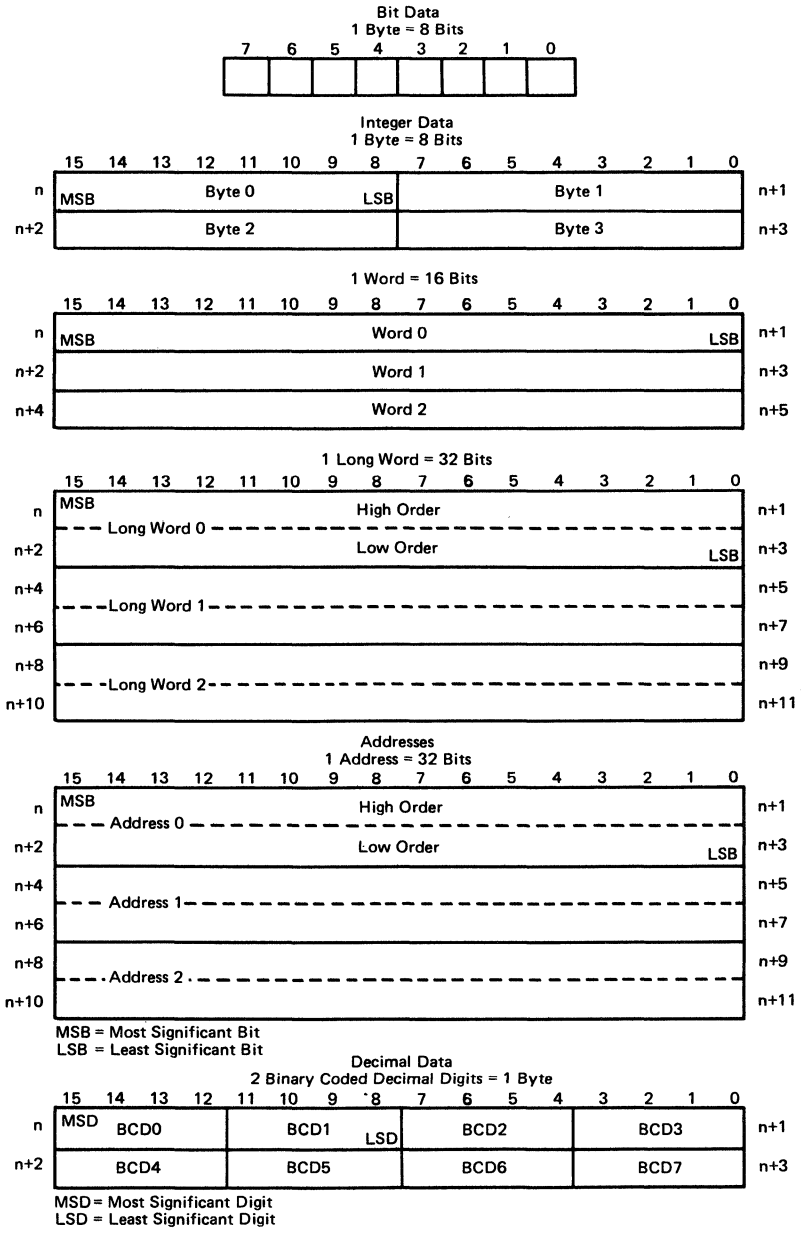


Figure 10 Data Organization in Memory

BUS MASTER **SLAVE**

Address Device
1) Set $R/\overline{W}$ to Read
2) Place Function Code on $FC_0 \sim FC_2$
3) Place Address on $A_1 \sim A_{23}$
4) Assert Address Strobe ($\overline{AS}$)
5) Assert Upper Data Strobe ($\overline{UDS}$) or Lower Data Strobe ($\overline{LDS}$)

↓

Input Data
1) Decode Address
2) Place Data on $D_0 \sim D_{15}$
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)

↓

Acquire Data
1) Latch Data
2) Negate $\overline{UDS}$ and $\overline{LDS}$
3) Negate $\overline{AS}$

↓

Terminate Cycle
1) Remove Data from $D_0 \sim D_{15}$
2) Negate $\overline{DTACK}$

↓

Start Next Cycle

Figure 11 Word Read Cycle Flow Chart

BUS MASTER **SLAVE**

Address Device
1) Set $R/\overline{W}$ to Read
2) Place Function Code on $FC_0 \sim FC_2$
3) Place Address on $A_1 \sim A_{23}$
4) Assert Address Strobe ($\overline{AS}$)
5) Assert Upper Data Strobe ($\overline{UDS}$) and Lower Data Strobe ($\overline{LDS}$) (based on A_0)

↓

Input Data
1) Decode Address
2) Place Data on $D_0 \sim D_7$ or $D_8 \sim D_{15}$ (based on $\overline{UDS}$ or $\overline{LDS}$)
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)

↓

Acquire Data
1) Latch Data
2) Negate $\overline{UDS}$ or $\overline{LDS}$
3) Negate $\overline{AS}$

↓

Terminate Cycle
1) Remove Data from $D_0 \sim D_7$ or $D_8 \sim D_{15}$
2) Negate $\overline{DTACK}$

↓

Start Next Cycle

Figure 12 Byte Read Cycle Flow Chart

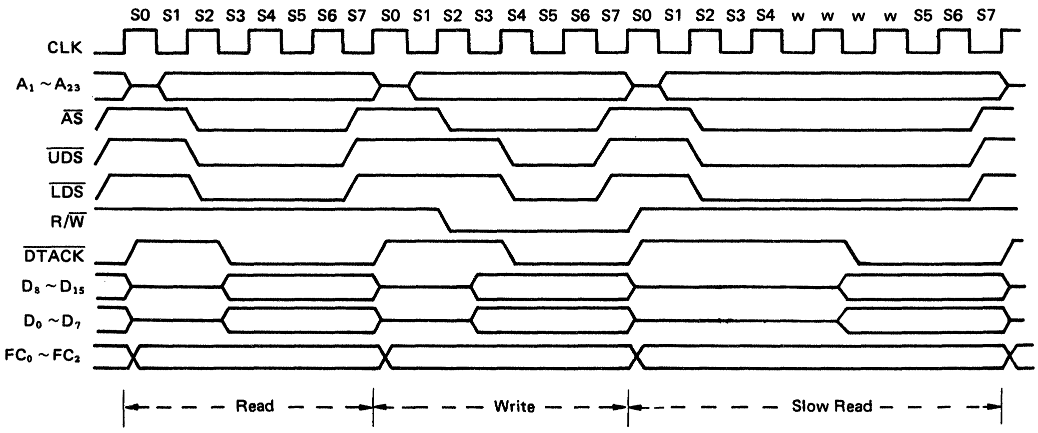

S0 S1 S2 S3 S4 S5 S6 S7 S0 S1 S2 S3 S4 S5 S6 S7 S0 S1 S2 S3 S4 w w w w S5 S6 S7
CLK
A1 ~ A23
AS
UDS
LDS
R/W
DTACK
D8 ~ D15
D0 ~ D7
FC0 ~ FC2
Read
Write
Slow Read

Figure 13 Read and Write Cycle Timing Diagram

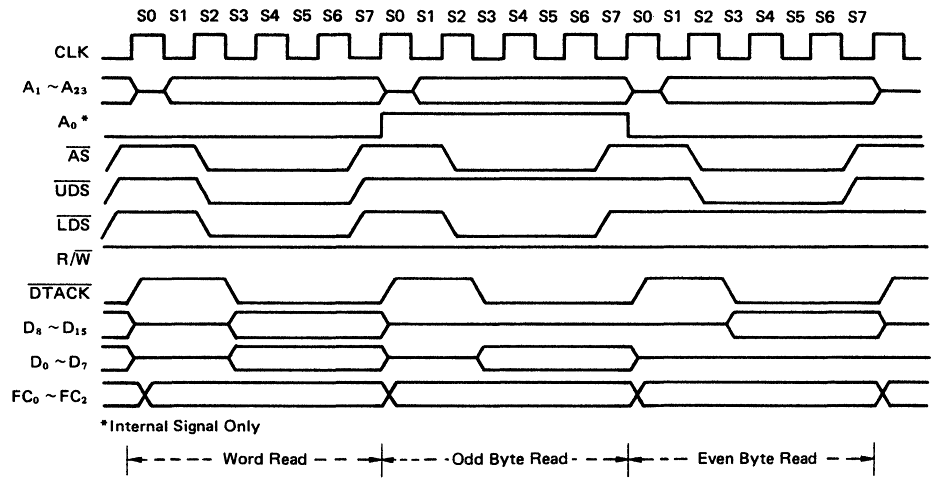


Figure 14 Word and Byte Read Cycle Timing Diagram

Write Cycle

During a write cycle, the processor sends data to memory or a peripheral device. The processor writes bytes of data in all cases. If the instruction specifies a word operation, the processor writes both bytes. When the instruction specifies a byte operation, the processor uses an internal A_0 bit to determine which byte to write and then issues the data strobe required for that byte. For byte operations, when the A_0 bit equals zero, the upper data strobe is issued. When the A_0 bit equals one, the lower data strobe is issued. A word write cycle flow chart is given in Figure 15. A byte write cycle flow chart is given in Figure 16. Write cycle timing is given in Figure 13. Figure 17 details word and byte write cycle operation. Refer to these illustrations during the following detailed discussion.

At state zero (S0) in the write cycle, the address bus (A_1 through A_{23}) is in the high impedance state. A function code is asserted on the function code output line (FC_0 through FC_2).

(NOTE) The read/write ($R/\overline{W}$) signal remains high until state 2 to prevent bus conflicts with preceding read cycles. The data bus is not driven until state 3.

One half clock later, at state 1, the address bus is released from the high impedance state. The function code outputs indicate which address space that this cycle will operate on.

In state 2, the address strobe ($\overline{AS}$) is asserted to indicate that there is a valid address on the address bus. The memory or peripheral device uses the address bus and the address strobe to determine if it has been selected. During state 2, the read/write signal is switched low to indicate a write cycle. When external processor data bus buffers are required, the read/write line provides sufficient directional control. Data is not asserted during this state to allow sufficient turn around time for external data buffers (if used). Data is asserted onto the data bus during state 3.

In state 4, the data strobes are asserted as required to indicate that the data bus is stable. The selected device uses the read/write signal and the data strobes to take its information from the data bus. The selected device asserts data transfer acknowledge ($\overline{DTACK}$) when it has successfully stored the data.

Data transfer acknowledge must be present at the processor at the start of state 5 or the processor will substitute wait states for states 5 and 6. State 5 starts the synchronization of the returning data transfer acknowledge.

During state 7, address strobe and the upper and/or lower data strobes are negated. The address and data buses are held valid through state 7 to allow for static memory operation and signal skew. The read/write signal and the function code outputs also remain valid through state 7 to ensure a correct transfer operation. The slave device keeps its data transfer acknowledge asserted until it detects the negation of either the address strobe or the upper and/or lower data strobe. The slave device must remove its data transfer acknowledge within one clock period after recognizing the negation of the address or data strobes. Note that the processor releases the data bus at the end of state 7 but that data transfer acknowledge might not be removed until state 0 or 1. When address strobe is negated, the slave device is released.

Read-Modify-Write Cycle

The read-modify-write cycle performs a read, modifies the data in the arithmetic-logic unit, and writes the data back to the same address. In the HD68000 this cycle is indivisible in that the address strobe is asserted throughout the entire cycle. The test and set (TAS) instruction uses this cycle to provide meaningful communication between processors in a multiple processor environment. This instruction is the only instruction that uses the read-modify-write cycle and since the test and set instruction only operates on bytes, all read-modify-write cycles are byte operations. A read-modify-write cycle flow chart is given in Figure 18 and a timing diagram is given in Figure 19. Refer to these illustrations during the following detailed discussions.

At state zero (S0) in the read-modify-write cycle, the address bus (A_1 through A_{23}) is in the high impedance state. A function code is asserted on the function code output line (FC_0 through FC_2). The read/write ($R/\overline{W}$) signal is switched high to indicate a read cycle. One half clock cycle later, at state 1, the address bus is released from the high impedance state. The function code outputs indicate which address space that this cycle will operate on.

In state 2, the address strobe (AS) is asserted to indicate that there is a valid address on the address bus and the upper or lower data strobe ($\overline{UDS}$, $\overline{LDS}$) is asserted as required. The memory or peripheral device uses the address bus and the address strobe to determine if it has been selected. The selected device uses the read/write signal and the data strobe to place its information on the data bus. Concurrent with placing data on the data bus, the selected device asserts data transfer acknowledge ($\overline{DTACK}$).

Data transfer acknowledge must be present at the processor at the start of state 5 or the processor will substitute wait states for states 5 and 6. State 5 starts the synchronization of the returning data transfer acknowledge. At the end of state 6 (beginning of state 7) incoming data is latched into an internal data bus holding register.

During state 7, the upper or lower data strobe is negated. The address bus, address strobe, read/write signal, and function code outputs remain as they were in preparation for the write portion of the cycle. The slave device keeps its data asserted until it detects the negation of the upper or lower data strobe. The slave device must remove its data and data transfer acknowledge within one clock period of recognizing the negation of the data strobes. Internal modification of data may occur from state 8 to state 11.

(NOTE) The read/write signal remains high until state 14 to prevent bus conflicts with the preceding read portion of the cycle and the data bus is not asserted by the processor until state 15.

In state 14, the read/write signal is switched low to indicate a write cycle. When external processor data bus buffers are required, the read/write line provides sufficient directional control. Data is not asserted during this state to allow sufficient turn around time for external data buffers (if used). Data is asserted onto the data bus during state 15.

In state 16, the data strobe is asserted as required to indicate that the data bus is stable. The selected device uses the read/write signal and the data strobe to take its information from the data bus. The selected device asserts data transfer acknowledge ($\overline{DTACK}$) when it has successfully stored its data.

Data transfer acknowledge must be present at the processor at the start of state 17 or the processor will substitute wait states for states 17 and 18. State 17 starts the synchronization of the returning data transfer acknowledge for the write portion of the cycle. The bus interface circuitry issues requests for subsequent internal cycles during state 18.

During state 19, address strobe and the upper or lower data strobe is negated. The address and data buses are held valid through state 19 to allow for static memory operation and signal skew. The read/write signal and the function code outputs also remain valid through state 19 to ensure a correct transfer operation. The slave device keeps its data transfer acknowledge asserted until it detects the negation of either the address strobe or the upper or lower data strobe. The slave device must remove its data transfer acknowledge within once clock period after recognizing the negation of the address or data strobes. Note that the processor releases the data bus at the end of state 19 but that data transfer acknowledge might not be removed until state 0 or 1. When address strobe is negated the slave device is released.

• BUS ARBITRATION

Bus arbitration is a technique used by master-type devices to request, be granted, and acknowledge bus mastership. In its simples form, it consists of:

(1) Asserting a bus mastership request.
(2) Receiving a grant that the bus is available at the end of the current cycle.
(3) Acknowledging that mastership has been assumed.

Figure 20 is a flow chart showing the detail involved in a request from a single device. Figure 21 is a timing diagram for the same operations. This technique allows processing of bus requests during data transfer cycles.

The timing diagram shows that the bus request is negated at the time that an acknowledge is asserted. This type of operation would be true for a system consisting of the processor and one device capable of bus mastership. In systems having a number of devices capable of bus mastership, the bus request line from each device is wire ORed to the processor. In this system, it is easy to see that there could be more that one bus request being made. The timing diagram shows that the bus grant signal is negated a few clock cycles after the transition of the acknowledge ($\overline{BGACK}$) signal.

However, if the bus requests are still pending, the processor will assert another bus grant within a few clock cycles after it was negated. This additional assertion of bus grant allows external arbitration circuitry to select the next bus master before the current bus master has completed its requirements. The following paragraphs provide additional information about the three steps in the arbitration process.

BUS MASTER **SLAVE**

Address Device

1) Place Function Code on $FC_0 \sim FC_2$
2) Place Address on $A_1 \sim A_{23}$
3) Assert Address strobe ($\overline{AS}$)
4) Set $R/\overline{W}$ to Write
5) Place Data on $D_0 \sim D_{15}$
6) Assert Upper Data Strobe ($\overline{UDS}$) and Lower Data Strobe ($\overline{LDS}$)

Input Data

1) Decode Address
2) Store Data on $D_0 \sim D_{15}$
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)

Terminate Output Transfer

1) Negate $\overline{UDS}$ and $\overline{LDS}$
2) Negate $\overline{AS}$
3) Remove Data from $D_0 \sim D_{15}$
4) Set $R/\overline{W}$ to Read

Terminate Cycle

1) Negate $\overline{DTACK}$

Start Next Cycle

Figure 15 Word Write Cycle Flow Chart

BUS MASTER **SLAVE**

Address Device

1) Place Function Code on $FC_0 \sim FC_2$
2) Place Address on $A_1 \sim A_{23}$
3) Assert Address Strobe ($\overline{AS}$)
4) Set $R/\overline{W}$ to Write
5) Place Data on $D_0 \sim D_7$ or $D_8 \sim D_{15}$ (according to A_0)
6) Assert Upper Data Strobe ($\overline{UDS}$) or Lower Data Strobe ($\overline{LDS}$) (based on A_0)

Input Data

1) Decode Address
2) Store Data on $D_0 \sim D_7$ if $\overline{LDS}$ is asserted
   Store Data on $D_8 \sim D_{15}$ if $\overline{UDS}$ is asserted
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)

Terminate Output Transfer

1) Negate $\overline{UDS}$ and $\overline{LDS}$
2) Negate $\overline{AS}$
3) Remove Data from $D_0 \sim D_7$ or $D_8 \sim D_{15}$
4) Set $R/\overline{W}$ to Read

Terminate Cycle

1) Negate $\overline{DTACK}$

Start Next Cycle

Figure 16 Byte Write Cycle Flow Chart

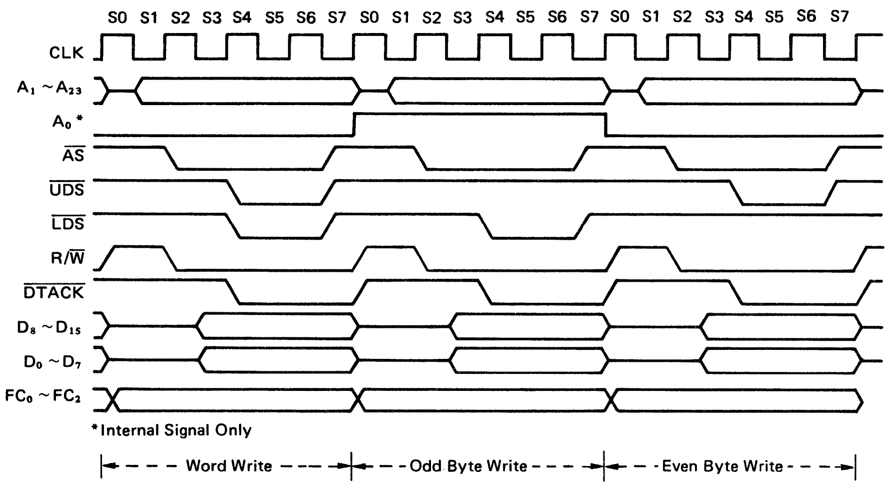
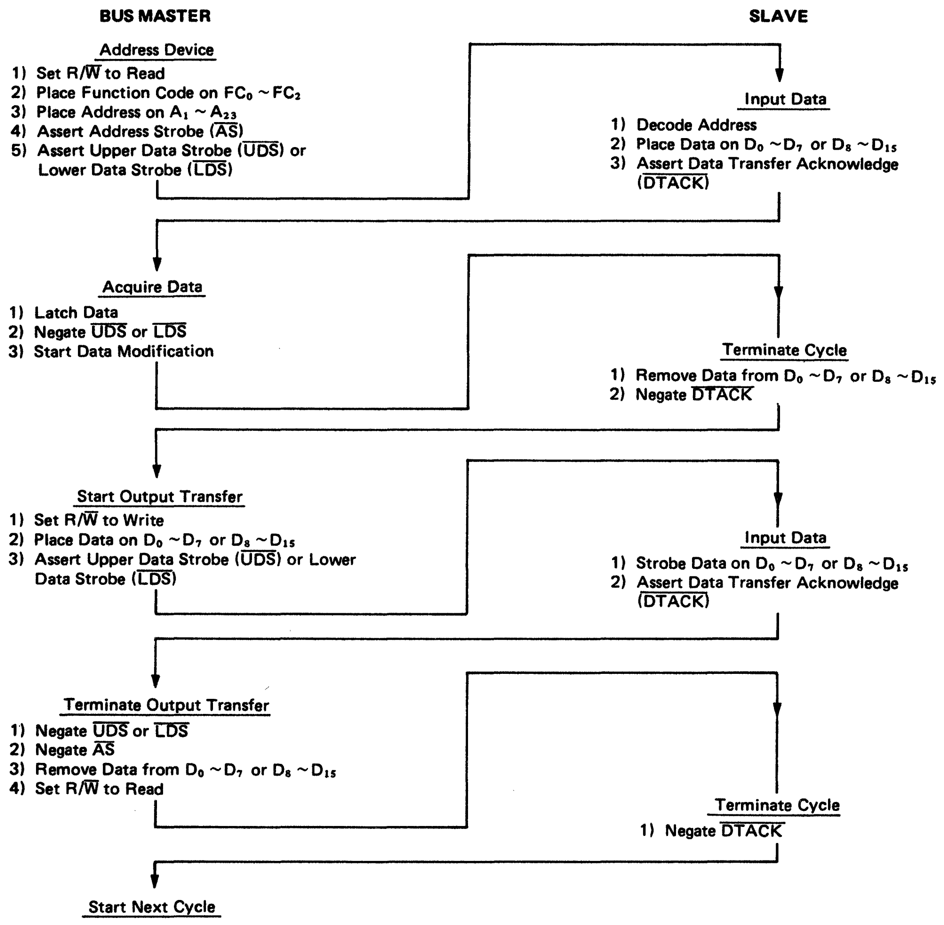


Figure 17 Word and Byte Write Cycle Timing Diagram



BUS MASTER

SLAVE

Address Device

1) Set R/$\overline{W}$ to Read
2) Place Function Code on $FC_0 \sim FC_2$
3) Place Address on $A_1 \sim A_{23}$
4) Assert Address Strobe ($\overline{AS}$)
5) Assert Upper Data Strobe ($\overline{UDS}$) or Lower Data Strobe ($\overline{LDS}$)

Input Data

1) Decode Address
2) Place Data on $D_0 \sim D_7$ or $D_8 \sim D_{15}$
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)

Acquire Data

1) Latch Data
2) Negate $\overline{UDS}$ or $\overline{LDS}$
3) Start Data Modification

Terminate Cycle

1) Remove Data from $D_0 \sim D_7$ or $D_8 \sim D_{15}$
2) Negate $\overline{DTACK}$

Start Output Transfer

1) Set R/$\overline{W}$ to Write
2) Place Data on $D_0 \sim D_7$ or $D_8 \sim D_{15}$
3) Assert Upper Data Strobe ($\overline{UDS}$) or Lower Data Strobe ($\overline{LDS}$)

Input Data

1) Strobe Data on $D_0 \sim D_7$ or $D_8 \sim D_{15}$
2) Assert Data Transfer Acknowledge ($\overline{DTACK}$)

Terminate Output Transfer

1) Negate $\overline{UDS}$ or $\overline{LDS}$
2) Negate $\overline{AS}$
3) Remove Data from $D_0 \sim D_7$ or $D_8 \sim D_{15}$
4) Set R/$\overline{W}$ to Read

Terminate Cycle

1) Negate $\overline{DTACK}$

Start Next Cycle

Figure 18 Read-Modify-Write Cycle Flow Chart

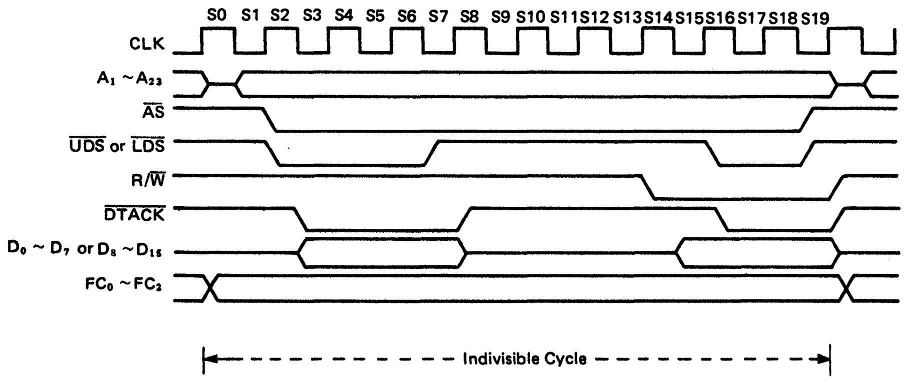
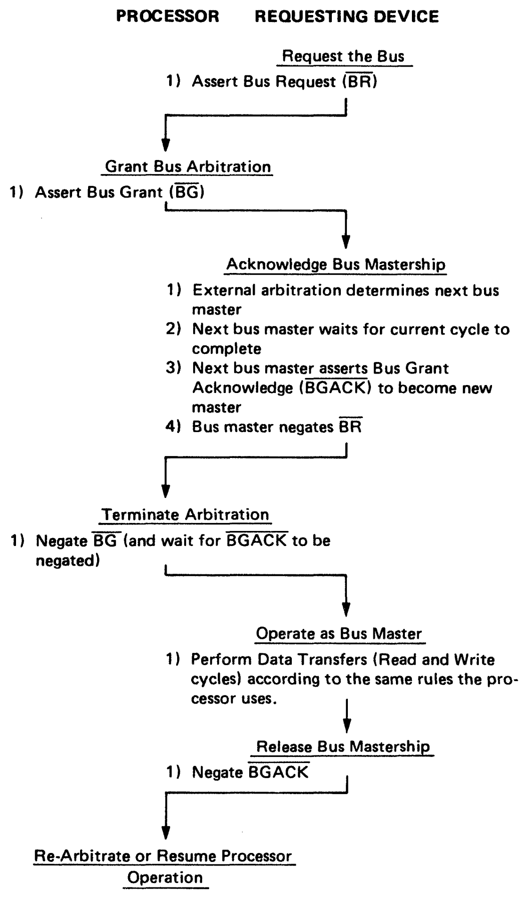

S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19
CLK
$A_1 \sim A_{23}$
$\overline{AS}$
$\overline{UDS}$ or $\overline{LDS}$
R/$\overline{W}$
$\overline{DTACK}$
$D_0 \sim D_7$ or $D_8 \sim D_{15}$
$FC_0 \sim FC_2$
Indivisible Cycle

Figure 19 Read-Modify-Write Cycle Timing Diagram



PROCESSOR REQUESTING DEVICE

Request the Bus
1) Assert Bus Request ($\overline{BR}$)

Grant Bus Arbitration
1) Assert Bus Grant ($\overline{BG}$)

Acknowledge Bus Mastership
1) External arbitration determines next bus master
2) Next bus master waits for current cycle to complete
3) Next bus master asserts Bus Grant Acknowledge ($\overline{BGACK}$) to become new master
4) Bus master negates $\overline{BR}$

Terminate Arbitration
1) Negate $\overline{BG}$ (and wait for $\overline{BGACK}$ to be negated)

Operate as Bus Master
1) Perform Data Transfers (Read and Write cycles) according to the same rules the processor uses.

Release Bus Mastership
1) Negate $\overline{BGACK}$

Re-Arbitrate or Resume Processor Operation

Figure 20 Bus Arbitration Cycle Flow Chart

Requesting the Bus

External devices capable of becoming bus masters request the bus by asserting the bus request ($\overline{BR}$) signal. This is a wire ORed signal (although it need not be constructed from open collector devices) that indicates to the processor that some external device requires control of the external bus. The processor is effectively at a lower bus priority level that the external device and will relinquish the bus after it has completed the last bus cycle it has started.

When no acknowledge is received before the bus request signal goes inactive, the processor will continue processing when it detects that the bus request is inactive. This allows ordinary processing to continue if the arbitration circuitry responded to noise inadvertently.

Receiving the Bus Grant

The processor asserts bus grant ($\overline{BG}$) as soon as possible. Normally this is immediately after internal synchronization. The only exception to this occurs when the processor has made an internal decision to execute the next bus cycle but has not progressed far enough into the cycle to have asserted the address strobe ($\overline{AS}$) signal. In this case, bus grant will not be asserted until one clock after address strobe is asserted to indicate to external devices that a bus cycle is being executed.

The bus grant signal may be routed through a daisy-chained network or through a specific priority-encoded network. The processor is not affected by the external method of arbitration as long as the protocol is obeyed.

Acknowledgement of Mastership

Upon receiving a bus grant, the requesting device waits until address strobe, data transfer acknowledge, and bus grant acknowledge are negated before issuing its own $\overline{BGACK}$. The negation of the address strobe indicates that the previous master has completed its cycle, the negation of bus grant acknowledge indicates that the previous master has released the bus. (While address strobe is asserted no device is allowed to "break into" a cycle.) The negation of data transfer acknowledge indicates the previous slave has terminated its connection to the previous master. Note that in some applications data

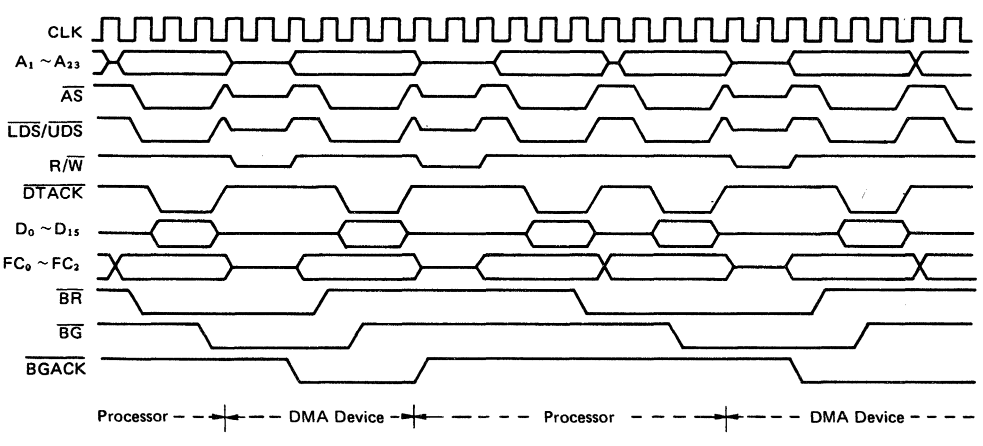


Figure 21 Bus Arbitration Cycle Timing Diagram

transfer acknowledge might not enter into this function. General purpose devices would then be connected such that they were only dependent on address strobe. When bus grant acknowledge is issued the device is bus master until it negates bus grant acknowledge. Bus grant acknowledge should not be negated until after the bus cycle(s) is (are) completed. Bus mastership is terminated at the negation of bus grant acknowledge.

The bus request from the granted device should be dropped after bus grant acknowledge is asserted. If a bus request is still pending, another bus grant will be asserted within a few clocks of the negation of bus grant. Refer to Bus Arbitration Control section. Note that the processor does not perform any external bus cycles before it re-asserts bus grant.

• BUS ARBITRATION CONTROL

The bus arbitration control unit in the HD68000 is implemented with a finite state machine. A state diagram of this machine is shown in Figure 22. All asynchronous signals to the HD68000 are synchronized before being used internally. This synchronization is accomplished in a maximum of one cycle of the system clock, assuming that the asynchronous input setup time (#47) has been met (see Figure 23). The input signal is sampled on the falling edge of the clock and is valid internally after the next falling edge.

As shown in Figure 22, input signals labeled R and A are internally synchronized on the bus request and bus grant acknowledge pins respectively. The bus grant output is lebeled G and the internal three-state control signal T. If T is true, the address, data, function code line, and control buses are placed in a high-impedance state when $\overline{AS}$ is negated. All signals are shown in positive logic (active high) regardless of their true active voltage level.

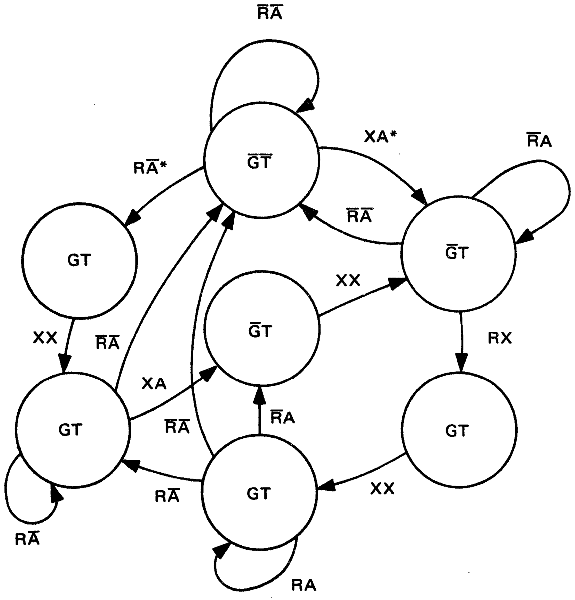
State changes (valid outputs) occur on the next rising edge after the internal signal is valid.

A timing diagram of the bus arbitration sequence during a processor bus cycle is shown in Figure 24. The bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 25.

If a bus request is made at a time when the MPU has already begun a bus cycle but $\overline{AS}$ has not been asserted (bus state S0), $\overline{BG}$ will not be asserted on the next rising edge. Instead, $\overline{BG}$ will be delayed until the second rising edge following it's internal assertion. This sequence is shown in Figure 26.

• BUS ERROR AND HALT OPERATION

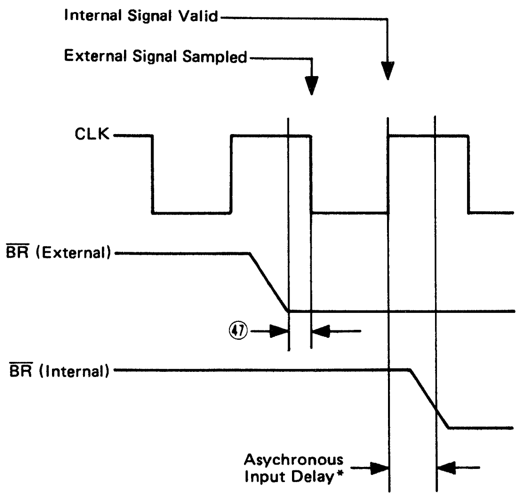
In a bus architecture that requires a handshake from an external device, the possibility exists that the handshake might not occur. Since different systems will require a different maximum response time, a bus error input is provided. External circuitry must be used to determine the duration between address strobe and data transfer acknowledge before issuing a bus error signal. When a bus error signal is received, the processor has two options initiate a bus error exception sequence or try running the bus cycle again.



R = Bus Request Internal
A = Bus Grant Acknowledge Internal
G = Bus Grant
T = Three-State Control to Bus Control Logic
X = Don't Care

* State machine will not change state if bus is in S0. Refer to BUS ARBITRATION CONTROL for additional information.

Figure 22 State Diagram of HD68000 Bus Arbitration Unit



* This delay time is equal to parameter #33, t_{CHGL}.

Figure 23 Timing Relationship of External Asynchronous Inputs to Internal Signals

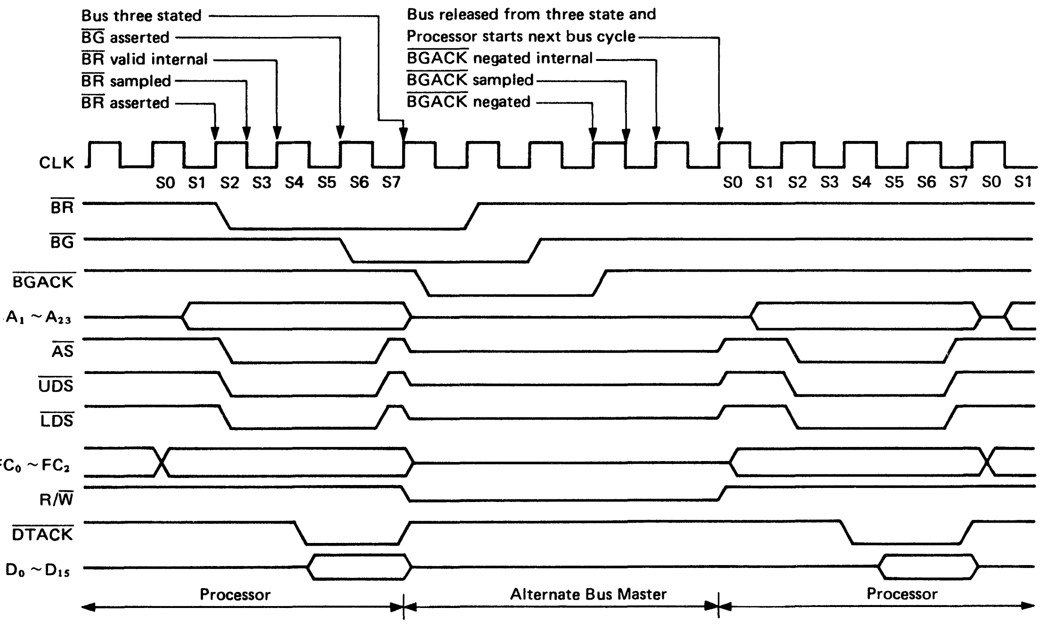


Figure 24 Bus Arbitration During Processor Bus Cycle

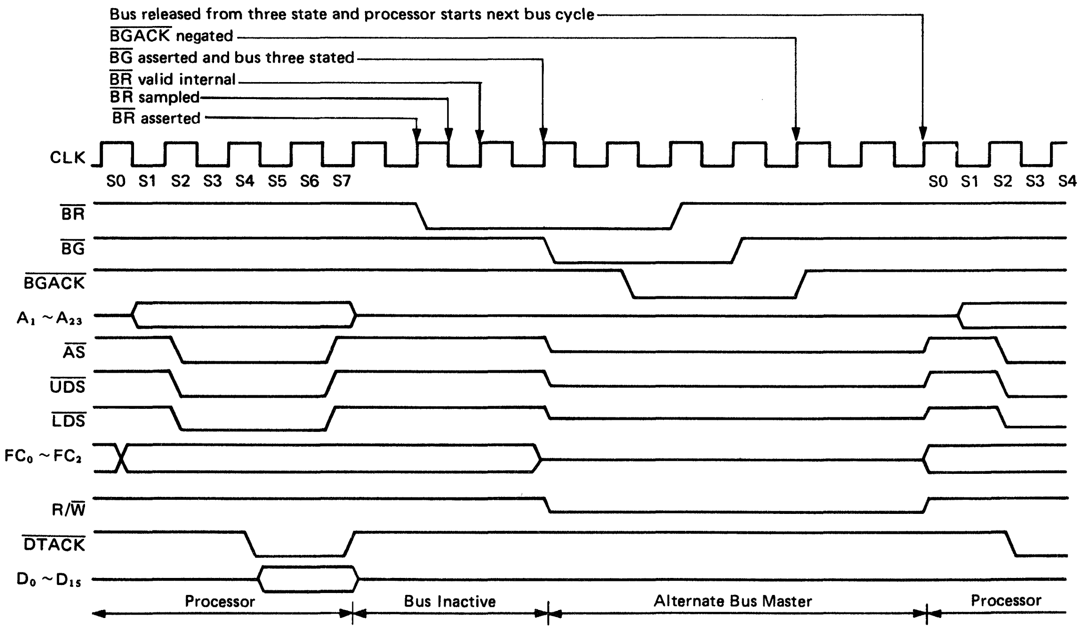


Figure 25 Bus Arbitration with Bus Inactive

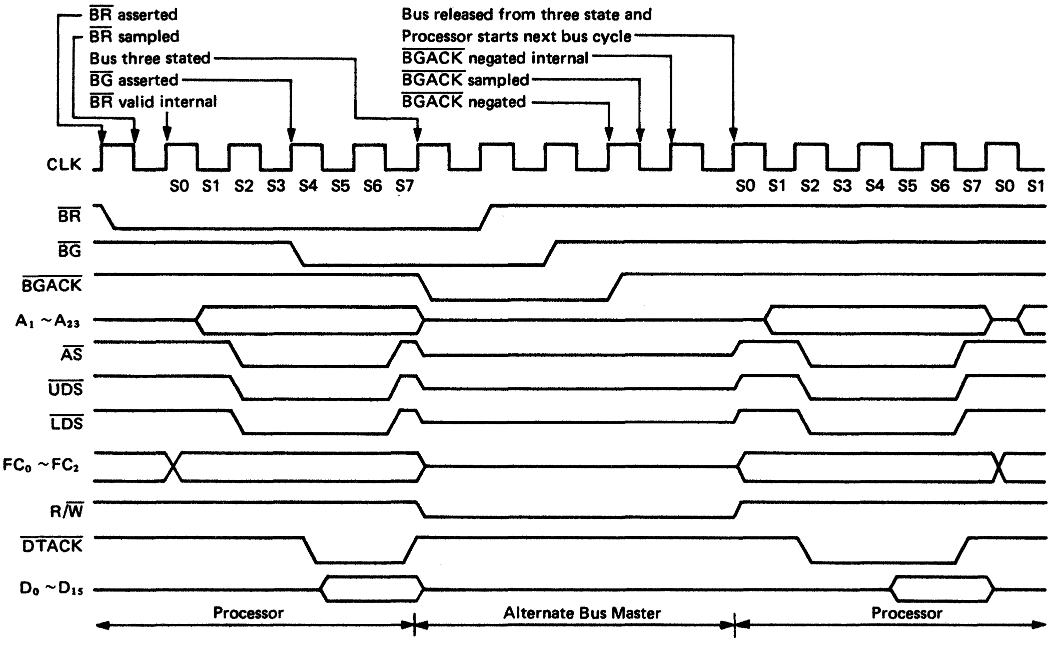


Figure 26 Bus Arbitration During Processor Bus Cycle Special Case

Exception Sequence

When the bus error signal is asserted, the current bus cycle is terminated. If BERR is asserted before the falling edge of S4, AS will be negated in S7 in either a read or write cycle. As long as BERR remains asserted, the data and address buses will be in the high-impedance state. When BERR is negated, the processor will begin stacking for exception processing. Figure 27 is a timing diagram for the exception sequence. The sequence is composed of the following elements.

(1) Stacking the program counter and status register
(2) Stacking the error information
(3) Reading the bus error vector table entry
(4) Executing the bus error handler routine

The stacking of the program counter and the status register is the same as if an interrupt had occurred. Several additional items are stacked when a bus error occurs. These items are used to determine the nature of the error and correct it, if possible. The bus error vector is vector number two located at address $000008. The processor loads the new program counter from this location. A software bus error handler routine is then executed by the processor. Refer to **EXCEPTION PROCESSING** for additional information.

Re-Running the Bus Cycle

When, during a bus cycle, the processor receives a bus error signal and the halt pin is being driven by an external device, the processor enters the re-run sequence. Figure 28 is a timing diagram for re-running the bus cycle.

The processor terminates the bus cycle, then puts the address and data output lines in the high-impedance state. The processor remains "halted," and will not run another bus cycle until the halt signal is removed by external logic. Then the processor will re-run the previous bus cycle using the same address, the same function codes, the same data (for a write operation), and the same controls. The bus error signal should be removed at least one clock cycle before the halt signal is removed.

(NOTE) The processor will not re-run a read-modify-write cycle. This restriction is made to guarantee that the entire cycle runs correctly and that the write operation of a Test-and-Set operation is performed without ever releasing $\overline{AS}$. If $\overline{BERR}$ and $\overline{HALT}$ are asserted during a read-modify-write bus cycle, a bus error operation results.

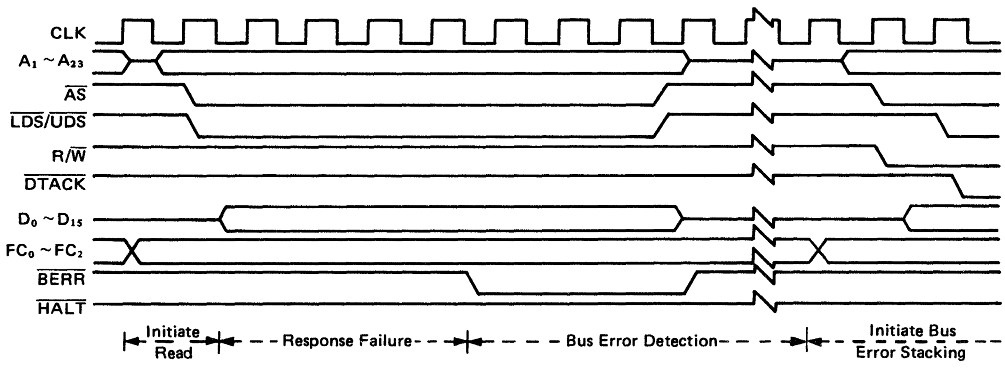


Figure 27 Bus Error Timing Diagram

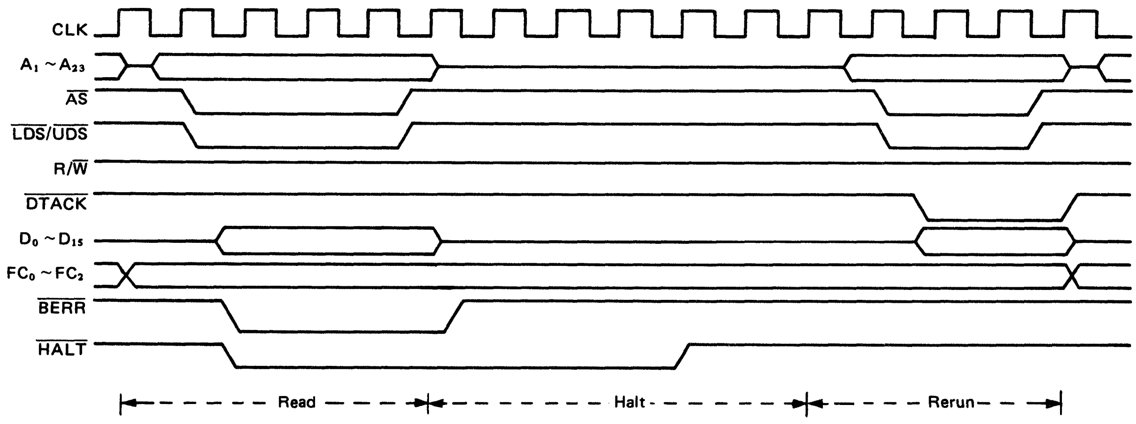


Figure 28 Re-Run Bus Cycle Timing Information

The processor terminates the bus cycle, then puts the address, data and function code output lines in the high-impedance state. The processor remains "halted," and will not run another bus cycle until the halt signal is removed by external logic. Then the processor will re-run the previous bus cycle using the same address, the same function codes, the same data (for a write operation), and the same controls. The bus error signal should be removed before the halt signal is removed.

Halt Operation with No Bus Error

The halt input signal to the HD68000 perform a Halt/Run/Single-Step function in a similar fashion to the HMCS6800 halt function. The halt and run modes are somewhat self explanatory in that when the halt signal is constantly active the processor "halts" (does nothing) and when the halt signal is constantly inactive the processor "runs" (does something).

The single-step mode is derived from correctly timed transitions on the halt signal input. It forces the processor to execute a single bus cycle by entering the "run" mode until the processor starts a bus cycle then changing to the "halt" mode. Thus, the single-step mode allows the user to proceed through (and therefore debug) processor operations one bus cycle at a time.

Figure 29 details the timing required for correct single-step operations. Some care must be exercised to avoid harmful interactions between the bus error signal and the halt pin when using the single cycle mode as a debugging tool. This is also true of interactions between the halt and reset lines since these can reset the machine.

When the processor completes a bus cycle after recognizing that the halt signal is active, most three-state signals are put in the high-impedance state. These include:

(1) Address lines
(2) Data lines

This is required for correct performance of the re-run bus cycle operation.

While the processor is honoring the halt request, bus arbitration performs as usual. That is, halting has no effect on bus arbitration. It is the bus arbitration function that removes the control signals from the bus.

The halt function and the hardware trace capability allow the hardware debugger to trace single bus cycles or single instructions at a time. These processor capabilities, along with a software debugging package, give total debugging flexibility.

Double Bus Faults

When a bus error exception occurs, the processor will attempt to stack several words containing information about the state of the machine. If a bus error exception occurs during the stacking operation, there have been two bus errors in a row. This is commonly referred to as a double bus fault. When a double bus fault occurs, the processor will halt. Once a bus error exception has occurred, any bus error exception occurring before the execution of the next instruction constitutes a double bus fault.

Note that a bus cycle which is re-run does not constitute a bus error exception, and does not contribute to a double bus fault. Note also that this means that as long as the external hardware requests it, the processor will continue to re-run the same bus cycle.

The bus error pin also has an effect on processor operation after the processor receives an external reset input. The processor reads the vector table after a reset to determine the address to start program execution. If a bus error occurs while reading the vector table (or at any time before the first instruction is executed), the processor reacts as if a double bus fault has occurred and it halts. Only an external reset will start a halted processor.

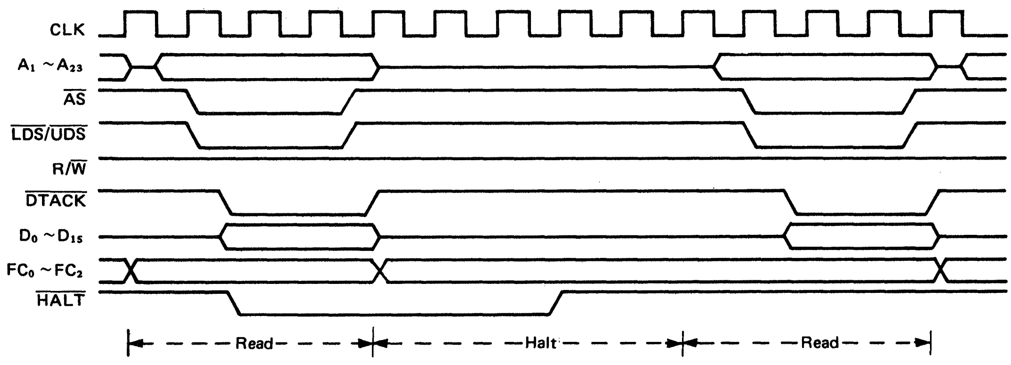


Figure 29 Halt Signal Timing Characteristics

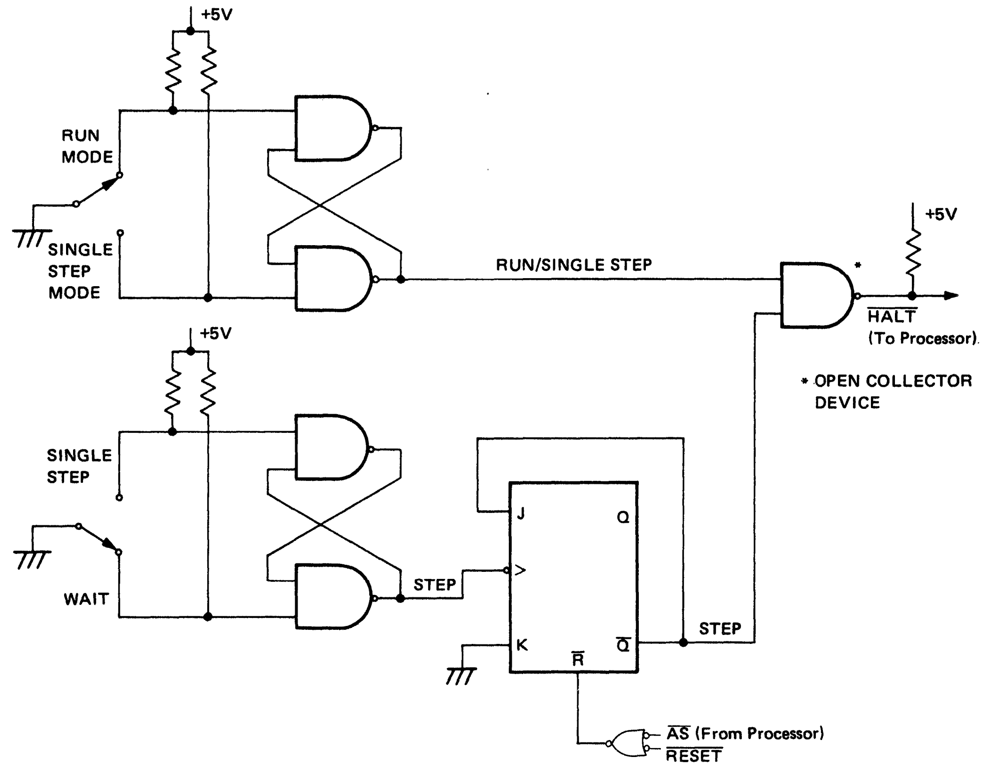


Figure 30 Simplified Single-Step Circuit

■ THE RELATIONSHIP OF $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, AND $\overline{\text{HALT}}$

In order to properly control termination of a bus cycle for a re-run or a bus error condition, $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ should be asserted and negated on the rising edge of the HD68000 clock. This will assure that when two signals are asserted simultaneously, the required setup time (#47) for both of them will be met during the same bus state.

This, or some equivalent precaution, should be designed external to the HD68000. Parameter #48 is intended to ensure this operation in a totally asynchronous system, and may be ignored if the above conditions are met.

The preferred bus cycle terminations may be summarized as follows (case numbers refer to Table 4):

Normal Termination: $\overline{\text{DTACK}}$ occurs first (case 1).

Halt Termination: $\overline{\text{HALT}}$ is asserted at same time, or precedes $\overline{\text{DTACK}}$ (no $\overline{\text{BERR}}$) cases 2 and 3.

Bus Error Termination: $\overline{\text{BERR}}$ is asserted in lieu of, at same time, or preceding $\overline{\text{DTACK}}$ (case 4); $\overline{\text{BERR}}$ negated at same time, or after $\overline{\text{DTACK}}$.

Re-Run Termination: $\overline{\text{HALT}}$ and $\overline{\text{BERR}}$ asserted at the same time, or before $\overline{\text{DTACK}}$ (cases 6 and 7); $\overline{\text{HALT}}$ must be negated at least 1 cycle after $\overline{\text{BERR}}$. (Case 5 indicates $\overline{\text{BERR}}$ may precede $\overline{\text{HALT}}$ which allows fully asynchronous assertion).*

Table 4 details the resulting bus cycle termination under various combinations of control signal sequences. The negation of these same control signals under several conditions is shown in Table 5 ($\overline{\text{DTACK}}$ is assumed to be negated normally in all cases; for best results, both $\overline{\text{DTACK}}$ and $\overline{\text{BERR}}$ should be negated when address strobe is negated.)

Example A: A system uses a watch-dog timer to terminate accesses to un-populated address space. The timer asserts $\overline{\text{DTACK}}$ and $\overline{\text{BERR}}$ simultaneously after time-out. (case 4)

Example B: A system uses error detection on RAM contents. Designer may (a) delay $\overline{\text{DTACK}}$ until data verified, and return $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ simultaneously to re-run error cycle (case 6), or if valid, return $\overline{\text{DTACK}}$; (b) delay $\overline{\text{DTACK}}$ until data verified, and return $\overline{\text{BERR}}$ at same time as $\overline{\text{DTACK}}$ if data in error (case 4); (c) return $\overline{\text{DTACK}}$ prior to data verification, as described in previous section. If data invalid, $\overline{\text{BERR}}$ is asserted (case 1) in next cycle. Error-handling software must know how to recover error cycle.

* For the mask version 68000, $\overline{\text{HALT}}$ and $\overline{\text{BERR}}$ must be asserted at the same time.

Table 4 $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, $\overline{\text{HALT}}$ Assertion Results

Case No.	Control Signal	Asserted on Rising Edge of State		Result
		N	N + 2	
	$\overline{\text{DTACK}}$	A	S	
1	$\overline{\text{BERR}}$	NA	X	Normal cycle terminate and continue.
	$\overline{\text{HALT}}$	NA	X	
	$\overline{\text{DTACK}}$	A	S	
2	$\overline{\text{BERR}}$	NA	X	Normal cycle terminate and halt. Continue when HALT removed.
	$\overline{\text{HALT}}$	A	S	
	$\overline{\text{DTACK}}$	NA	A	
3	$\overline{\text{BERR}}$	NA	NA	Normal cycle terminate and halt. Continue when HALT removed.
	$\overline{\text{HALT}}$	A	S	
	$\overline{\text{DTACK}}$	X	X	
4	$\overline{\text{BERR}}$	A	S	Terminate and take bus error trap.
	$\overline{\text{HALT}}$	NA	NA	
	$\overline{\text{DTACK}}$	NA	X	
5	$\overline{\text{BERR}}$	A	S	Terminate and re-run*.
	$\overline{\text{HALT}}$	NA	A	
	$\overline{\text{DTACK}}$	X	X	
6	$\overline{\text{BERR}}$	A	S	Terminate and re-run.
	$\overline{\text{HALT}}$	A	S	
	$\overline{\text{DTACK}}$	NA	X	
7	$\overline{\text{BERR}}$	NA	A	Terminate and re-run when HALT removed.
	$\overline{\text{HALT}}$	A	S	

Legend:
N — The number of the current even bus state (e.g., S4, S6, etc.)
A — Signal is asserted in this bus state
NA — Signal is not asserted in this state
X — Don't care
S — Signal was asserted in previous state and remains asserted in this state

* For the mask version 68000, unpredictable results, no re-run, no error trap; usually traps to vector number 0.

Table 5 $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ Negation Results

Conditions of Termination in Table A	Control Signal	Negated on Rising Edge of State			Results — Next Cycle
		N		N + 2	
Bus Error	$\overline{\text{BERR}}$	●	or	●	Takes bus error trap.
	$\overline{\text{HALT}}$	●	or	●	
Re-run	$\overline{\text{BERR}}$	●	or	●	Illegal sequence; usually traps to vector number 0.
	$\overline{\text{HALT}}$	●			
Re-run	$\overline{\text{BERR}}$	●			Re-runs the bus cycle.
	$\overline{\text{HALT}}$			●	
Normal	$\overline{\text{BERR}}$	●			May lengthen next cycle.
	$\overline{\text{HALT}}$	●	or	●	
Normal	$\overline{\text{BERR}}$			●	If next cycle is started it will be terminated as a bus error.
	$\overline{\text{HALT}}$	●	or	none	

• RESET OPERATION

The reset signal is a bidirectional signal that allows either the processor or an external signal to reset the system. Figure 31 is a timing diagram for reset operations. Both the halt and reset lines must be applied to ensure total reset of the processor.

When the reset and halt lines are driven by an external device, it is recognized as an entire system reset, including the processor. The processor responds by reading the reset vector table entry (vector unumber zero, address \$000000) and loads it into the supervisor stack pointer (SSP). Vector table entry number one at address \$000004 is read next and loaded into the program counter. The processor initializes the status register to an interrupt level of seven. No other registers are affected by the reset sequence.

When a RESET sequence is executed, the processor drives the reset pin for 124 clock pulses. In this case, the processor is trying to reset the rest of the system. Therefore, there is no effect on the internal state of the processor. All of the processor's internal registers and the status register are unaffected by the execution of a RESET instruction. All external devices connected to the reset line should be reset at the completion of the RESET instruction.

Asserting the Reset and Halt pins for 10 clock cycles will cause a processor reset, except when V_{CC} is initially applied to the processor. In this case, an external reset must be applied for 100 milliseconds.

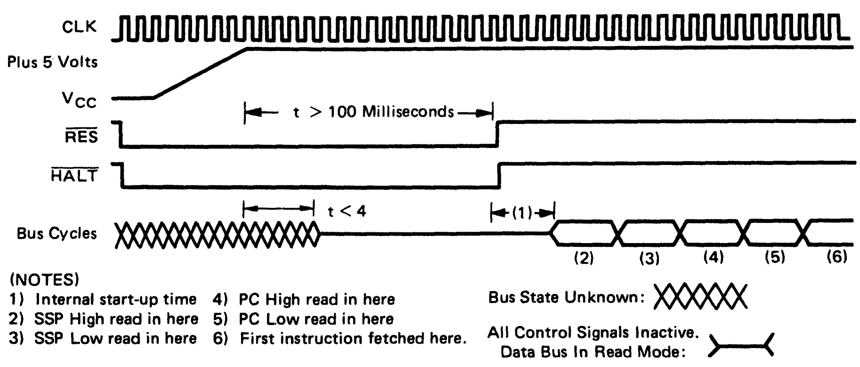


Figure 31 Reset Operation Timing Diagram

■ PROCESSING STATES

This section describes the HD68000 which are outside the normal processing associated with the execution of instructions. The functions of the bits in the supervisor portion of the status register are covered: the supervisor/user bit, the trace enable bit, and the processor interrupt priority mask. Finally, the sequence of memory references and actions taken by the processor on exception conditions is detailed.

The HD68000 is always in one of three processing states: normal, exception, or halted. The normal processing state is that associated with instruction execution; the memory references are to fetch instructions and operands, and to store results. A special case of the normal state is the stopped state which the processor enters when a STOP instruction is executed. In this state, no further memory references are made.

The exception processing state is associated with interrupts, trap instructions, tracing and other exceptional conditions. The exception may be internally generated by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, by a bus error, or by a reset. Exception processing is designed to provide an efficient context switch so that the processor may handle unusual conditions.

The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes that the system is unusable and halts. Only an external reset can restart a halted processor. Note that a processor in the stopped state is not in the halted state, nor vice versa.

PROCESSING STATES

NORMAL	INSTRUCTION EXECUTION (INCLUDING STOP)
EXCEPTION	INTERRUPTS TRAPS TRACING ETC.
HALTED	HARDWARE HALT DOUBLE BUS FAULT

● PRIVILEGE STATES

The processor operates in one of two states of privilege: the "user" state or the "supervisor" state. The privilege state determines which operations are legal, is used by the external memory management device to control and translate accesses, and is used to choose between the supervisor stack pointer and the user stack pointer in instruction references.

The privileges state is a mechanism for providing security in a computer system. Programs should access only their own code and data areas, and ought to be restricted from accessing information which they do not need and must not modify.

The privilege mechanism provides security by allowing most programs to execute in user state. In this state, the accesses are controlled, and the effects on other parts of the system are limited. The operating system executes in the supervisor state, has access to all resources, and performs the overhead tasks for the user state programs.

SUPERVISOR STATE

The supervisor state is the higher state of privilege. For instruction execution, the supervisor state is determined by the S-bit of the status register; if the S-bit is asserted (high), the processor is in the supervisor state. All instructions can be executed in the supervisor state. The bus cycles generated by instructions executed in the supervisor state are classified as supervisor references. While the processor is in the supervisor privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly access the supervisor stack pointer.

All exception processing is done in the supervisor state, regardless of the setting of the S-bit. The bus cycles generated during exception processing are classified as supervisor references. All stacking operations during exception processing use the supervisor stack pointer.

USER STATE

The user state is the lower state of privilege. For instruction execution, the user state is determined by the S-bit of the status register; if the S-bit is negated (low), the processor is executing instructions in the user state.

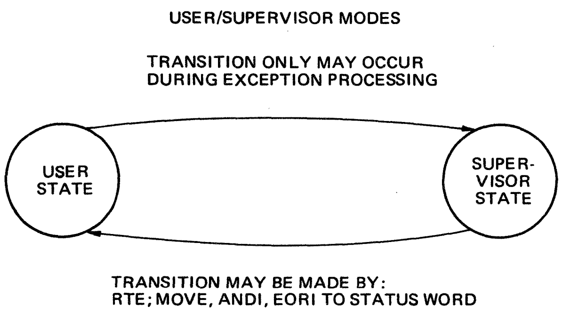
Most instructions execute the same in user state as in the supervisor state. However, some instructions which have important system effects are made privileged. User programs are not permitted to execute the STOP instruction, or the

RESET instruction. To ensure that a user program cannot enter the supervisor state except in a controlled manner, the instructions which modify the whole status register are privileged. To aid in debugging programs which are to be used as operating systems, the move to user stack pointer (MOVE USP) and move from user stack pointer (MOVE from USP) instructions are also privileged.

The bus cycles generated by an instruction executed in user state are classified as user state references. This allows an external memory management device to translate the address and to control access to protected portions of the address space. While the processor is in the user privilege state, those instructions which use either the system stack pointer implicitly, or address register seven explicitly, access the use stack pointer.

PRIVILEGE STATE CHANGES

Once the processor is in the user state and executing instructions, only exception processing can change the privilege state. During exception processing, the current setting of the S-bit of the status register is saved and the S-bit is asserted, putting the processing in the supervisor state. Therefore, when instruction execution resumes at the address specified to process the exception, the processor is in the supervisor privilege state.


USER/SUPERVISOR MODES
TRANSITION ONLY MAY OCCUR DURING EXCEPTION PROCESSING
USER STATE
SUPER-VISOR STATE
TRANSITION MAY BE MADE BY: RTE; MOVE, ANDI, EORI TO STATUS WORD

REFERENCE CLASSIFICATION

When the processor makes a reference, it classifies the kind of reference being made, using the encoding on the three function code output lines. This allows external translation of addresses, control of access, and differentiation of special processor states, such as interrupt acknowledge. Table 6 lists the classification of references.

Table 6 Reference Classification

Function Code Output			Reference Class
FC_2	FC_1	FC_0	
0	0	0	(Unassigned)
0	0	1	User Data
0	1	0	User Program
0	1	1	(Unassigned)
1	0	0	(Unassigned)
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	Interrupt Acknowledge

• EXCEPTION PROCESSING

Before discussing the details of interrupts, traps, and tracing, a general description of exception processing is in order. The processing of an exception occurs in four steps, with variations for different exception causes. During the first step, a temporary copy of the status register is made, and the status register is set for exception processing. In the second step the exception vector is determined, and the third step is the saving of the current processor context. In the fourth step a new context is obtained, and the processor switches to instruction processing.

EXCEPTION VECTORS

Exception vectors are memory locations from which the processor fetches the address of a routine which will handle that exception. All exception vectors are two words in length (Figure 32), except for the reset vector, which is four words. All exception vectors lie in the supervisor data space, except for the reset vector which is in the supervisor program space. A vector number is an eight-bit number which, when multiplied by four, gives the address of an exception vector. Vector numbers are generated internally or externally depending on the cause of the exception. In the case of interrupts, during the interrupt acknowledge bus cycle, a peripheral provides an 8-bit vector number (Figure 33) to the processor on data bus lines D_0 through D_7. The processor translates the vector number into a full 24-bit address, as shown in Figure 34. The memory layout for exception vectors is given in Table 7.

As shown in Table 7, the memory layout is 512 words long (1024 bytes). It starts at address 0 and proceeds through address 1023. This provides 255 unique vectors; some of these are reserved for TRAPS and other system functions. Of the 255, there are 192 reserved for user interrupt vectors. However, there is no protection on the first 64 entries, so user interrupt vectors may overlap at the discretion of the systems designer.

KINDS OF EXCEPTIONS

Exceptions can be generated by either internal or external causes. The externally generated exceptions are the interrupts and the bus error and reset requests. The interrupts are requests from peripheral devices for processor action while the bus error and reset inputs are used for access control and processor restart. The internally generated exceptions come from instructions, or from address error or tracing. The trap (TRAP), trap on overflow (TRAPV), check register against bounds (CHK) and divide (DIV) instructions all can generate exceptions as part of their instruction execution. In addition, illegal instructions, word fetches from odd addresses and privilege violations cause exceptions. Tracing behaves like a very high priority, internally generated interrupt after each instruction execution.

EXCEPTION PROCESSING SEQUENCE

Exception processing occurs in four identifiable steps. In the first step, an internal copy is made of the status register. After the copy is made, the S-bit is asserted, putting the processor into the supervisor privilege state. Also, the T-bit is negated which will allow the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated.

In the second step, the vector number of the exception is determined. For interrupts, the vector number is obtained by a processor fetch, classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the address of the exception vector.

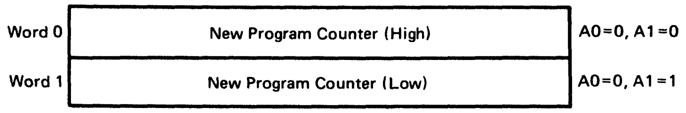
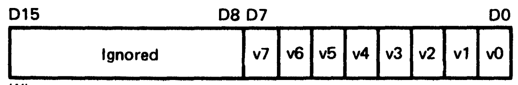


Figure 32 Exception Vector Format



Where:
v7 is the MSB of the Vector Number
v0 is the LSB of the Vector Number

Figure 33 Peripheral Vector Number Format

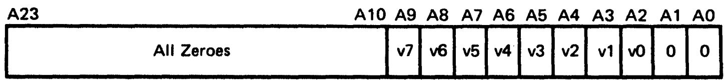


Figure 34 Address Translated From 8-Bit Vector Number

Table 7 Exception Vector Assignment

Vector Number(s)	Address			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial SSP
–	4	004	SP	Reset: Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12*	48	030	SD	(Unassigned, reserved)
13*	52	034	SD	(Unassigned, reserved)
14*	56	038	SD	(Unassigned, reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16 ~ 23*	64	04C	SD	(Unassigned, reserved)
	95	05F		
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32 ~ 47	128	080	SD	TRAP Instruction Vectors
	191	0BF		
48 ~ 63*	192	0C0	SD	(Unassigned, reserved)
	255	0FF		
64 ~ 255	256	100	SD	User Interrupt Vectors
	1023	3FF		

SP: Supervisor program, SD: Supervisor data

* Vector numbers 12, 13, 14, 16 through 23 and 48 through 63 are reserved for future enhancements by Hitachi. No user peripheral devices should be assigned these numbers.

The third step is to save the current processor status, except for the reset exception. The current program counter value and the saved copy of the status register are stacked using the supervisor stack pointer. The program counter value stacked usually points to the next unexecuted instruction, however for bus error and address error, the value stacked for the program counter is unpredictable, and may be incremented from the address of the instruction which caused the error. Additional information defining the current context is stacked for the bus error and address error exceptions.

The last step is the same for all exceptions. The new program counter value is fetched from the exception vector. The processor then resumes instruction execution. Then instruction at the address given in the exception vector is fetched, and normal instruction decoding and execution is started.

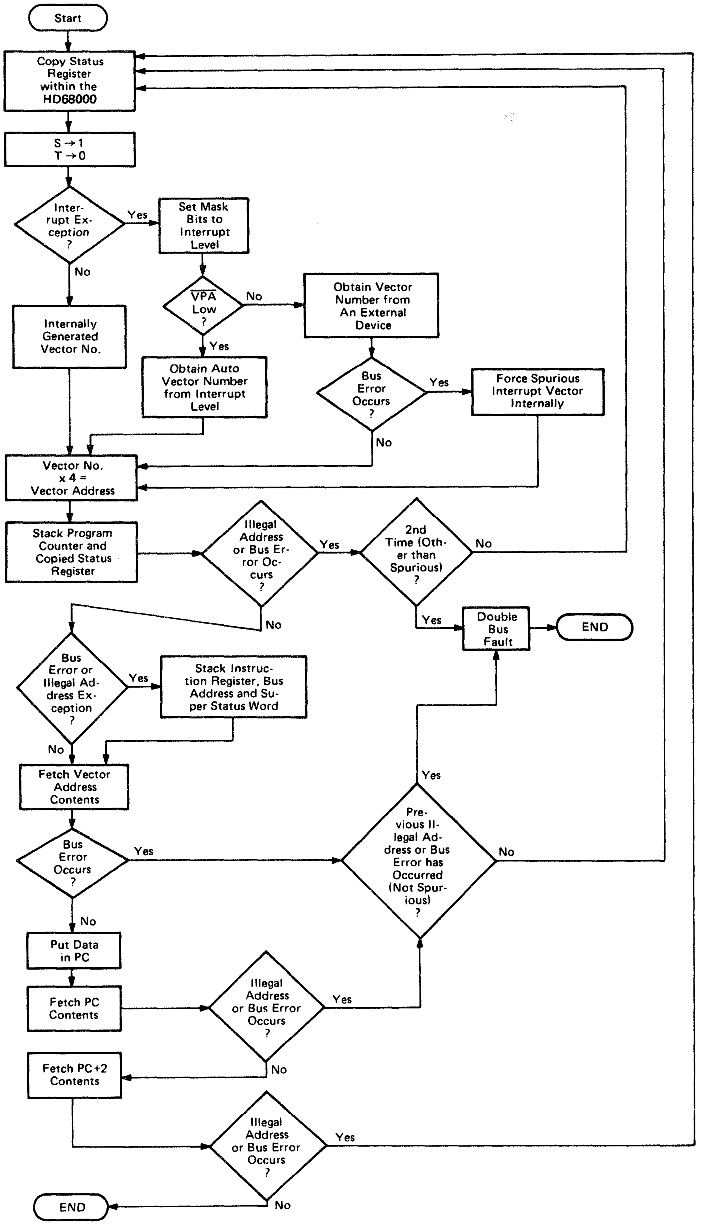

Start
Copy Status Register within the HD68000
S→1
T→0
Interrupt Exception ?
Yes
No
Set Mask Bits to Interrupt Level
VPA Low ?
No
Yes
Obtain Vector Number from An External Device
Internally Generated Vector No.
Obtain Auto Vector Number from Interrupt Level
Bus Error Occurs ?
Yes
No
Force Spurious Interrupt Vector Internally
Vector No. x 4 = Vector Address
Stack Program Counter and Copied Status Register
Illegal Address or Bus Error Occurs ?
Yes
No
2nd Time (Other than Spurious) ?
No
Yes
Double Bus Fault
END
Bus Error or Illegal Address Exception ?
Yes
No
Stack Instruction Register, Bus Address and Super Status Word
Fetch Vector Address Contents
Bus Error Occurs ?
Yes
No
Previous Illegal Address or Bus Error has Occurred (Not Spurious) ?
Yes
No
Put Data in PC
Fetch PC Contents
Illegal Address or Bus Error Occurs ?
Yes
No
Fetch PC+2 Contents
Illegal Address or Bus Error Occurs ?
Yes
No
END

Figure 35 Exception Processing Sequence (Not Reset)

MULTIPLE EXCEPTIONS

These paragraphs describe the processing which occurs when multiple exceptions arise simultaneously. Exceptions can be grouped according to their occurrence and priority. The Group 0 exceptions are reset, bus error, and address error. These exceptions cause the instruction currently being executed to be aborted, and the exeception processing to commence within two clock cycles. The Group 1 exceptions are trace and interrupt, as well as the privilege violations and illegal instructions. These exceptions allow the current instruction to execute to completion, but preempt the execution of the next instruction by forcing exception processing to occur (privilege violations and illegal instructions are detected when they are the next instruction to be executed). The Group 2 exceptions occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, and zero divide exceptions are in this group. For these exceptions, the normal execution of an instruction may lead to exception processing.

Group 0 exceptions have highest priority, while Group 2 exceptions have lowest priority. Within Group 0, reset has highest priority, followed by address error and then bus error. Within Group 1, trace has priority over external interrupts, which in turn takes priority over illegal instruction and privilege violation. Since only one instruction can be executed at a time, there is no priority relation within Group 2.

The priority relation between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. Therefore, if a bus error occurs during a TRAP instruction, the bus error takes precedence, and the TRAP instruction processing is aborted. In another example, if an interrupt request occurs during the execution of an instruction while the T-bit is asserted, the trace exception has priority, and is processed first. Before instruction processing resumes, however, the interrupt exception is also processed, and instruction processing commences finally in the interrupt handler routine. A summary of exception grouping and priority is given in Table 8.

Table 8 Exception Grouping and Priority

Group	Exception	Processing
0	Reset Address Error Bus Error	Exception processing begins within two clock cycles.
1	Trace Interrupt Illegal Privilege	Exception processing begins before the next instruction
2	TRAP, TRAPV CHK, Zero Divide	Exception processing is started by normal instruction execution

RECOGNITION TIMES OF EXCEPTIONS, HALT, AND BUS ARBITRATION

- END OF A CLOCK CYCLE
  - RESET
- END OF A BUS CYCLE
  - ADDRESS ERROR
  - BUS ERROR
  - HALT
  - BUS ARBITRATION
- END OF AN INSTRUCTION CYCLE
  - TRACE EXCEPTION
  - INTERRUPT EXCEPTIONS
  - ILLEGAL INSTRUCTION
  - UNIMPLEMENTED INSTRUCTION
  - PRIVILEGE VIOLATION
- WITHIN AN INSTRUCTION CYCLE
  - TRAP, TRAPV
  - CHK
  - ZERO DIVIDE

• EXCEPTION PROCESSING DETAILED DISCUSSION

Exceptions have a number of sources, and each exception has processing which is peculiar to it. The following paragraphs detail the sources of exceptions, how each arises, and how each is processed.

RESET

The reset input provides the highest exception level. The processing of the reset signal is designed for system initiation, and recovery from catastrophic failure. Any processing in progress at the time of the reset is aborted and cannot be recovered. The processor is forced into the supervisor state, and the trace state is forced off. The processor interrupt priority mask is set at level seven. The vector number is internally generated to reference the reset exception vector at location 0 in the supervisor program space. Because no assumptions can be made about the validity of register contents, in particular the supervisor stack pointer, neither the program counter nor the status register is saved. The address contained in the first two words of the reset exception vector is fetched as the initial supervisor stack pointer, and the address in the last two words of the reset exception vector is fetched as the initial program counter. Finally, instruction execution is started at the address in the program counter. The power-up/restart code should be pointed to by the initial program counter.

The RESET instruction does not cause loading of the reset vector, but does assert the reset line to reset external devices. This allows the software to reset the system to a known state and then continue processing with the next instruction.

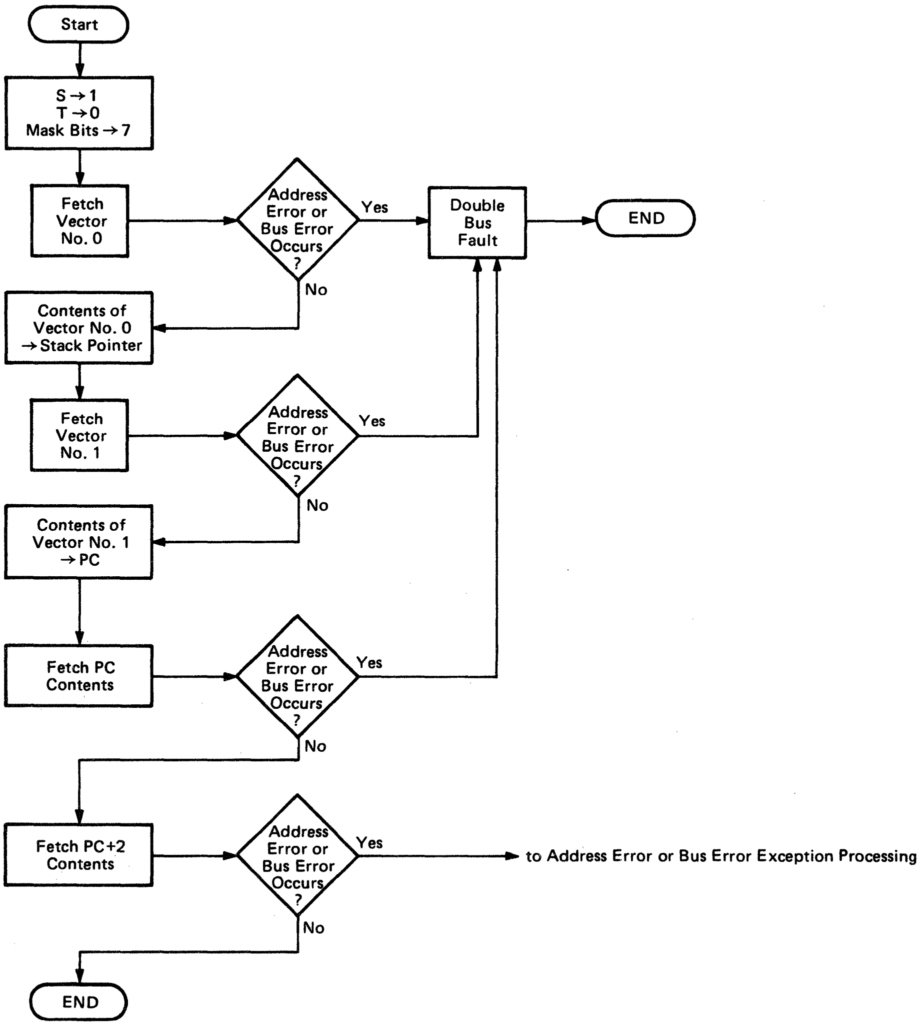

Start
S→1
T→0
Mask Bits→7
Fetch Vector No. 0
Address Error or Bus Error Occurs ?
Yes
No
Double Bus Fault
END
Contents of Vector No. 0 →Stack Pointer
Fetch Vector No. 1
Address Error or Bus Error Occurs ?
Yes
No
Contents of Vector No. 1 →PC
Fetch PC Contents
Address Error or Bus Error Occurs ?
Yes
No
Fetch PC+2 Contents
Address Error or Bus Error Occurs ?
Yes
No
to Address Error or Bus Error Exception Processing
END

Figure 36 Reset Exception Processing

INTERRUPTS

Seven levels of interrupt priorities are provided. Devices may be chained externally within interrupt priority levels, allowing an unlimited number of peripheral devices to interrupt the processor. Interrupt priority levels are numbered from one to seven, level seven being the highest priority. The status register contains a three-bit mask which indicates the current processor priority, and interrupts are inhibited for all priority levels less than or equal to the current processor priority.

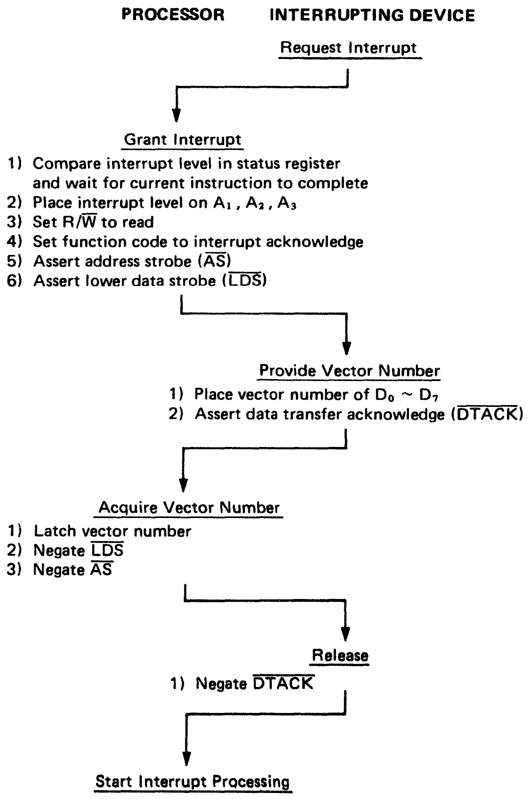
An interrupt request is made to the processor by encoding the interrupt request level on the interrupt request lines; a zero indicates no interrupt request. Interrupt requests arriving at the processor do not force immediate exception processing, but are made pending. Pending interrupts are detected between instruction executions. If the priority of the pending interrupt is lower than or equal to the current processor priority, execution continues with the next instruction and the interrupt exception processing is postponed. (The recognition of level seven is slightly different, as explained in a following paragraph.)

If the priority of the pending interrupt is greater than the current processor priority, the exception processing sequence is started. First a copy of the status register is saved, and the privilege state is set to supervisor, tracing is suppressed, and the processor priority level is set to the level of the interrupt being acknowledged. The processor fetches the vector number from the interrupting device, classifying the reference as an interrupt acknowledge and displaying the level number of the interrupt being acknowledged on the address bus. If external logic requests an automatic vectoring, the processor internally generates a vector number which is determined by the interrupt level number. If external logic indicates a bus error, the interrupt is taken to be spurious, and the generated vector number references the spurious interrupt vector. The processor then proceeds with the usual exception processing, saving the program counter and status register on the supervisor stack. The saved value of the program counter is the address of the instruction which would have been executed had the interrupt not been present. The content of the interrupt vector whose vector number was previously obtained is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine. A flow chart for the interrupt acknowledge sequence is given in Figure 37, a timing diagram is given in Figure 38, and the interrupt exception timing sequence is shown in Figure 39.

Table 9 Internal Interrupt Level

Level	I2	I1	I0	Interrupt
7	1	1	1	Non-Maskable Interrupt
6	1	1	0	Maskable Interrupt
5	1	0	1	
4	1	0	0	
3	0	1	1	
2	0	1	0	
1	0	0	1	
0	0	0	0	No Interrupt

(NOTE) The internal interrupt mask level (I2, I1, I0) are inverted to the logic level applied to the pins ($\overline{IPL_2}$, $\overline{IPL_1}$, $\overline{IPL_0}$).



PROCESSOR
INTERRUPTING DEVICE
Request Interrupt
Grant Interrupt
1) Compare interrupt level in status register and wait for current instruction to complete
2) Place interrupt level on A1, A2, A3
3) Set R/W̄ to read
4) Set function code to interrupt acknowledge
5) Assert address strobe (AS̄)
6) Assert lower data strobe (LDS̄)
Provide Vector Number
1) Place vector number of D0 ~ D7
2) Assert data transfer acknowledge (DTACK̄)
Acquire Vector Number
1) Latch vector number
2) Negate LDS̄
3) Negate AS̄
Release
1) Negate DTACK̄
Start Interrupt Processing

Figure 37 Interrupt Acknowledge Sequence Flow Chart

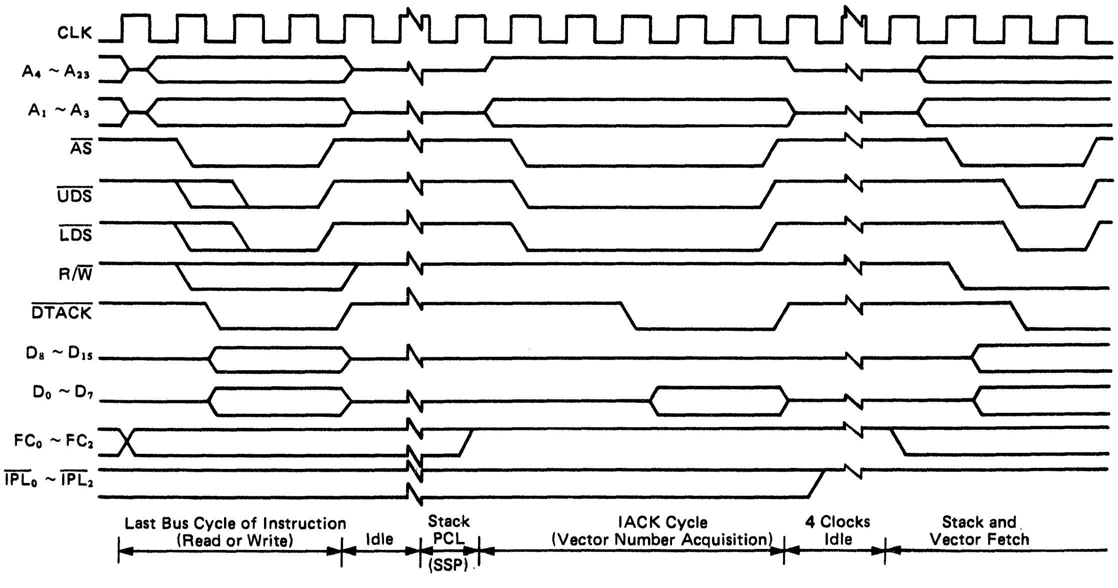


Figure 38 Interrupt Acknowledge Sequence Timing Diagram

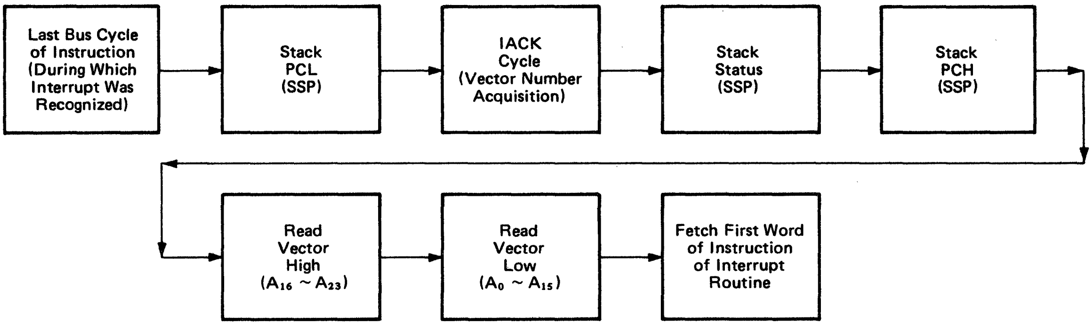

Last Bus Cycle of Instruction (During Which Interrupt Was Recognized)
Stack PCL (SSP)
IACK Cycle (Vector Number Acquisition)
Stack Status (SSP)
Stack PCH (SSP)
Read Vector High ($A_{16} \sim A_{23}$)
Read Vector Low ($A_0 \sim A_{15}$)
Fetch First Word of Instruction of Interrupt Routine

Figure 39 Interrupt Exception Timing Sequence

Priority level seven is a special case. Level seven interrupts cannot be inhibited by the interrupt priority mask, thus providing a "non-maskable interrupt" capability. An interrupt is generated each time the interrupt request level changes from some lower level to level seven. Note that a level seven interrupt may still be caused by the level comparison if the request level is a seven and the processor priority is set to a lower level by an instruction.

UNINITIALIZED INTERRUPT

An interrupting device asserts $\overline{\text{VPA}}$ or provides an interrupt vector during an interrupt acknowledge cycle to the HD68000. If the vector register has not been initialized, the responding HMCS68000 Family peripheral will provide vector 15, the unitialized interrupt vector. This provides a uniform way to recover from a programming error.

SPURIOUS INTERRUPT

If during the interrupt acknowledge cycle no device responds by asserting $\overline{\text{DTACK}}$ or $\overline{\text{VPA}}$, the bus error line should be asserted to terminate the vector acquisition. The processor separates the processing of this error from bus error by fetching the spurious interrupt vector instead of the bus error vector. The processor then proceeds with the usual exception processing.

INSTRUCTION TRAPS

Traps are exceptions caused by instructions. They arise either from processor recognition of abnormal conditions during instruction execution, or from use of instructions whose normal behavior is trapping.

Some instructions are used specifically to generate traps. The TRAP instruction always forces an exception, and is useful for implementing system calls for user programs. The TRAPV and CHK instructions force an exception if the user program detects a runtime error, which may be an arithmetic overflow or a subscript out of bounds.

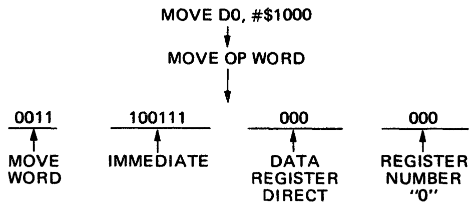
The signed divide (DIVS) and unsigned divide (DIVU) instructions will force an exception if a division operation is attempted with a divisor of zero.

ILLEGAL AND UNIMPLEMENTED INSTRUCTIONS

Illegal instruction is the term used to refer to any of the word bit patterns which are not the bit pattern of the first word of a legal instruction. During instruction execution, if such an instruction is fetched, an illegal instruction exception occurs.

Word patterns with bits 15 through 12 equaling 1010 or 1111 are distinguished as unimplemented instructions and separate exception vectors are given to these patterns to permit efficient emulation. This facility allows the operating system to detect program errors, or to emulate unimplemented instructions in software.

ILLEGAL INSTRUCTION EXAMPLE



PRIVILEGE VIOLATIONS

In order to provide system security, various instructions are privileged. An attempt to execute one of the privileged instructions while in the user state will cause an exception. The privileged instruction are:

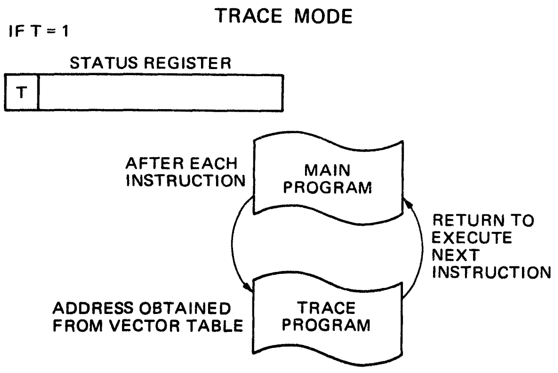
STOP	AND (word) Immediate to SR
RESET	EOR (word) Immediate to SR
RTE	OR (word) Immediate to SR
MOVE to SR	MOVE USP

TRACING

To aid in program development, the HD68000 includes a facility to allow instruction by instruction tracing. In the trace state, after each instruction is executed an exceptions is forced, allowing a debugging program to monitor the execution of the program under test.

The trace facility uses the T-bit in the supervisor portion of the status register. If the T-bit is negated (off), tracing is disabled, and instruction execution proceeds from instruction to instruction as normal. If the T-bit is asserted (on) at the beginning of the execution of an instruction, a trace exception will be generated after the execution of that instruction is completed. If the instruction is not executed. either because an interrupt is taken, or the instruction is illegal or privileged, the trace exception does not occur. The trace exception also does not occur if the instruction is aborted by a reset, bus error, or address error exception. If the instruction is indeed executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception. If, during the execution of the instruction, an exception is forced by that instruction, the forced exception is processed before the trace exception.

As an extreme illustration of the above rules, consider the arrival of an interrupt during the execution of a TRAP instruction while tracing is enabled. First the trap exception is processed, then the trace exception, and finally the interrupt exception. Instruction execution resumes in the interrupt handler routine.



1. If, upon completion of an instruction, T = 1, go to trace exception processing.
2. Execute trace exception sequence.
3. Execute trace service routine.
4. At the end of the service routine, execute return from exception (RTE).

BUS ERROR

Bus error exceptions occur when the external logic requests that a bus error be processed by an exception. The current bus cycle which the processor is making is then aborted. Whether the processor was doing instruction or exception processing, that processing is terminated, and the processor immediately begins exception processing.

Exception processing for bus error follows the usual sequence of steps. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is generated to refer to the bus error vector. Since the processor was not between instructions when the bus error exception request was made, the context of the processor is more detailed. To save more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are of course saved. The value saved for the program counter is advanced by some amount, two to ten bytes beyond the address of the first word of the instruction which made the reference causing the bus error. If the bus error occurred during the fetch of the next instruction, the saved program counter has a value in the vicinity of the current instruction, even if the current instruction is a branch, a jump, or a return instruction. Besides the usual information, the processor saves its internal copy of the first word of the instruction being processed, and the address which was being accessed by the aborted bus cycle. Specific information about the access is also saved: whether it was a read or a write, whether the processor was processing an instruction or not, and the classification displayed on the function code outputs when the bus error occurred. The processor is processing an instruction if it is in the normal state or processing a Group 2 exception; the processor is not processing an instruction if it is processing a Group 0 or a Group 1 exception. Figure 40 illustrates how this information is organized on the supervisor stack. Although this information is not sufficient in general to effect full recovery from the bus error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address contained in the vector. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

If a bus error occurs during the exception processing for a bus error, address error, or reset, the processor is halted, and all processing cases. This simplifies the detection of catastrophic system failure, since the processor removes itself from the system rather than destroy all memory contents. Only the RESET pin can restart a halted processor.

ADDRESS ERROR

Address error exceptions occur when the processor attempts to access a word or a long word operand or an instruction at an odd address. The effect is much like an internally generated bus error, so that the bus cycle is aborted, and the processor ceases whatever processing it is currently doing and begins exception processing. After exception processing commences, the sequence is the same as that for bus error including the information that is stacked, except that the vector number refers to the address error vector instead. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the processor is halted. As shown in Figure 42, an address error will execute a short bus cycle followed by exception processing.

■ INTERFACE WITH HMCS6800 PERIPHERALS

Hitachi's extensive line of HMCS6800 peripherals are directly compatible with the HD68000. Some of these devices that are particularly useful are:

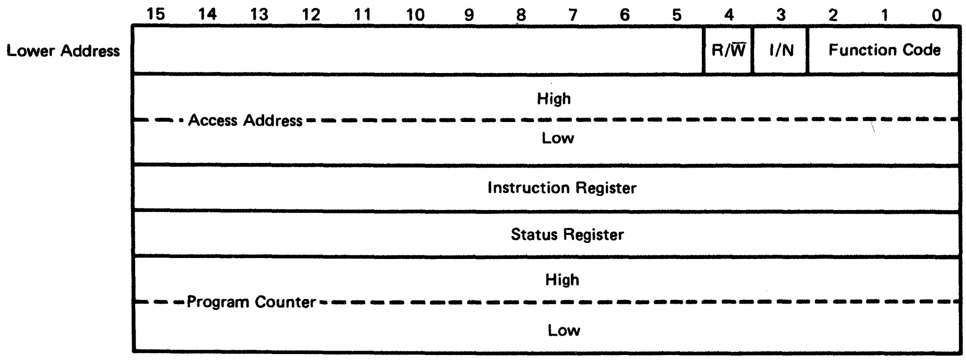
HD6821	Peripheral Interface Adapter
HD6843	Floppy Disk Controller
HD6845S	CRT Controller
HD46508	Data Acquisition Unit
HD6850	Asynchronous Communication Interface Adapter
HD6852	Synchronous Serial Data Adapter

To interface the synchronous HMCS6800 peripherals with the asynchronous HD68000, the processor modifies its bus cycle to meet the HMCS6800 cycle requirements whenever an HMCS6800 device address is detected. This is possible since both processors use memory mapped I/O. Figure 44 is a flow chart of the interface operation between the processor and HMCS6800 devices.

● DATA TRANSFER OPERATION

Three signal on the processor provide the HMCS6800 interface. They are: enable (E), valid memory address ($\overline{VMA}$), and valid peripheral address ($\overline{VPA}$). Enable corresponds to the E or ϕ_2 signal in existing HMCS6800 systems. The bus frequency is one tenth of the incoming HD68000 clock frequency. The timing of E allows 1 MHz peripherals to be used with an 8 MHz HD68000. Enable has a 60/40 duty cycle; that is, it is low for six input clocks and high for four input clocks. This duty cycle allows the processor to do successive VPA accesses on successive E pulses.

HMCS6800 cycle timing is given in Figure 45 and 46. At



R/$\overline{W}$ (read/write): write = 0, read = 1. I/N (instruction/not): instruction = 0, not = 1

Figure 40 Supervisor Stack Order (Group 0)

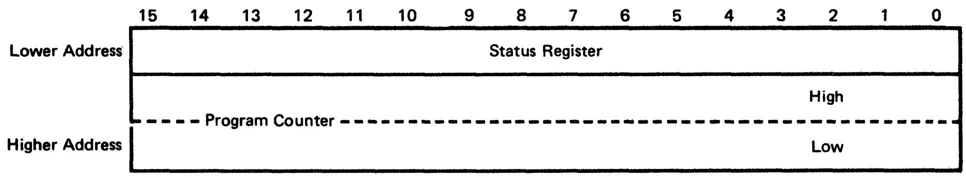


Figure 41 Supervisor Stack Order (Group 1, 2)

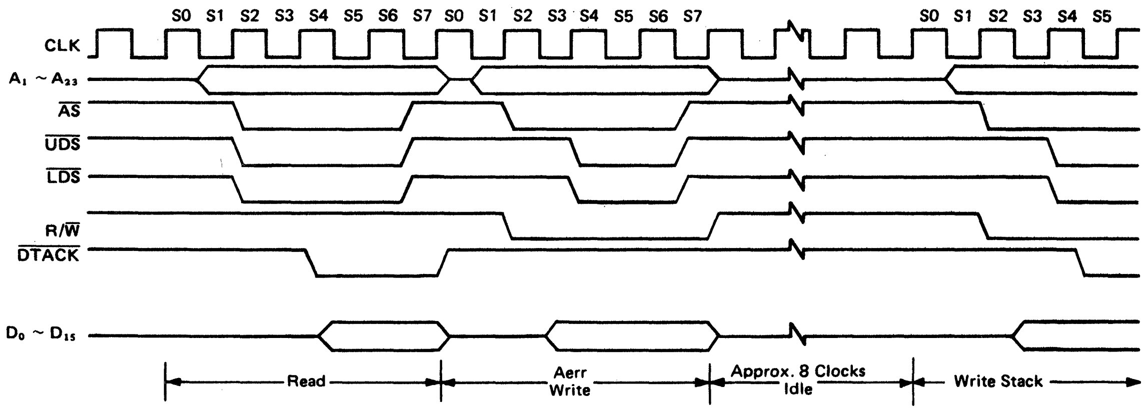


Figure 42 Address Error Timing

state zero (S0) in the cycle, the address bus is in the high-impedance state. A function code is asserted on the function code output lines. One-half clock later, in state 1 the address bus is released from the high-impedance state.

During state 2, the address strobe ($\overline{AS}$) is asserted to indicate that there is a valid address on the address bus. If the bus cycle is a read cycle, the upper and/or lower data strobes are also asserted in state 2. If the bus cycle is a write cycle,

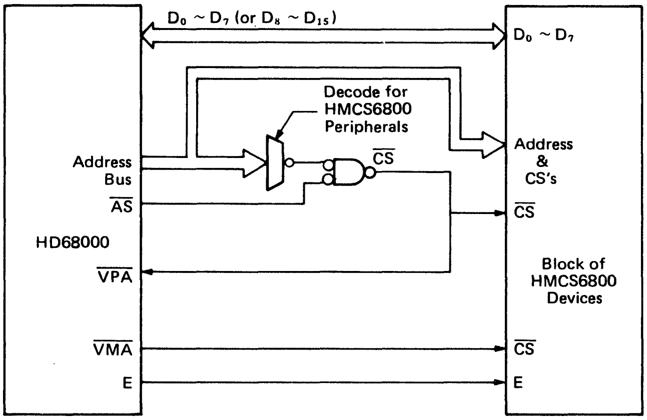


Figure 43 Connection of HMCS6800 Peripherals

the read/write (R/$\overline{W}$) signal is switched to low (write) during state 2. One half clock later, in state 3, the write data is placed on the data bus, and in state 4 the data strobes are issued to indicate valid data on the data bus. The processor now inserts wait states until it recognizes the assertion of $\overline{VPA}$.

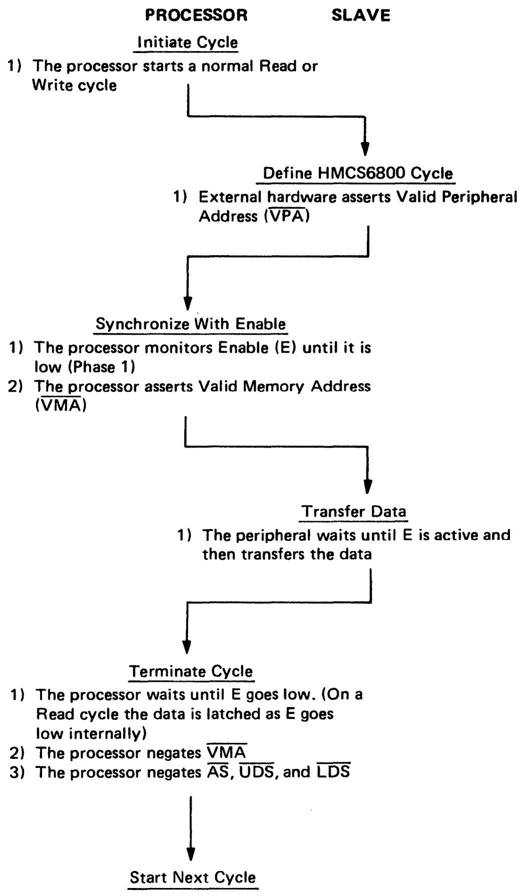
The $\overline{VPA}$ input signals the processor that the address on the bus is the address of an HMCS6800 device (or an area reserved for HMCS6800 devices) and that the bus should conform to the ϕ_2 transfer characteristics of the HMCS6800 bus. Valid peripheral address is derived by decoding the address bus, conditioned by address strobe.

After the recognition of $\overline{VPA}$, the processor assures that the Enable (E) is low, by waiting if necessary, and subsequently asserts $\overline{VMA}$. Valid memory address is then used as part of the chip select equation of the peripheral. This ensures that the HMCS6800 peripherals are selected and deselected at the correct time. The peripheral now runs its cycle during the high portion of the E signal. Figures 45 and 46 depict the best and worst case HMCS6800 cycle timing. This cycle length is dependent strictly upon when $\overline{VPA}$ is asserted in relationship to the E clock.

During a read cycle, the processor latches the peripheral data in state 6. For all cycles, the processor negates the address and data strobes one half clock cycle later in state 7, and the Enable signal goes low at this time. Another half clock later, the address bus is put in the high-impedance state. During a write cycle, the data bus is put in the high-impedance state and the read/write signal is switched high. The peripheral logic must remove $\overline{VPA}$ within one clock after address strobe is negated.

Figure 47 shows the timing required by HMCS6800 peripherals, the timing specified for HDCS6800, and the corresponding timing for the HD68000. Two example systems with HMCS6800 peripherals are showin in Figures 48 and 49. The system in Figure 48 reserves the upper eight megabytes of memory for HMCS6800 peripherals. The system in Figure 49 is more efficient with memory and easily expandable, but more complex.

$\overline{DTACK}$ should not be asserted while $\overline{VPA}$ is asserted. Notice that the HD68000 $\overline{VMA}$ is active low, contrasted with the active high HMCS6800 VMA. This allows the processor to put its buses in the high-impedance state on DMA requests without inadvertently selecting peripherals.



Initiate Cycle
1) The processor starts a normal Read or Write cycle

Define HMCS6800 Cycle
1) External hardware asserts Valid Peripheral Address ($\overline{VPA}$)

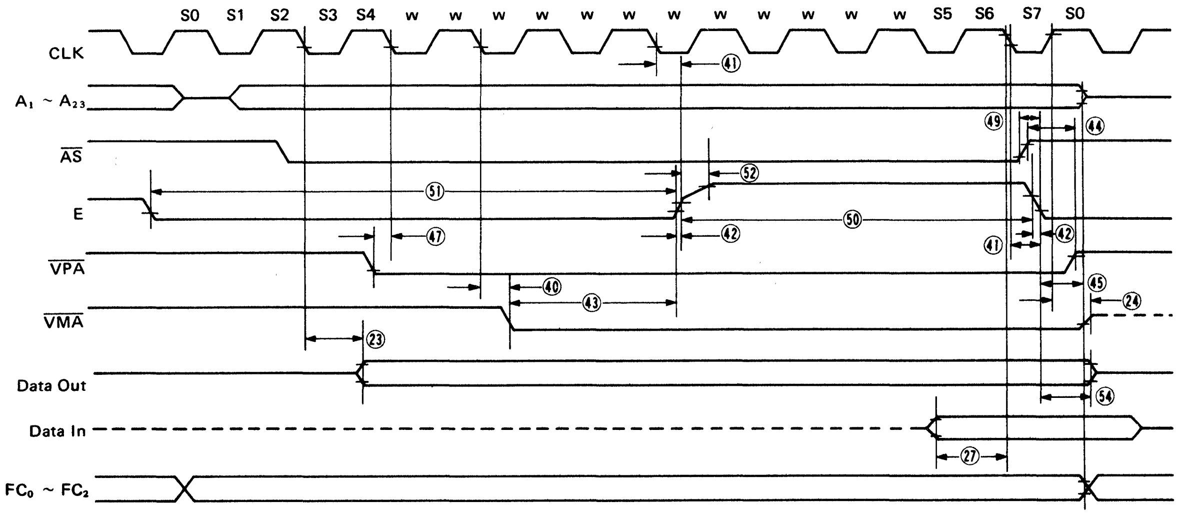
Synchronize With Enable
1) The processor monitors Enable (E) until it is low (Phase 1)
2) The processor asserts Valid Memory Address ($\overline{VMA}$)

Transfer Data
1) The peripheral waits until E is active and then transfers the data

Terminate Cycle
1) The processor waits until E goes low. (On a Read cycle the data is latched as E goes low internally)
2) The processor negates $\overline{VMA}$
3) The processor negates $\overline{AS}$, $\overline{UDS}$, and $\overline{LDS}$

Start Next Cycle

Figure 44 HMCS6800 Interface Flow Chart



(NOTE) This figure represents the best case HMCS6800 timing where $\overline{\text{VPA}}$ falls before the third system clock cycle after the falling edge of E.

Figure 45 HMCS6800 Timing – Best Case

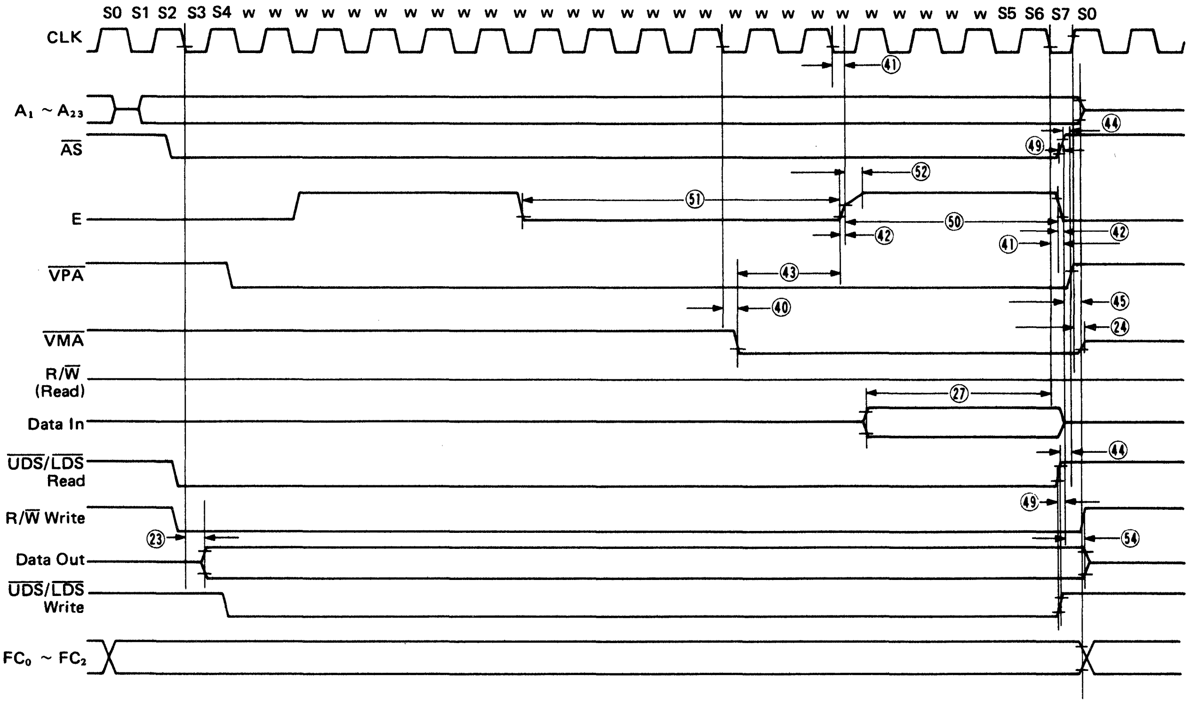
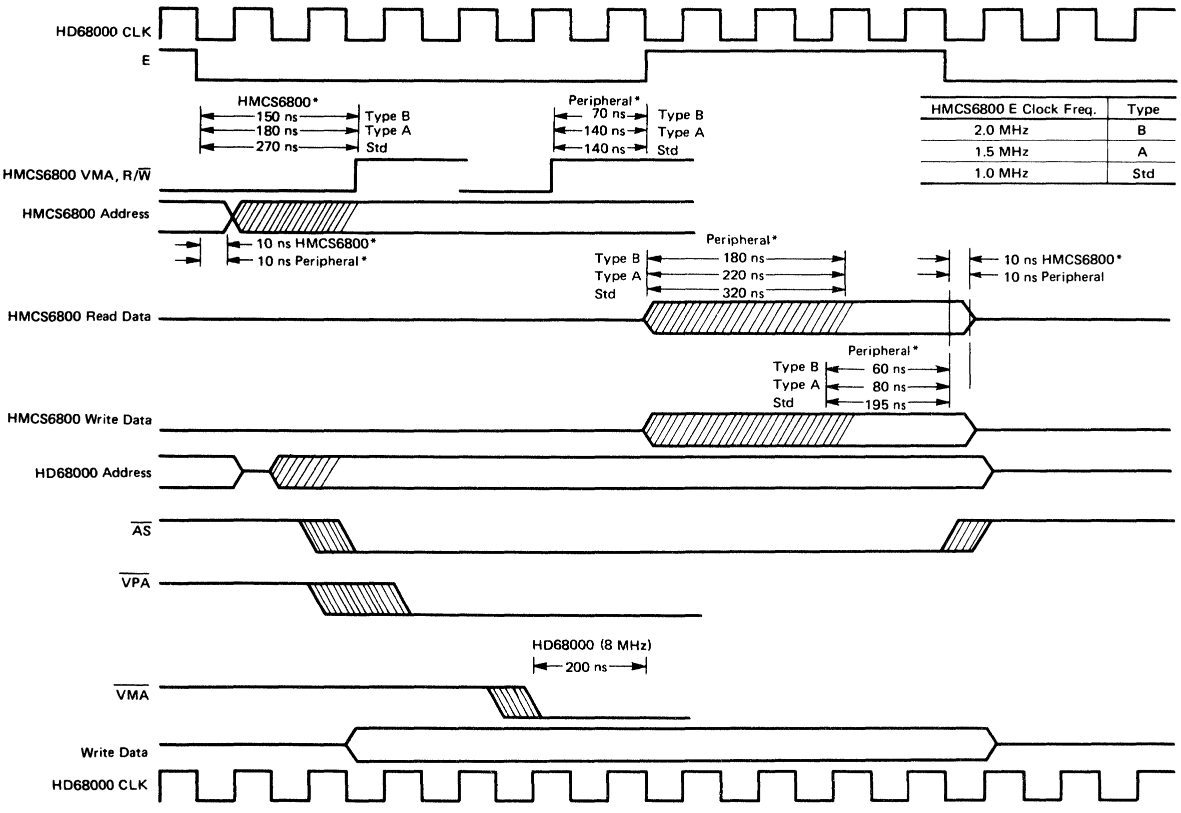


Figure 46 HMCS6800 Timing – Worst Case



HMCS6800 E Clock Freq.	Type
2.0 MHz	B
1.5 MHz	A
1.0 MHz	Std

* Times are expressed for different device clock frequencies.

Figure 47 HD68000 to HMCS6800 Peripheral Timing Diagram

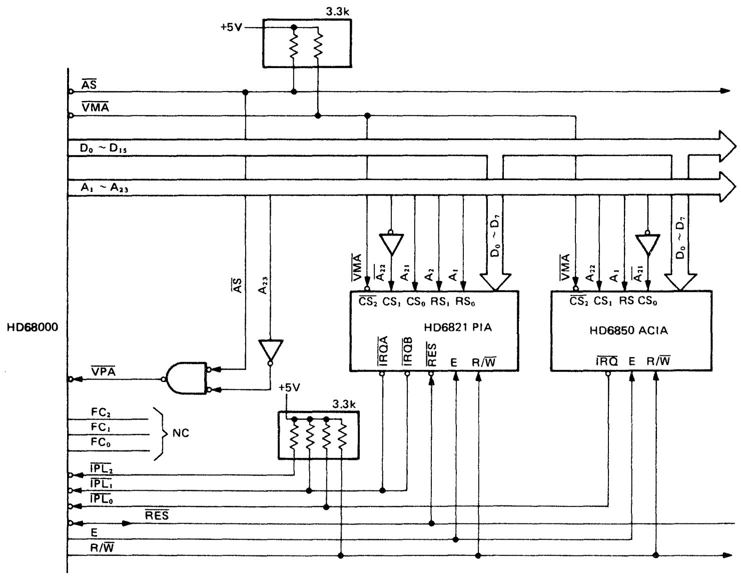


Figure 48 HMCS6800 Interface — Example 1

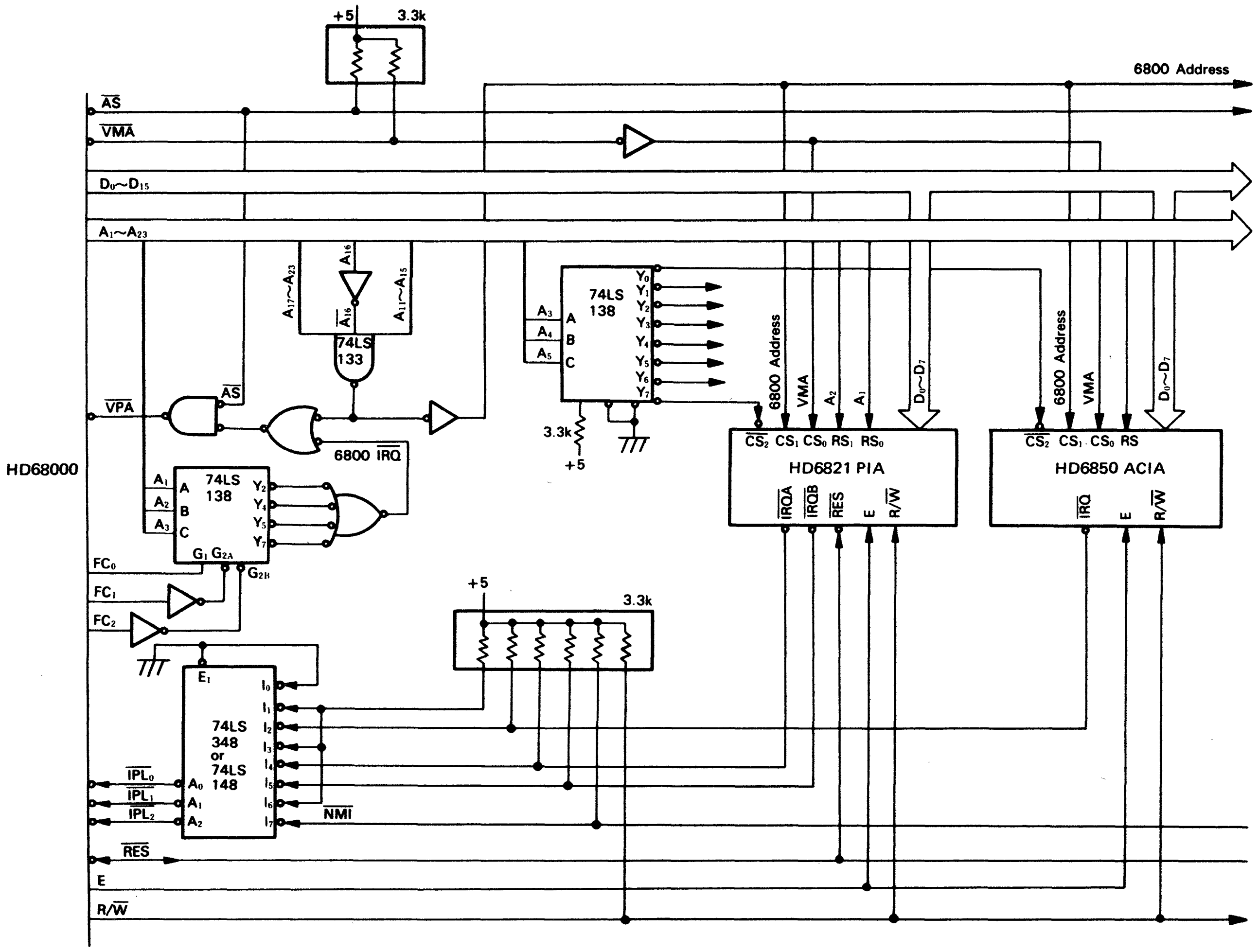


Figure 49 HMCS6800 Interface — Example 2

• INTERRUPT OPERATION

During an interrupt acknowledge cycle while the processor is fetching the vector, if $\overline{VPA}$ is asserted, the HD68000 will assert $\overline{VMA}$ and complete a normal HMCS6800 read cycle as shown in Figure 50. The processor will then use an internally generated vector that is a function of the interrupt being serviced. This process is known as autovectoring. The seven autovectors are vector numbers 25 through 31 (decimal).

This operates in the same fashion (but is not restricted to) the HMCS6800 interrupt sequence. The basic difference is that there are six normal interrupt vectors and one NMI type vector. As with both the HMCS6800 and the HD68000's normal vectored interrupt, the interrupt service routine can be located anywhere in the address space. This is due to the fact that while the vector numbers are fixed, the contents of the vector table entries are assigned by the user.

Since $\overline{VMA}$ is asserted during autovectoring, the HMCS6800 peripheral address decoding should prevent unintended accesses.

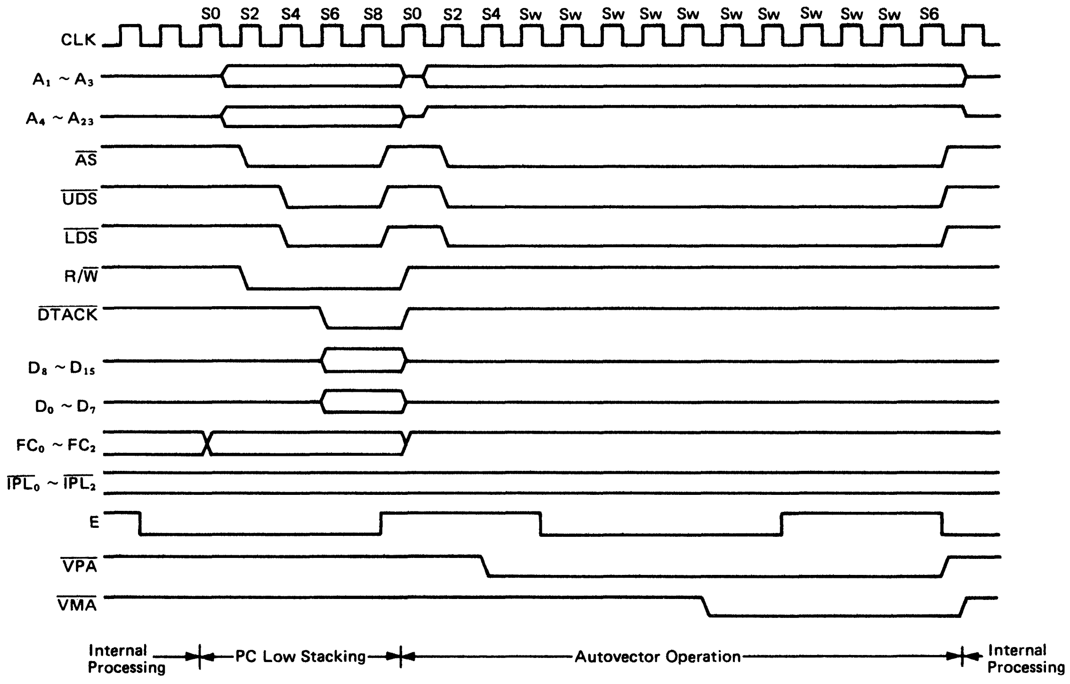


Figure 50 Autovector Operation Timing Diagram

■ DATA TYPES AND ADDRESSING MODES

Five basic data types are supported. These data types are:

- Bits
- BCD Digits (4-bits)
- Bytes (8-bits)
- Word (16-bits)
- Long Words (32-bits)

In addition, operations on other data types such as memory addresses, status word data, etc., are provided for in the instruction set.

The 14 addressing modes, shown in Table 10, includs six basic types:

- Register Direct
- Register Indirect
- Absolute
- Immediate
- Program Counter Relative
- Implied

Included in the register indirect addressing modes is the capability to do postincrementing, predecrementing, offsetting and indexing. Program counter relative mode can also be modified via indexing and offsetting.

Table 10 Addressing Modes

Mode	Generation
Register Direct Addressing	
Data Register Diredt	EA = Dn
Address Register Direct	EA = An
Absolute Data Addressing	
Absolute Short	EA = (Next Word)
Absolute Long	EA = (Next Two Words)
Program Counter Relative Addressing	
Relative with Offset	EA = (PC) + d_{16}
Relative with Index and Offset	EA = PC) + (Xn) + d_8
Register Indirect Addressing	
Register Indirect	EA = (An)
Postincrement Register Indirect	EA = (AN), An ← An + N
Predecrement Register Indirect	An ← An − N, EA = (An)
Register Indirect with Offset	EA = (An) + d_{16}
Indexed Register Indirect with Offset	EA = (An) + (Xn) + d_8
Immediate Data Addressing	
Immediate	DATA = Next Word(s)
Quick Immediate	Inherent Data
Implied Addressing	
Implied Register	EA = SR, USP, SP, PC

(NOTES)

EA = Effective Address	d_8 = Eight-bit Offset (displacement)
An = Address Register	
Dn = Data Register	d_{16} = Sixteen-bit Offset (displacement)
Xn = Address or Data Register used as Index Register	N = 1 for Byte, 2 for Words and 4 for Long Words
SR = Status Register	
PC = Program Counter	
( ) = Contents of	← = Replaces

■ INSTRUCTION SET OVERVIEW

The HD68000 instruction set is shown in Table 11. Some additional instructions are variations, or subsets, of these and they appear in Table 12. Special emphasis has been given to the instruction set's support of structured high-level languages to facilitate ease of programming. Each instruction, with few exceptions, operates on bytes, words, and long words and most instructions can use any of the 14 addressing modes. Combining instruction types, data types, and addressing modes, over 1000 useful instructions are provided. These instructions include signed and unsigned multiply and divide, "quick" arithmetic operations, BCD arithmetic and expanded operations (through traps).

The following paragraphs contain an overview of the form and structure of the HD68000 instruction set. The instructions form a set of tools that include all the machine functions to perform the following operations:

Data Movement
Integer Arithmetic
Logical
Shift and Rotate
Bit Manipulation
Binary Coded Decimal
Program Control
System Control

The complete range of instruction capabilities combined with the flexible addressing modes described previously provide a very flexible base for program development.

Table 11 Instruction Set

Mnemonic	Description	Mnemonic	Description	Mnemonic	Description
ABCD	Add Decimal with Extend	EOR	Exclusive Or	PEA	Push Effective Address
ADD	Add	EXG	Exchange Registers	RESET	Reset External Devices
AND	Logical And	EXT	Sign Extend	ROL	Rotate Left without Extend
ASL	Arithmetic Shift Left	JMP	Jump	ROR	Rotate Right without Extend
ASR	Arithmetic Shift Right	JSP	Jump to Subroutine	ROXL	Rotate Left with Extend
B_{CC}	Branch Conditionally	LEA	Load Effective Address	ROXR	Rotate Right with Extend
BCHG	Bit Test and Change	LINK	Link Stack	RTE	Return from Exception
BCLR	Bit Test and Clear	LSL	Logical Shift Left	RTR	Return and Restore
BRA	Branch Always	LSR	Logical Shift Right	RTS	Return from Subroutine
BSET	Bit Test and Set	MOVE	Move	SBCD	Subtract Decimal with Extend
BSR	Branch to Subroutine	MOVEM	Move Multiple Registers	S_{CC}	Set Conditional
BTST	Bit Test	MOVEP	Move Peripheral Data	STOP	Stop
CHK	Check Register Against Bounds	MULS	Signed Multiply	SUB	Subtract
CLR	Clear Operand	MULU	Unsigned Multiply	SWAP	Swap Data Register Halves
CMP	Compare	NBCD	Negate Decimal with Extend	TAS	Test and Set Operand
DB_{CC}	Test Condition, Decrement and Branch	NEG	Negate	TRAP	Trap
DIVS	Signed Divide	NOP	No Operation	TRAPV	Trap on Overflow
DIVU	Unsigned Divide	NOT	One's Complement	TST	Test
		OR	Logical Or	UNLK	Unlink

Table 12 Variations of Instruction Types

Instruction Type	Variation	Description	Instruction Type	Variation	Description
ADD	**ADD**	Add	**MOVE**	**MOVE**	Move
	ADDA	Add Address		MOVEA	Move Address
	ADDQ	Add Quick		MOVEQ	Move Quick
	ADDI	Add Immediate		MOVE from SR	Move from Status Register
	ADDX	Add with Extend		MOVE to SR	Move to Status Register
AND	**AND**	Logical And		MOVE to CCR	Move to Condition Codes
	ANDI	And Immediate		MOVE USP	Move User Stack Pointer
CMP	**CMP**	Compare	**NEG**	**NEG**	Negate
	CMPA	Compare Address		NEGX	Negate with Extend
	CMPM	Compare Memory	**OR**	**OR**	Logical Or
	CMPI	Compare Immediate		ORI	Or Immediate
EOR	**EOR**	Exclusive Or	**SUB**	**SUB**	Subtract
	EORI	Exclusive Or Immediate		SUBA	Subtract Address
				SUBI	Subtract Immediate
				SUBQ	Subtract Quick
				SUBX	Subtract with Extend

• ADDRESSING

Instructions for the HD68000 contain two kinds of information: the type of function to be performed, and the location of the operand(s) on which to perform that function. The methods used to locate (address) the operand(s) are explained in the following paragraphs.

Instructions specify an operand location in one of three ways:

Register Specification – the number of the register is given in the register field of the instruction.

Effective Address – use of the different effective address modes.

Implicit Reference – the definition of certain instructions implies the use of specific registers.

• DATA MOVEMENT OPERATIONS

The basic method of data acquisition (transfer and storage) is provided by the move (MOVE) instruction. The move instruction and the effective addressing modes allow both address and data manipulation. Data move instructions allow byte, word, and long word operands to be transferred from memory to memory, memory to register, register to memory, and register to memory, and register to register. Address move instructions allow word and long word operand transfers and ensure that only legal address manipulations are executed. In addition to the general move instruction there are several special data movement instructions: move multiple registers (MOVEM), move peripheral data (MOVEP), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), unlink stack (UNLK), and move quick (MOVEQ). Table 13 is a summary of the data movement operations.

• INTEGER ARITHMETIC OPERATIONS

The arithmetic operations include the four basic operations of add (ADD), subtract (SUB), multiply (MUL), and divide (DIV) as well as arithmetic compare (CMP), clear (CLR), and negate (NEG). The add and subtract instructions are available for both address and data operations, with data operations accepting all operand sizes. Address operations are limited to legal address size operands (16 or 32 bits). Data, address, and memory compare operations are also available. The clear and negate instructions may be used on all sizes of data operands.

The multiply and divide operations are available for signed and unsigned operands using word multiply to produce a long word product, and a long word dividend with word divisor to produce a word quotien with a word remainder.

Multiprecision and mixed size arithmetic can be accomplished using a set of extended instructions. These instructions are: add extended (ADDX), subtract extended (SUBX), sign extend (EXT), and negate binary with extend (NEGX).

A test operand (TST) instruction that will set the condition codes as a result of a compare of the operand with zero is also available. Test and set (TAS) is a synchronization instruction useful in multiprocessor systems. Table 14 is a summary of the integer arithmetic operations.

Table 13 Data Movement Operations

Instruction	Operand Size	Operation
EXG	32	Rx ↔ Ry
LEA	32	EA → An
LINK	–	An → SP@-; SP → An; SP + d → SP
MOVE	8, 16, 32	(EA)s → EAd
MOVEM	16, 32	(EA) → An, Dn An, Dn → EA
MOVEP	16, 32	(EA) → Dn Dn → EA
MOVEQ	8	#xxx → Dn
PEA	32	EA → SP@-
SWAP	32	Dn[31:16] ↔ Dn[15:0]
UNLK	–	An → Sp; SP@+ → An

(NOTES)
s = source
d = destination
[ ] = bit numbers
@- = indirect with predecrement
@+ = indirect with postdecrement

Table 14 Integer Arithmetic Operations

Instruction	Operand Size	Operation
ADD	8, 16, 32	Dn + (EA) → Dn
		(EA + Dn → EA
		(EA) + #xxx → EA
	16, 32	AN + (EA) → An
ADDX	8, 16, 32	Dx + Dy + X → Dx
	16, 32	Ax@ - +Ay@ - + X → Ax@
CLR	8, 16, 32	0 → EA
CMP	8, 16, 32	Dn - (EA)
		(EA) - #xxx
		Ax@+ - Ay@+
	16, 32	An - (EA)
DIVS	32 ÷ 16	Dn/(EA) → Dn
DIVU	32 ÷ 16	Dn/(EA) → Dn
EXT	8 → 16	$(Dn)_8 \rightarrow Dn_{16}$
	16 → 32	$(Dn)_{16} \rightarrow Dn_{32}$
MULS	16*16 → 32	Dn*(EA) → Dn
MULU	16*16 → 32	Dn*(EA) → Dn
NEG	8, 16, 32	0 - (EA) → EA
NEGX	8, 16, 32	0 - (EA) - X - EA
SUB	8, 16, 32	Dn - (EA) → Dn
		(EA) - Dn → EA
		(EA) - #xxx → EA
	16, 32	An - (EA) → An
SUBX	8, 16, 32	Dx - Dy - X → Dx
		Ax@ - - Ay@ - - X → Ax@
TAS	8	(EA) - 0, 1 → EA[7]
TST	8, 16, 32	(EA) - 0

(NOTE) [ ] = bit number

• INSTRUCTION FORMAT

Instructions are from one to five words in length, as shown in Figure 51. The length of the instruction and the operation to be performed is specified by the first word of the instruction which is called the operation word. The remaining words further specify the operands. These words are either immediate operands or extensions to the effective address mode specified in the operation word.

• PROGRAM/DATA REFERENCES

The HD68000 separates memory references into two classes: program references, and data references. Program references, as the name implies, are references to that section of memory that contains the program being executed. Data references refer to that section of memory that contains data. Generally, operand reads are from the data space. All operand writes are to the data space.

• REGISTER SPECIFICATION

The register field within an instruction specifies the register to be used. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used.

• EFFECTIVE ADDRESS

Most instructions specify the location of an operand by using the effective address field in the operation word. For example, Figure 52 shows the general format of the single effective address is composed of two 3-bit fields: the mode field, and the register field. The value in the mode field selects the different address modes. The register field contains the number of a register.

The effective address field may require additional information to fully specify the operand. This additional information, called the effective address extension, is contained in the following word or words and is considered part of the instruction, as shown in Figure 51. The effective address modes are grouped into three categories: register direct, memory addressing, and special.

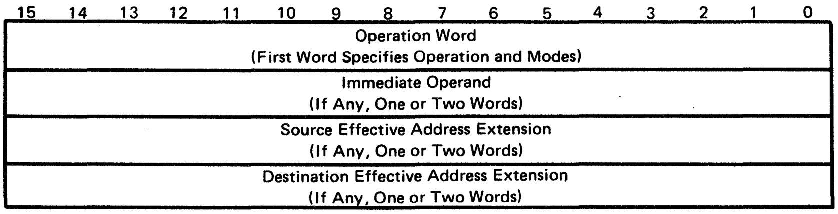

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Operation Word
(First Word Specifies Operation and Modes)
Immediate Operand
(If Any, One or Two Words)
Source Effective Address Extension
(If Any, One or Two Words)
Destination Effective Address Extension
(If Any, One or Two Words)

Figure 51 Instruction Format

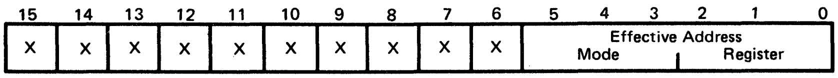


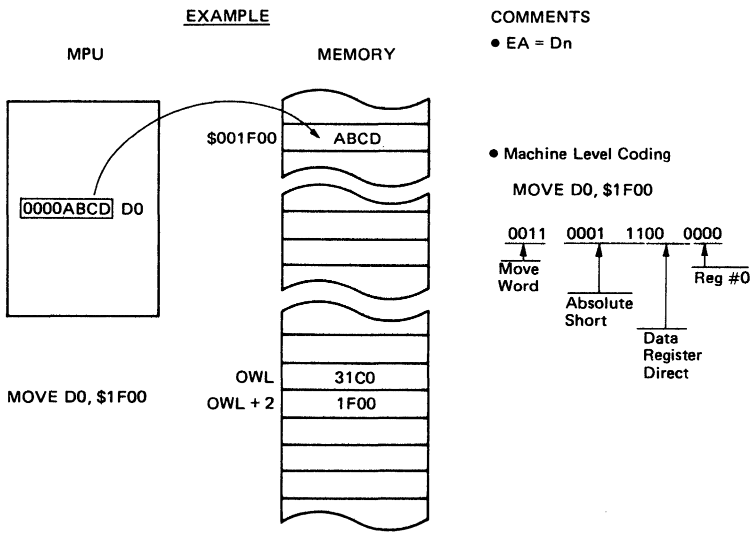
Figure 52 Single-Effective-Address Instruction Operation Word General Format

REGISTER DIRECT MODES

These effective addressing modes specify that the operand is in one of the 16 multifunction registers.

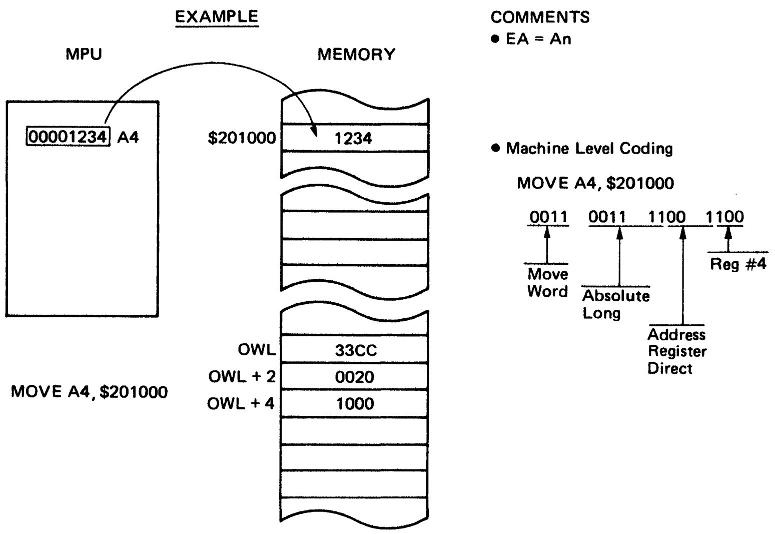
Data Register Direct

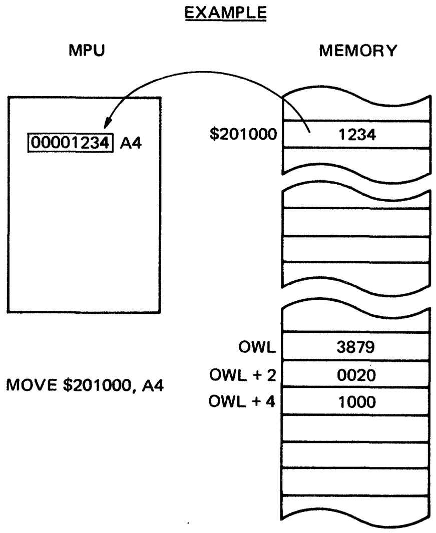
The operand is in the data register specified by the effective address register field.



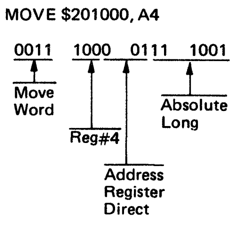
Address Register Direct

The operand is in the address register specified by the effective address register field.





COMMENTS
- EA = An
- Address Register Sign Extended
- Machine Level Coding

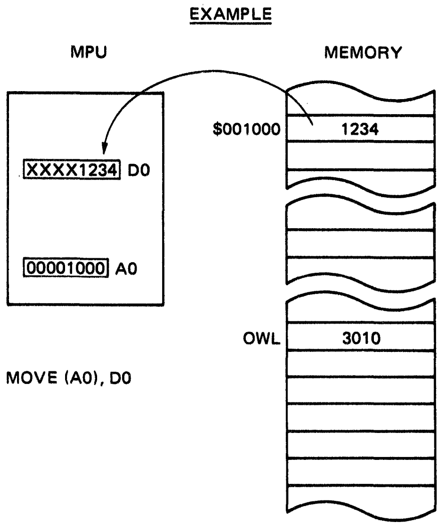


MEMORY ADDRESS MODES

These effective addressing modes specify that the operand is in memory and provide the specific address of the operand.

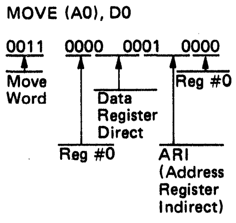
Address Register Indirect

The address of the operand is in the address register specified by the register field. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.



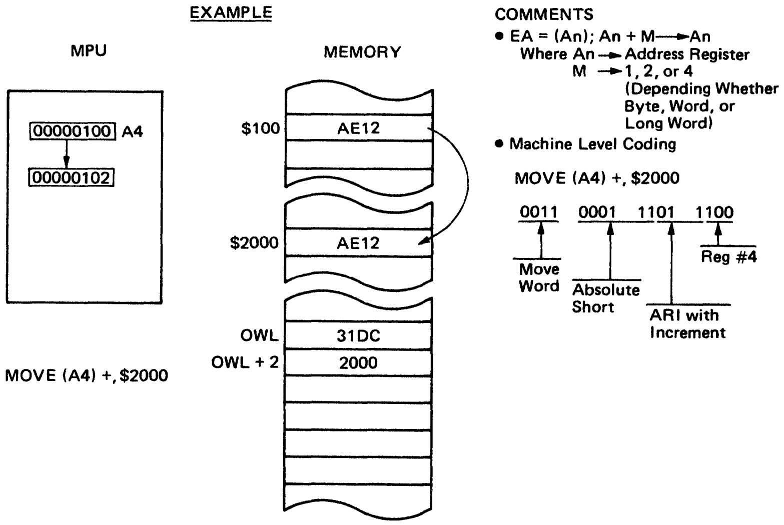
COMMENTS
- EA = (An)

- Machine Level Coding



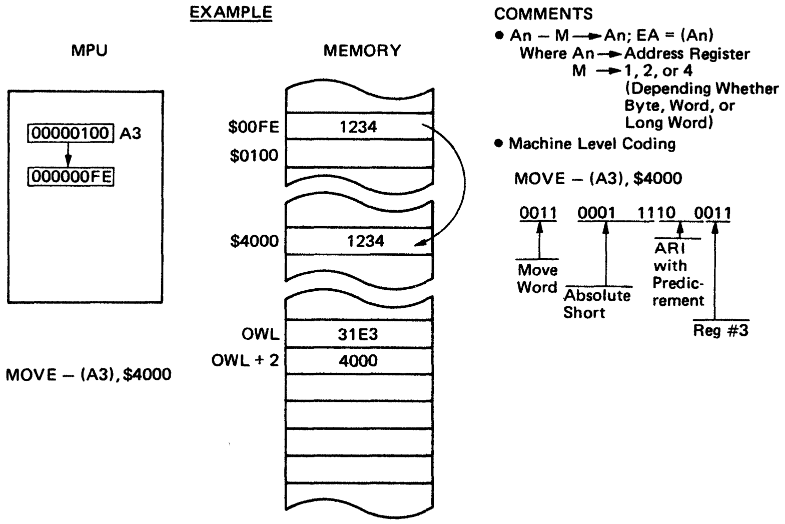
Address Register Indirect With Postincrement

The address of the operand is in the address register specified by the register field. After the operand address is used, it is incremented by one, two, or four depending upon whether the size of the operand is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is incremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.



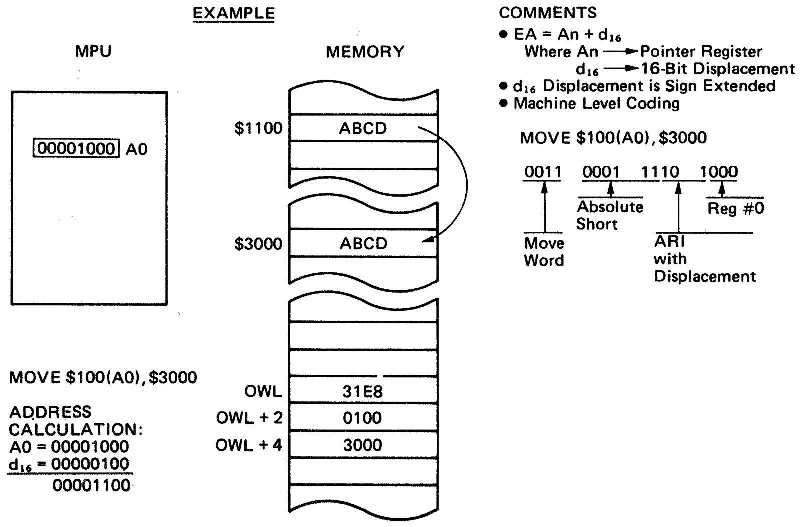
Address Register Indirect With Predecrement

The address of the operand is in the address register specified by the register field. Before the operand address is used, it is decremented by one, two, or four depending upon whether the operand size is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is decremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.



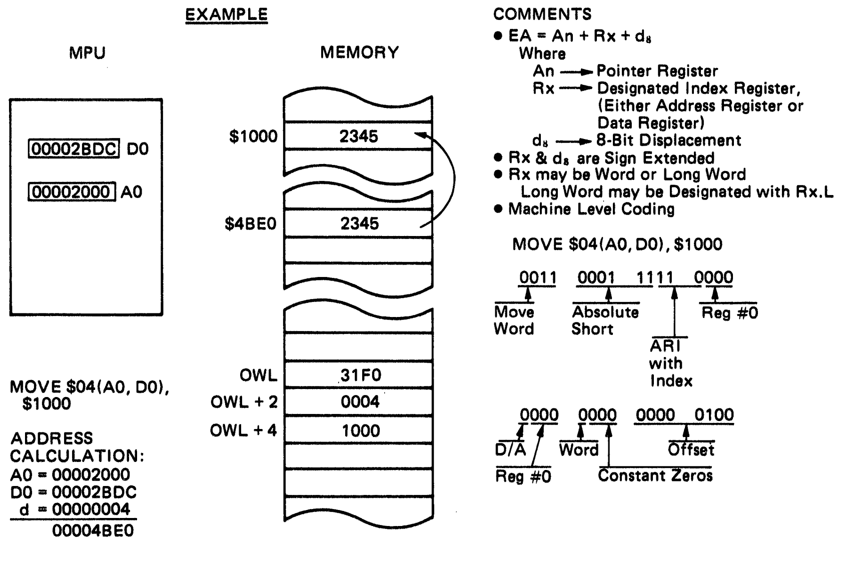
Address Register Indirect With Displacement

This address mode requires one word of extension. The address of the operand is the sum of the address in the address register and the sign-extended 16-bit displacement integer in the extension word. The reference is classified as a data reference with the exception of the jump to subroutine instructions.



Address Register Indirect With Index

This address mode requires one word of extension. The address of the operand is the sum of the address in the address register, the sign-extended displacement integer in the low order eight bits of the extension word, and the contents of the index register. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

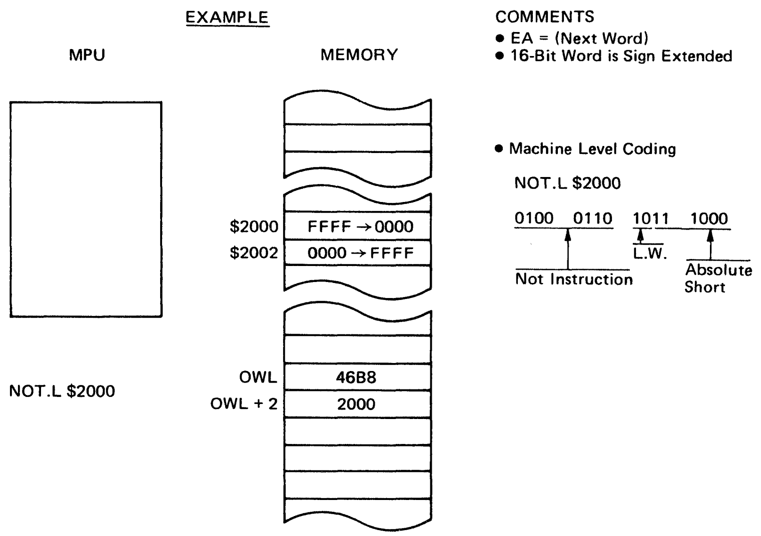


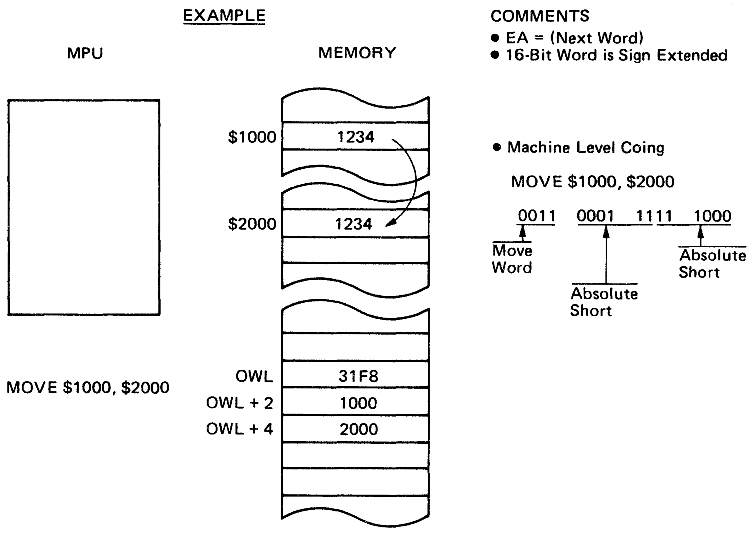
SPECIAL ADDRESS MODE

The special address modes use the effective address register field to specify the special addressing mode instead of a register number.

Absolute Short Address

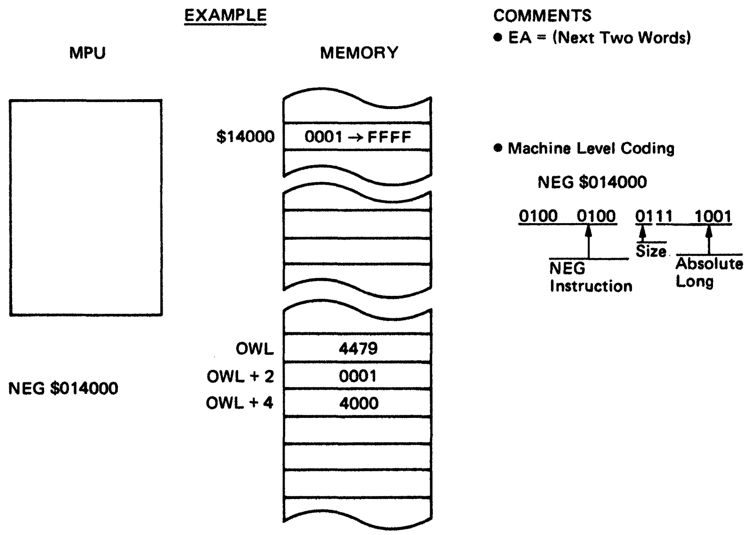
This address mode requires one word of extension. The address of the operand is the extension word. The 16-bit address is sign extended before it is used. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.





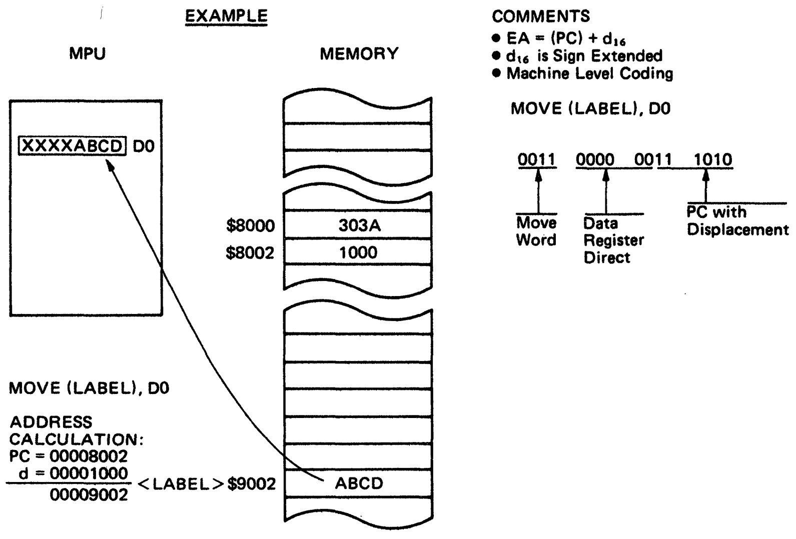
Absolute Long Address

This address mode requires two words of extension. The address of the operand is developed by the concatenation of the extension words. The high-order part of the address is the first extension word; the low-order part of the address is the second extension word. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.



Program Counter With Displacement

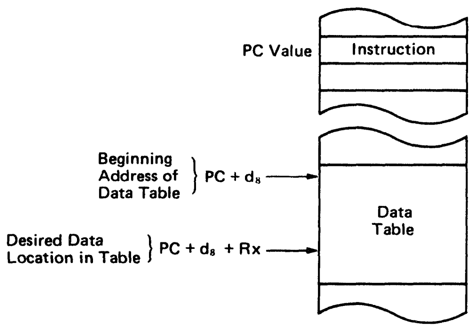
This address mode requires one word of extension. The address of the operand is the sum of the address in the program counter and the sign-extended 16-bit displacement integer in the extension word. The value in the program counter is the address of the extension word. The reference is classified as a program reference.



Program Counter With Index

This address mode requires one word of extension. This address is the sum of the address in the program counter, the sign-extended displacement integer in the lower eight bits of the extension word, and the contents of the index register. The value in the program counter is the address of the extension word. This reference is classified as a program reference.

EA = (PC) + (Rx) + d_8


PC Value
Instruction
Beginning Address of Data Table
PC + d8
Desired Data Location in Table
PC + d8 + Rx
Data Table

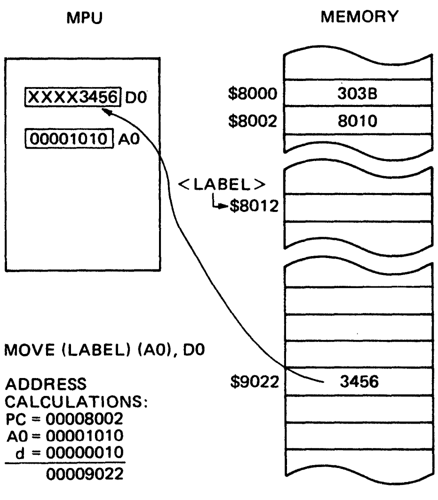
(NOTE)

Extension Word

15	14 13 12	11	10	9	8	7 6 5 4 3 2 1 0
D/A	Register	W/L	0	0	0	Displacement Integer

D/A	: Data Register = 0, Address Register = 1
Register	: Index Register Number
W/L	: Sign-extented, low order Word integer in Index Register = 0 Long Word in Index Register = 1

EXAMPLE

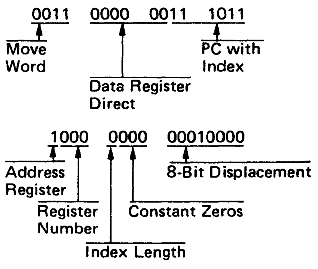


MOVE (LABEL) (A0), D0

ADDRESS CALCULATIONS:
PC = 00008002
A0 = 00001010
d = 00000010
00009022

COMMENTS
- EA = (PC) + (Rx) + d_8
  Where
  PC→Current Program Counter
  Rx→Designated Index Register (Either Data or Address Register)
  d_8→8-Bit Displacement
- Rx and d_8 are Sign Extended
- Rx may be Word or Long Word
  Long Word is Designated with Rx.L
- Machine Level Coding

MOVE (LABEL) (A0), D0



Immediate Data

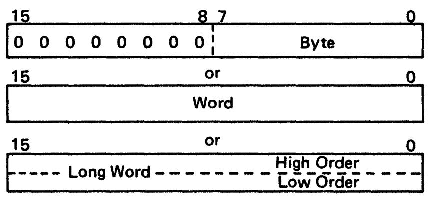
This address mode requires either one or two words of extension depending on the size of the operation.

Byte operation – operand is low order byte of extension word

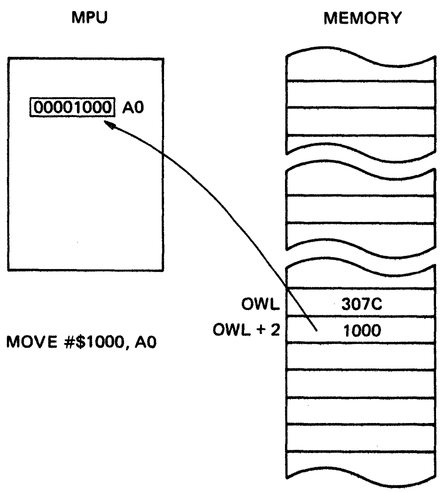
Word operation – operand is extension word

Long word operation – operand is in the two extension words, high-order 16 bits are in the first extension word, low-order 16 bits are in the second extension word.

Extension Word



EXAMPLE



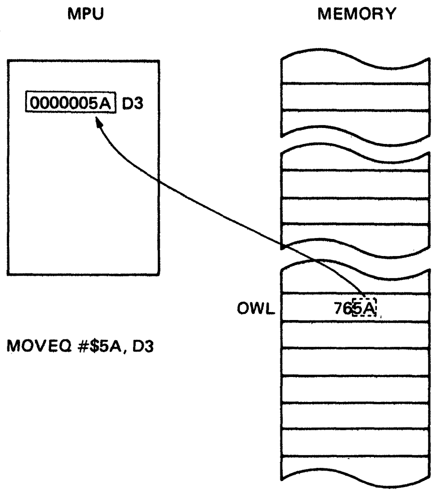
COMMENTS

- Data = Next Word(s)
- Data is Sign Extended for Address Register but not Data Register
- Machine Level Coding

MOVE #$1000, A0

```
0011   0000   0111   1100
Move   Reg #0 Address Immediate
Word          Register Data
              Direct
```

EXAMPLE



COMMENTS

- Inherent Data
- Data is Sign Extended to Long Word
- Destination must be a Data Register
- Machine Level Coding

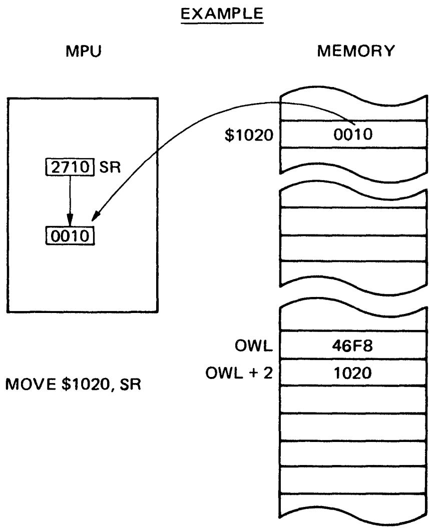
MOVEQ #$5A, D3

```
0111   011    0      0101   1010
Move   Reg #3 Fixed  Immediate
Quick         Zero   Data
```

Condition Codes or Status Register

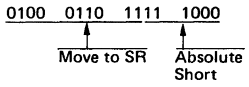
A selected set of instructions may reference the status register by means of the effective address field. These are:

ANDI to CCR
ANDI to SR
EORI to CCR
EORI to SR
ORI to CCR
ORI to SR



COMMENTS
- EA = (Next Word)
- Note: This Example is a Privileged Instruction
- Machine Level Coding

MOVE $1020, SR



• EFFECTIVE ADDRESS ENCODING SUMMARY

Table 15 is a summary of the effective addressing modes discussed in the previous paragraphs.

Table 15 Effective Address Encoding Summary

Addressing Mode	Mode	Register
Data Register Direct	000	register number
Address Register Direct	001	register number
Address Register Indirect	010	register number
Address Register Indirect with Postincrement	011	register number
Address Register Indirect with Predecrement	100	register number
Address Register Indirect with Displacement	101	register number
Address Register Indirect with Index	110	register number
Absolute Short	111	000
Absolute Long	111	001
Program Counter with Displacement	111	010
Program Counter with Index	111	011
Immediate	111	100

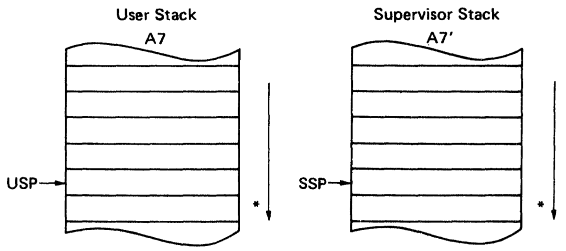
• IMPLICIT REFERENCE

Some instructions make implicit reference to the program counter (PC), the system stack pointer (SP), the supervisor stack pointer (SSP), the user stack pointer (USP), or the status register (SR).

SYSTEM STACK

The system stack is used implicitly by many instructions; user stacks and queues may be created and maintained through the addressing modes. Address register seven (A7) is the system stack pointer (SP). The system stack pointer is either the supervisor stack pointer (SSP) or the user stack pointer (USP), depending on the state of the S-bit in the status register. If the S-bit indicates supervisor state, SSP is the active system stack pointer, and the USP cannot be referenced as an address register. If the S-bit indicates user state, the USP is the active system stack pointer, and the SSP cannot be referenced. Each system stack fills from high memory to low memory.

SYSTEM STACK POINTERS



- Accessed when S = 0
- PC is Stacked on Subroutine Calls in User State

* Increasing Addresses

- Accessed when S = 1
- PC is Stacked on Subroutine Calls in Supervisor State
- Used for Exception Processing

The address mode SP @- creates a new item on the active system stack, and the address mode SP @+ deletes an item from the active system stack.

The program counter is saved on the active system stack on subroutine calls, and restored from the active system stack on returns. On the other hand, both the program counter and the status register are saved on the supervisor stack during the processing of traps and interrupts. Thus, the correct execution of the supervisor state code is not dependent on the behavior of user code and user programs may use the user stack pointer arbitrarily.

In order to keep data on the system stack aligned properly, data entry on the stack is restricted so that data is always put in the stack on a word boundary. Thus byte data is pushed on or pulled from the system stack in the high order half of the word; the lower half is unchanged.

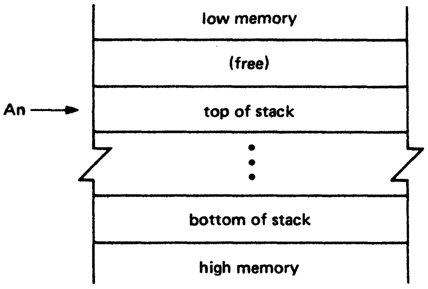
USER STACKS

User stacks can be implemented and manipulated by employing the address register indirect with postincrement and predecrement addressing modes. Using an address register (on of A0 through A6), the user may implement stacks which are filled either from high memory to low memory, or vice versa. The important things to remember are:

- using predecrement, the register is decremented before its contents are used as the pointer into the stack,
- using postincrement, the register is incremented after its contents are used as the pointer into the stack,
- byte data must be put on the stack in pairs when mixed with word or long data so that the stack will not get misaligned when the data is retrieved. Word and long accesses must be on word boundary (even) addresses.

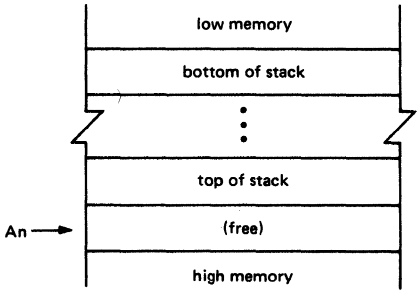
Stack growth from high to low memory is implemented with
An@- to push data on the stack,
An@+ to pull data from the stack.

After eigher a push or a pull operation, register An points to the last (top) item on the stack. This is illustrated as:



Stack growth from low to high memory is implemented with
An@+ to push data on the stack,
An@- to pull data from the stack.

After either a push or a pull operation, register An points to the next available space on the stack. This is illustrated as:

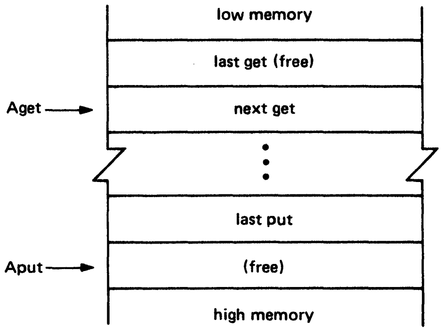


QUEUES

User queues can be implemented and manipulated with the address register indirect with postincrement or predecrement addressing modes. Using a pair of address registers (two of A0 through A6), the user may implement queues which are filled either from high memory to low memory, or vice versa. Because queues are pushed from one end and pulled from the other, two registers are used: the put and get pointers.

Queue growth from low to high memory is implemented with
Aput@+ to put data into the queue,
Aget@+ to get data from the queue.

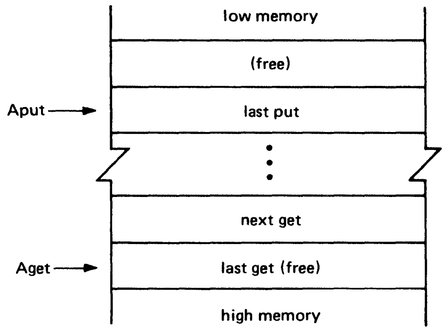
After a put operation, the put address register points to the next available space in the queue and the unchanged get address register points to the next item to remove from the queue. After a get operation, the get address register points to the next item to remove from the queue and the unchanged put address register points to the next available space in the queue. This is illustrated as:



If the queue is to be implemented as a circular buffer, the address register should be checked and, if necessary, adjusted before the put or get operation is performed. The address register is adjusted by subtracting the buffer length (in bytes).

Queue growth from high to low memory is implemented with
Aput@- to put data into the queue,
Aget@ - to get data from the queue.

After a put operation, the put address register points to the last item put in the queue, and the unchanged get address register points to the last item removed from the queue. After a get operation, the get address register points to the last item removed from the queue and the unchanged put address register points to the last item put in the queue. This is illustrated as:



If the queue is to be implemented as a circular buffer, the get or put operation should be performed first, and then the address register should be checked and, if necessary, adjusted. The address register is adjusted by adding the buffer length (in bytes).

• LOGICAL OPERATIONS

Logical operation instructions AND, OR, EOR, and NOT are available for all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provide these logical operations with all sizes of immediate data. Table 16 is a summary of the logical operations.

Table 16 Logical Operations

Instruction	Operand Size	Operation
AND	8, 16, 32	$Dn \wedge (EA) \to Dn$ $(EA) \wedge Dn \to EA$ $(EA) \wedge \#xxx \to EA$
OR	8, 16, 32	$Dn \vee (EA) \to Dn$ $(EA) \vee Dn \to EA$ $(EA) \vee \#xxx \to EA$
EOR	8, 16, 32	$(EA) \oplus Dy \to EA$ $(EA) \oplus \#xxx \to EA$
NOT	8, 16, 32	$\sim (EA) \to EA$

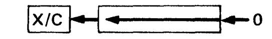
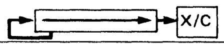
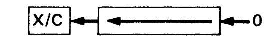
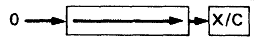
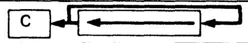
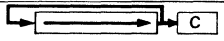
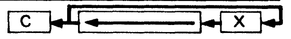
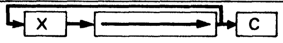
[NOTE] ~ = invert

• SHIFT AND ROTATE OPERATIONS

Shift operations in both directions are provided by the arithmetic instructions ASR and ASL and logical shift instructions LSR and LSL. The rotate instructions (with and without extend) available are ROXR, ROXL, ROR, and ROL. All shift and rotate operations can be performed in either registers or memory. Register shifts and rotates support all operand sizes and allow a shift count specified in the instruction of one to eight bits, or 0 to 63 specified in a data register.

Memory shifts and rotates are for word operands only and allow only single-bit shifts or rotates. Table 17 is a summary of the shift and rotate operations.

Table 17 Shift and Rotate Operations

Instruction	Operand Size	Operation
ASL	8, 16, 32	
ASR	8, 16, 32	
LSL	8, 16, 32	
LSR	8, 16, 32	
ROL	8, 16, 32	
ROR	8, 16, 32	
ROXL	8, 16, 32	
ROXR	8, 16, 32	

• BIT MANIPULATION OPERATIONS

Bit manipulation operations are accomplished using the following instructions: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). Table 18 is a summary of the bit manipulation operations. (Bit 2 of the status register is Z.)

Table 18 Bit Manipulation Operations

Instruction	Operand Size	Operation
BTST	8, 32	~ bit of (EA) → Z
BSET	8, 32	~ bit of (EA) → Z; 1 → bit of EA
BCLR	8, 32	~ bit of (EA) → Z; 0 → bit of EA
BCHG	8, 32	~ bit of (EA) → Z; ~ bit of (EA) → bit of EA

• BINARY CODED DECIMAL OPERATIONS

Multiprecision arithmetic operations on binary coded decimal numbers are accomplished using the following instructions: add decimal with extend (ABCD), subtract decimal with extend (SBCD), and negate decimal with extend (NBCD). Table 19 is a summary of the binary coded decimal operations.

Table 19 Binary Coded Decimal Operations

Instruction	Operand Size	Operation
ABCD	8	$Dx_{10} + Dy_{10} + X \to Dx$ $Ax@ -_{10} + Ay@ -_{10} + X \to Ax@$
SBCD	8	$Dx_{10} - Dy_{10} - X \to Dx$ $Ax@ -_{10} - Ay@ -_{10} - X \to Ax@$
NBCD	8	$0 - (EA)_{10} - X \to EA$

• PROGRAM CONTROL OPERATIONS

Program control operations are accomplished using a series of conditional and unconditional branch instructions and return instructions. These instructions are summarized in Table 20.

The conditional instructions provide setting and branching for the following conditions:

CC	– carry clear	LS	– low or same
CS	– carry set	LT	– less than
EQ	– equal	MI	– minus
F	– never true	NE	– not equal
GE	– greater or equal	PL	– plus
GT	– greater than	T	– always true
HI	– high	VC	– no overflow
LE	– less or equal	VS	– overflow

Table 20 Program Control Operations

Instruction	Operation
Conditional	
B_{CC}	Branch conditionally (14 conditions) 8- and 16-bit displacement
DB_{CC}	Test condition, decrement, and branch 16-bit displacement
S_{CC}	Set byte conditionally (16 conditions)
Unconditional	
BRA	Branch always 8-and 16-bit displacement
BSR	Branch to subroutine 8- and 16-bit displacement
JMP	Jump
JSR	Jump to subroutine
Returns	
RTR	Return and restore condition codes
RTS	Return from subroutine

• SYSTEM CONTROL OPERATIONS

System control operations are accomplished by using privileged instructions, trap generating instructions, and instructions that use or modify the status register. These instructions are summarized in Table 21.

Table 21 System Control Operations

Instruction	Operation
Privileged	
RESET	Reset external devices
RTE	Return from exception
STOP	Stop program execution
ORI to SR	Logical OR to status register
MOVE USP	Move user stack pointer
ANDI to SR	Logical AND to status register
EORI to SR	Logical EOR to status register
MOVE EA to SR	Load new status register
Trap Generating	
TRAP	Trap
TRAPV	Trap on overflow
CHK	Check register against bounds
Status Register	
ANDI to CCR	Logical AND to condition codes
EORI to CCR	Logical EOR to condition codes
MOVE EA to CCR	Load new condition codes
ORI to CCR	Logical OR to condition codes
MOVE SR to EA	Store status register

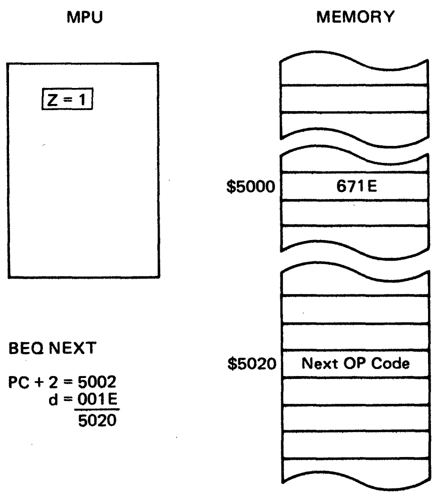
• BRANCH INSTRUCTION ADDRESSING

BRANCH INSTRUCTION FORMAT

	15 … 8	7 … 0
Operation Word	Operation Code	8 bit Displacement
Extension Word	16 bit Displacement if 8 bit Displacement = 0	

RELATIVE, FORWARD REFERENCE, 8-BIT OFFSET

EXAMPLE



BEQ NEXT

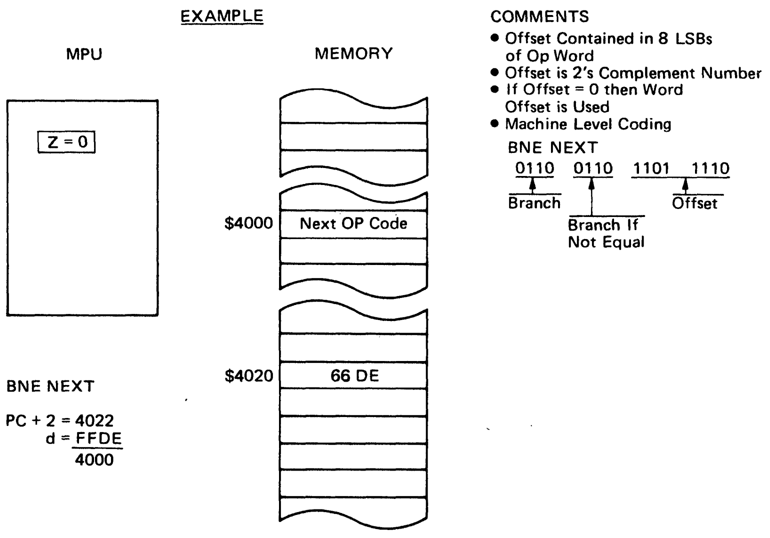
PC + 2 = 5002
d = 001E
5020

COMMENTS
- Offset Contained in 8 LSBs of Op Word
- Offset is 2's Complement Number
- If Offset = 0 then Word Offset is Used
- Machine Level Coding

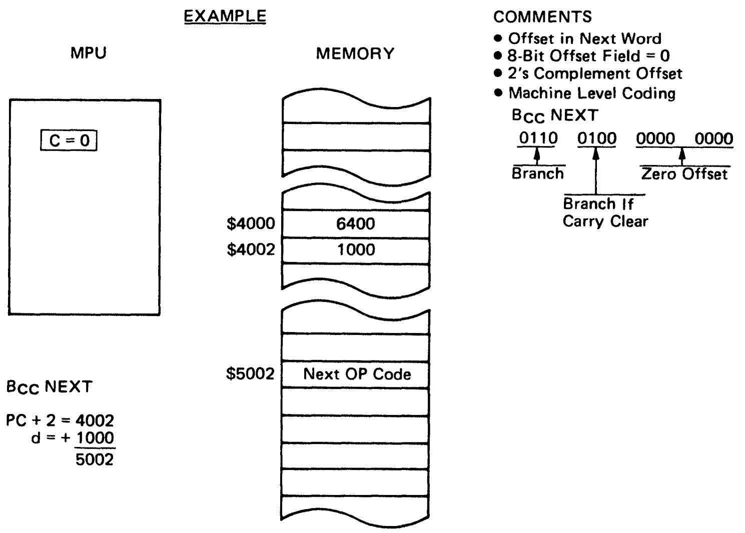
BEQ NEXT

$\underbrace{0110}_{\text{Branch}}\ \underbrace{0111}_{\text{Branch If Equal}}\ \underbrace{0001\ \ 1110}_{\text{Offset}}$

RELATIVE, BACKWARD REFERENCE 8-BIT OFFSET



RELATIVE, FORWARD REFERENCE, 16-BIT OFFSET



■ CONDITION CODES COMPUTATION

This provides a discussion of how the condition codes were developed, the meanings of each bit, how they are computed, and how they are represented in the instruction set details.

• CONDITION CODE REGISTER

The condition code register portion of the status register contains five bits:

N – Negative
Z – Zero
V – Overflow
C – Carry
X – Extend

The first four bits are true condition code bits in that they reflect the condition of the result of a processor operation. The X-bit is an operand for multiprecision computations. The carry bit (C) and the multiprecision operand extend bit (X) are separate in the HD68000 to simplify the programming model.

• CONDITION CODE REGISTER NOTATION

In the instruction set details, the description of the effect on the condition codes is given in the following form:

Condition Codes:

X	N	Z	V	C

Where

N (negative)	set if the most significant bit of the result is set. Cleared otherwise.
Z (zero)	set if the result equals zero. Cleared otherwise.
V (overflow)	set if there was an arithmetic overflow. This implies that the result is not representable in the operand size. Cleared otherwise.
C (carry)	set if a carry is generated out of the most significant bit of the operands for an addition. Also set if a borrow is generated in a subtraction. Cleared otherwise.
X (extend)	transparent to data movement. When affected, it is set the same as the C-bit.

The notational convention that appears in the representation of the condition code registers is:

*	set according to the result of the operation
–	not affected by the operation
0	cleared
1	set
U	undefined after the operation

• CONDITION CODE COMPUTATION

Most operations take a source operand and a destination operand, compute, and store the result in the destination location. Unary operations take a destination operand, compute, and store the result in the destination location. Table 22 details how each instruction sets the condition codes.

Table 22 Condition Code Computations

Operations	X	N	Z	V	C	Special Definition
ABCD	*	U	?	U	?	C = Decimal Carry $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
ADD, ADDI, ADDQ	*	*	*	?	?	$V = Sm \cdot Dm \cdot \overline{Rm} + \overline{Sm} \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot Dm + \overline{Rm} \cdot Dm + Sm \cdot \overline{Rm}$
ADDX	*	*	?	?	?	$V = Sm \cdot Dm \cdot \overline{Rm} + \overline{Sm} \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot Dm + \overline{Rm} \cdot Dm + Sm \cdot \overline{Rm}$ $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT, NOT, TAS, TST	–	*	*	0	0	
CHK	–	*	U	U	U	
SUB, SUBI SUBQ	*	*	*	?	?	$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + Rm \cdot \overline{Dm} + Sm \cdot Rm$
SUBX	*	*	?	?	?	$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + Rm \cdot \overline{Dm} + Sm \cdot Rm$ $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
CMP, CMPI, CMPM	–	*	*	?	?	$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + Rm \cdot \overline{Dm} + Sm \cdot Rm$
DIVS, DIVU	–	*	*	?	0	V = Division Overflow
MULS, MULU	–	*	*	0	0	
SBCD, NBCD	*	U	?	U	?	C = Decimal Borrow $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
NEG	*	*	*	?	?	$V = Dm \cdot Rm$, $C = Dm + Rm$
NEGX	*	*	?	?	?	$V = Dm \cdot Rm$, $C = Dm + Rm$ $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
BTST, BCHG, BSET, BCLR	–	–	?	–	–	$Z = \overline{Dn}$
ASL	*	*	*	?	?	$V = Dm \cdot (\overline{D_{m-1}} + \ldots + \overline{D_{m-r}})$ $+ \overline{Dm} \cdot (D_{m-1} + \ldots + D_{m-r})$ $C = D_{m-r+1}$
ASL (r = 0)	–	*	*	0	0	
LSL, ROXL	*	*	*	0	?	$C = D_{m-r+1}$
LSR (r = 0)	–	*	*	0	0	
ROXL (r = 0)	–	*	*	0	?	C = X
ROL	–	*	*	0	?	$C = D_{m-r+1}$
ROL (r = 0)	–	*	*	0	0	
ASR, LSR, ROXR	*	*	*	0	?	$C = D_{r-1}$
ASR, LSR (r = 0)	–	*	*	0	0	
ROXR (r = 0)	–	*	*	0	?	C = X
ROR	–	*	*	0	?	$C = D_{r-1}$
ROR (r = 0)	–	*	*	0	0	

– Not affected
U Undefined
? Other— see Special Definition

* General Case:
X = C
N = Rm
$Z = \overline{Rm} \cdot \ldots \cdot \overline{R0}$

Sm – Source operand most significant bit
Dm – Destination operand most significant bit
Rm – Result bit most significant bit
n – bit number
r – shift amount

• CONDITIONAL TESTS

Table 23 lists the condition names, encodings, and tests for the conditional branch and set instructions. The test associated with each condition is a logical formula based on the current state of the condition codes. If this formula evaluates to 1, the condition succeeds, or is true. If the formula evaluates to 0, the condition is unsuccessful, or false. For example, the T condition always succeeds, while the EQ condition succeeds only if the Z bit is currently set in the condition codes.

Table 23 Conditional Tests

Mnemonic	Condition	Encoding	Test
T	true	0000	1
F	false	0001	0
HI	high	0010	$\overline{C} \cdot \overline{Z}$
LS	low or same	0011	$C + Z$
CC	carry clear	0100	$\overline{C}$
CS	carry set	0101	C
NE	not equal	0110	$\overline{Z}$
EQ	equal	0111	Z
VC	overflow clear	1000	$\overline{V}$
VS	overflow set	1001	V
PL	plus	1010	$\overline{N}$
MI	minus	1011	N
GE	greater or equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
LT	less than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
GT	greater than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
LE	less or equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

■ INSTRUCTION SET

The following paragraphs provide information about the addressing categories and instruction set of the HD68000.

• ADDRESSING CATEGORIES

Effective address modes may be categorized by the ways in which they may used. The following classifications will be used in the instruction definitions.

Data	If an effective address mode may be used to refer to data operands, it is considered a data addressing effective address mode.
Memory	If an effective address mode may be used to refer to memory operands, it is considered a memory addressing effective address mode.
Alterable	If an effective address mode may be used to refer to alterable (writeable) operands, it is considered an alterable addressing effective address mode.
Control	If an effective address mode may be used to refer to memory operands without an associated size, it is considered a control addressing effective address mode.

Table 24 shows the various categories to which each of the effective address modes belong. Table 25 is the instruction set summary.

The status register addressing mode is not permitted unless it is explicitly mentioned as a legal addressing mode.

These categories may be combined so that additional, more restrictive, classifications may be defined. For example, the instruction descriptions use such classifications as alterable memory or data alterable. The former refers to those addressing modes which are both alterable and memory addresses, and the latter refers to addressing modes which are both data and alterable.

• INSTRUCTION PRE-FETCH

The HD68000 uses a 2-word tightly-coupled instruction prefetch mechanism to enhance performance. This mechanism is described in terms of the microcode operations involved. If the execution of an instruction is defined to begin when the microroutine for that instruction is entered, some features of the prefetch mechanism can be described.

1) When execution of an instruction begins, the operation word and the word following have already been fetched. The operation word is in the instruction decoder.
2) In the case of multi-word instructions, as each additional word of the instruction is used internally, a fetch is made to the instruction stream to replace it.
3) The last fetch from the instruction stream is made when the operation word is discarded and decoding is started on the next instruction.
4) If the instruction is a single-word instruction causing a branch, the second word is not used. But because this word is fetched by the preceding instruction, it is impossible to avoid this superfluous fetch. In the case of an interrupt or trace exception, both words are not used.
5) The program counter usually points to the last word fetched from the instruction stream.

Table 24 Effective Addressing Mode Categories

Effective Address Modes	Mode	Register	Data	Addressing Categories		
				Memory	Control	Alterable
Dn	000	register number	X	–	–	X
An	001	register number	–	–	–	X
An@	010	register number	X	X	X	X
An@+	011	register number	X	X	–	X
An@-	100	register number	X	X	–	X
An@(d)	101	register number	X	X	X	X
An@(d, ix)	110	register number	X	X	X	X
xxx.W	111	000	X	X	X	X
xxx.L	111	001	X	X	X	X
PC@(d)	111	010	X	X	X	–
PC@(d, ix)	111	011	X	X	X	–
#xxx	111	100	X	X	–	–

The following example illustrates many of the features of instruction prefetch. The contents of memory are assumed to be as illustrated in Figure 53.

```
            ORG      0                         DEFINE RESTART VECTOR
            DC.L     INISSP                    INITIAL SYSTEM STACK POINTER
            DC.L     RESTART                   RESTART SYSTEM ENTRY POINT
            ORG      INTVECTOR                 DEFINE AN INTERRUPT VECTOR
            DC.L     INTHANDLER                HANDLER ADDRESS FOR THIS VECTOR
            ORG                                SYSTEM RESTART CODE
RESTART:
            NOP                                NO OPERATION EXAMPLE
            BRA.S    LABEL                     SHORT BRANCH
            ADD.W    D0, D1                    ADD REGISTER TO REGISTER
LABEL:
            SUB.W    DISP(A0), A1              SUBTRACT REGISTER INDIRECT WITH OFFSET
            CMP.W    D2, D3                    COMPARE REGISTER TO REGISTER
            SGE.B    D7                        Scc TO REGISTER
            ...
            ...
INTHANDLER:
            MOVE.W   LONGADR1, LONGADR2        MOVE WORD FROM AND TO LONG ADDRESS
            NOP                                NO OPERATION
            SWAP.W                             REGISTER SWAP
```

Figure 53 Instruction Prefetch Example, Memory Contents

The sequence we shall illustrate consists of the power-up reset, the execution of NOP, BRA, SUB, the taking of an interrupt, and the execution of the MOVE.W xxx. L to yyy.L. The order of operations described within each microroutine is not exact, but is intended for illustrative purpose only.

Microroutine	Operation	Location	Operand
Reset	Read	0	SSP High
	Read	2	SSP Low
	Read	4	PC High
	Read	6	PC Low
	Read	(PC)	NOP
	Read	+(PC)	BR.A
	<begin NOP>		
NOP	Read	+(PC)	ADD
	<begin BRA>		
BRA	PC=PC+d		
	Read	(PC)	SUB
	Read	+(PC)	DISP
	<begin SUB>		
SUB	Read	+(PC)	CMP
	Read	DISP(A0)	<src>
	Read	+(PC)	SGE
	<begin CMP>	<take INT>	
INTERRUPT	Write	−(SSP)	PC Low
	Read	<INT ACK>	Vector #
	Write	−(SSP)	SR
	Write	−(SSP)	PC High
	Read	(VR)	PC High
	Read	+(VR)	PC Low
	Read	(PC)	MOVE
	Read	+(PC)	xxx High
	<begin MOVE>		
MOVE	Read	+(PC)	xxx Low
	Read	+(PC)	yyy High
	Read	xxx	<src>
	Read	+(PC)	yyy Low
	Write	yyy	<dest>
	Read	+(PC)	NOP
	Read	+(PC)	SWAP
	<begin NOP>		

Figure 54 Instruction Prefetch Example

• DATA PREFETCH

Normally the HD68000 prefetches only instructions and not data. However, when the MOVEM instruction is used to move data from memory to registers, the data stream is prefetched in order to optimize performance. As a result, the processor reads one extra word beyond the higher end of the source area. For example, the instruction sequence in Figure 55 will operate as shown in Figure 56.

```
          ...                           MOVE TWO
          MOVEM.L      A, D0/D1         LONGWORDS
                                        INTO REGISTERS
          ...
A         DC.W         1                WORD 1
B         DC.W         2                WORD 2
C         DC.W         3                WORD 3
D         DC.W         4                WORD 4
E         DC.W         5                WORD 5
F         DC.W         6                WORD 6
```

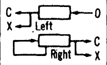
Figure 55 MOVEM Example, Memory Contents

Assume Effective Address Evaluation is Already Done

Microroutine	Operation	Location	Other Operations
	...		
MOVEM	Read	A	
			Prepare to Fill D0
	Read	B	A → D0H
	Read	C	B → D0L
			Prepare to Fill D1
	Read	D	C → D1H
	Read	E	D → D1L
			Detect Register List Complete

Figure 56 MOVEM Example, Operation Sequence

Table 25 Instruction Set

Mnemonic Operation	Size	Addr. Mode	Dn		An		(An)		(An)+		−(An)		d(An)		d(An.Xi)		Abs.W		Abs.L		d(PC)		d(PC,Xi)		s=Immed d=SR/CC		Opcode Bit Pattern 1111 11 5432 1098	7654 3210	Boolean	Condition Codes XNZVC
			#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~				
ABCD	B	s=Dn d=	2	6																							1100 RRR1	0000 0rrr	d10+s10+X→d	*U*U*
Add Digits		s=-(An) d=									2	18															1100 RRR1	0000 1rrr		
ADD	B/W	s=Dn d=			ADDA		2	12	2	12	2	14	4	16	4	18	4	16	6	20							1101 DDD1	SSEE EEEE	d+Dn→d	*****
Add		d=Dn s=	2	4	2*	4	2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	8	1101 DDD0	SSee eeee	Dn+s→Dn	
Binary	L	s=Dn d=			ADDA		2	20	2	20	2	22	4	24	4	26	4	24	6	28							1101 DDD0	10EE EEEE	d+Dn→d	
		d=Dn s=	2	8	2	8	2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1101 DDD1	10ee eeee	Dn+s→Dn	
ADDA	W	d=An s	2	8	2	8	2	12	2	12	2	14	4	16	4	18	4	16	6	20	4	16	4	18	4	12	1101 AAA0	11ee eeee	An+s→An	-----
Add Address	L	d=An s	2	8	2	8	2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1101 AAA1	11ee eeee		
ADDI	B/W	s=Imm d=	4	8	ADDA		4	16	4	16	4	18	6	20	6	22	6	20	8	24							0000 0110	SSEE EEEE	d+#→d	*****
Add Immed	L	s=Imm d=	6	16	ADDA		6	28	6	28	6	30	8	32	8	34	8	32	10	36										
ADDQ	B/W	s=Imm3 d=	2	4	2*	4	2	12	2	12	2	14	4	16	4	18	4	16	6	20							0101 QQQ0	SSEE EEEE	d+#→d	*****
Add Quick	L	s=Imm3 d=	2	8	2	8	2	20	2	20	2	22	4	24	4	26	4	24	6	28										
ADDX	B/W	s=Dn d=	2	4																							1101 RRR1	SS00 0rrr	d+s+X→d	*****
Add Multi-		s= (An) d=									2	18															1101 RRR1	SS00 1rrr		
precision	L	s=Dn d=	2	8																							1101 RRR1	1000 0rrr		
		s= (An) d=									2	30															1101 RRR1	1000 1rrr		
AND	B/W	s=Dn d=					2	12	2	12	2	14	4	16	4	18	4	16	6	20							1100 DDD1	SSEE EEEE	d<and>Dn→d	-**00
Logical And		d=Dn s=	2	4			2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	8	1100 DDD0	SSee eeee	Dn<and>s→Dn	
	L	s=Dn d=					2	20	2	20	2	22	4	24	4	26	4	24	6	28							1100 DDD1	10EE EEEE	d<and>Dn→d	
		d=Dn s=	2	8			2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1100 DDD0	10ee eeee	Dn<and>s→Dn	
ANDI	B/W	s=Imm d=	4	8			4	16	4	16	4	18	6	20	6	22	6	20	8	24					4	20	0000 0010	SSEE EEEE	d<and>#→d	-**00
And Immed.	L	s=Imm d=	6	16			6	28	6	28	6	30	8	32	8	24	8	32	10	36										
ASL, ASR	B/W	count=Dn d=	2	6+2n																							1110 rrrf	SS10 0DDD		*****
Arithmetic		count=#1~8 d=	2	6+2n																							1110 QQQf	SS00 0DDD		
Shift	L	count=Dn d=	2	8+2n																							1110 rrrf	1010 0DDD		
		count=#1~8 d=	2	8+2n																							1110 QQQf	1000 0DDD		
Memory	W	count=1 d=					2*	12	2*	12	2*	14	4*	16	4*	18	4*	16	6*	20							1110 000f	11EE EEEE		
BCHG	B	bit#=Dn d=					2	12	2	12	2	14	4	16	4	18	4	16	6	20							0000 rrr1	01EE EEEE	~(bit)# of d→Z,	--*--
Test and		bit#=Imm d=					4	16	4	16	4	18	6	20	6	22	6	20	8	24							0000 1000	01EE EEEE	~(bit)# of d→	
Change	L	bit#=Dn d=	2	<8																							0000 rrr1	01EE EEEE	(bit)# of d	
		bit#=Imm d=	4	<12																							0000 1000	01EE EEEE		
BCLR	B	bit#=Dn d=					2	12	2	12	2	14	4	16	4	18	4	17	6	20							0000 rrr1	10EE EEEE		
Test and		bit#=Imm d=					4	16	4	16	4	18	6	20	6	22	6	20	8	24							0000 1000	10EE EEEE	~(bit)# of d→Z,	--*--
Clear	L	bit#=Dn d=	2	<10																							0000 rrr1	10EE EEEE	0→(bit)# of d	
		bit#=Imm d=	4	<14																							0000 1000	10EE EEEE		
BSET	B	bit#=Dn d=					2	12	2	12	2	14	4	16	4	18	4	16	6	20							0000 rrr1	11EE EEEE	~(bit)#of d→Z,	--*--
Test and		bit#=Imm d=					4	16	4	16	4	18	6	20	6	22	6	20	8	24							0000 1000	11EE EEEE	1→(bit)# of d	
Set	L	bit#=Dn d=	2	<8																							0000 rrr1	11EE EEEE		
		bit#=Imm d=	4	<12																							0000 1000	11EE EEEE		
BTST	B	bit#=Dn d=					2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14			0000 rrr1	00EE EEEE	~(bit)# of d→Z	--*--
Bit Test		bit#=Imm d=					4	12	4	12	4	14	6	16	6	18	6	16	8	20	6	16	6	18			0000 1000	00EE EEEE		
	L	bit#=Dn d=	2	6																							0000 rrr1	00EE EEEE		
		bit#=Imm d=	4	10																							0000 1000	00EE EEEE		
CHK	W			<40	←trap→			<44		<44		<46		<48		<50		<48		<52		<48		<50		<44	0100 DDD1	10ee eeee	If Dn<0, or	-*UUU
Check Reg-		d=Dn s=	2				2		2		2		4		4		4		6		4		4		4				Dn>(bound),	
ister Against		(bound)		10	←no→			14		14		16		18		20		18		22		18		20		14			then trap	
Bounds					trap																									
CLR	B/W	d=	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 0010	SSEE EEEE	d→MPU	-0100
Clear Operand	L	d=	2	6			2	20	2	20	2	22	4	24	4	26	4	24	6	28									0→d	
CMP	B/W	d=Dn s=	2	4	2*	4	2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	8	1011 DDD0	SSee eeee	Dn−s	-****
Compare	L	d=Dn s=	2	6	2	6	2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14				
Binary																														
CMPA	W	d=An s=	2	6	2	6	2	10	2	10	2	12	4	14	4	16	4	12	6	18	4	14	4	16	4	10	1011 AAA0	11ee eeee	An−s	-****
Compare	L	d=An s=	2	6	2	6	2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1011 AAA1	11ee eeee		
Address																														
CMPI	B/W	s=Imm d=	4	8	CMPA		4	12	4	12	4	14	6	16	6	18	6	16	8	20							0000 1100	SSEE EEEE	d−#	-****
Compare Imm.	L	s=Imm d=	6	14	CMPA		6	20	6	20	6	22	8	24	8	26	8	24	10	28										
CMPM	B/W	s=(An)+ d=							2	12																	1011 RRR1	SS00 1rrr	d−s	-****
Compare	L	s=(An)+ d=							2	20																				
Memory																														
DIVS	W	d=Dn s=	2	<158			2	<162	2	<162	2	<164	4	<166	4	<168	4	<166	6	<170	4	<166	4	<168	4	<162	1000 DDD1	11ee eeee	Dn32/s16→	-***0
Divide Signed																													Dn(r:q)	
DIVU	W	d=Dn s=	2	<140			2	<144	2	<144	2	<146	4	<148	4	<150	4	<148	6	<152	4	<148	4	<150	4	<144	1000 DDD0	11ee eeee	Dn32/s16→	-***0
Divide																													Dn(r:q)	
Unsigned																														
EOR	B/W	s=Dn d=	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20							1011 rrr1	SSEE EEEE	d●Dn→d	-**00
Exclusive OR	L	s=Dn d=	2	8			2	20	2	20	2	22	4	24	4	26	4	24	6	28										
Logical																														
EORI	B/W	s=Imm d=	4	8			4	16	4	16	4	18	6	20	6	22	6	20	8	24					4	20	0000 1010	SSEE EEEE	d●#→d	-**00
Exclusive OR	L	s=Imm d=	6	16			6	28	6	28	6	30	8	32	8	34	8	32	10	36										
Immediate																														
EXG	L	s=Dn d=	2	6																							1100 DDD1	0100 0DDD	s↔d	-----
Exchange		s=An d=	2	6	2	6																					1100 AAA1	0100 1AAA		
Registers																											1100 DDD1	1000 1AAA		
EXT	W	d=	2	4																							0100 1000	1000 0DDD	bit 7→bit 8~15	
Sign Extend	L	d=	2	4																							0100 1000	1100 0DDD	bit 15→bit 16~31	-**00
LEA	L	d=An s=					2	4					4	8	4	12	4	8	6	12	4	8	4	12			0100 AAA1	11ee eeee	s→An	-----
Load Effect-																														
ive Address																														
LINK		disp=Imm s=			4	16																					0100 1110	0101 0AAA	An→−(SP)	-----
Link and																													SP→An	
Allocate																													SP+disp→SP	

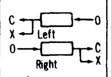
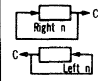
Note : Refer to "Condition Code Computations" as for condition Code.
* Word only
<; Maximum value
#; Number of Program Bytes
~; Number of Clock Periods

Opcode Bit Pattern Key

A; Address Register #
C; Test Condition
D; Data Register #
e; Source Effective Address
E; Destination Effective Address
f; Direction; 0−Right, 1−Left
M; Destination EA Mode
P; Displacement
Q; Quick Immediate Data
r; Source Register
R; Destination Register
S; Size; 00−Byte
01−Word
10−Long Word
11−Another Operation
V; Vector #

(In the MOVE Instruction
01 Byte
10 - Long Word
11 - Word)

(to be continued)

Mnemonic Operation	Size	Addr. Mode	Dn		An		(An)		(An)+		−(An)		d(An)		d(An, Xi)		Abs.W		Abs.L		d(PC)		d(PC, Xi)		s=Immed d=SR/CC		Opcode Bit Pattern 1111 11 5432 1098	7654 3210	Boolean	Condition Codes XNZVC
			#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~				
LSL, LSR	B/W	count=Dn d=	2	6+2n																							1110 rrrf	SS10 1DDD		***0*
Logical Shift		count=#1~8 d=	2	6+2n																							1110 QQQf	SS00 1DDD		
	L	count=Dn d=	2	8+2n																							1110 rrrf	1010 1DDD		
		count=#1~8 d=	2	8+2n																							1110 QQQf	1000 1DDD		
Memory	W	count=1 d=					2*	12	2*	12	2*	14	4*	16	4*	18	4*	16	6*	20							1110 001f	11EE EEEE		
MOVE	B/W	s=Dn d=	2	4	MOVEA		2	8	2	8	2	8	4	12	4	14	4	12	6	16							00SS RRRM	MMee eeee	s→d	-**00
Move Data		s=An s=	2	4	MOVEA		2	8	2	8	2	8	4	12	4	14	4	12	6	16										
		s=(An) d=	2	8	MOVEA		2	12	2	12	2	12	4	16	4	18	4	16	6	20										
		s=(An)+ d=	2	8	MOVEA		2	12	2	12	2	12	4	16	4	18	4	16	6	20										
		s=−(An) d=	2	10	MOVEA		2	14	2	14	2	14	4	18	4	20	4	18	6	22										
		s=d(An) d=	4	12	MOVEA		4	16	4	16	4	16	6	20	6	22	6	20	8	24										
		s=d(An,X) d=	4	14	MOVEA		4	18	4	18	4	18	6	22	6	24	6	22	8	26										
		s=Abs.W d=	4	12	MOVEA		4	16	4	16	4	16	6	20	6	22	6	20	8	24										
		s=Abs.L d=	6	16	MOVEA		6	20	4	20	6	20	8	24	8	26	8	24	10	28										
		s=d(PC) d=	4	12	MOVEA		4	16	4	16	4	16	6	20	6	22	6	20	8	24										
		s=d(PC,X) d=	4	14	MOVEA		4	18	4	18	4	18	6	22	6	24	6	22	8	26										
		s=Imm d=	4	8	MOVEA		4	12	4	12	4	12	6	16	6	18	6	16	8	20										
	L	s=Dn d=	2	4	MOVEA		2	12	2	12	2	12	4	16	4	18	4	16	6	20										
		s=An d=	2	4	MOVEA		2	12	2	12	2	12	4	16	4	18	4	16	6	20										
		s=(An) d=	2	12	MOVEA		2	20	2	20	2	20	4	24	4	26	4	24	6	28										
		s=(An)+ d=	2	12	MOVEA		2	20	2	20	2	20	4	24	4	26	4	24	6	28										
		s= (An) d=	2	14	MOVEA		2	22	2	22	2	22	4	26	4	28	4	26	6	30										
		s=d(An) d=	4	16	MOVEA		4	24	4	24	4	24	6	28	6	30	6	28	8	32										
		s=d(An,X) d=	4	18	MOVEA		4	26	4	26	4	26	6	30	6	32	6	30	8	34										
		s=Abs W d=	4	16	MOVEA		4	24	4	24	4	24	6	28	6	30	6	28	8	32										
		s=Abs.L d=	6	20	MOVEA		6	28	4	28	6	28	8	32	8	34	8	32	10	36										
		s=d(PC) d=	4	16	MOVEA		4	24	4	24	4	24	6	28	6	30	6	28	8	32										
		s=d(PC,X) d=	4	18	MOVEA		4	26	4	26	4	26	6	30	6	32	6	30	8	34										
		s=Imm d=	6	12	MOVEA		6	20	6	20	6	20	8	24	8	26	8	24	10	28										
MOVE	W	d=CCR s=	2	12			2	16	2	16	2	18	4	20	4	22	4	20	6	24	4	20	4	22	4	16	0100 0100	11ee eeee	s→CCR	*****
Move to Condition Codes																														
MOVE	W	d=SR s=	2	12			2	16	2	16	2	18	4	20	4	22	4	20	6	24	4	20	4	22	4	16	0100 0110	11ee eeee	s→SR	*****
Move to/from		s=SR d=	2	6			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 0000	11EE EEEE	d→MPU	-----
Status Reg.																													SR→d	
MOVE	L	s=USP d			2	4																					0100 1110	0110 1AAA	USP→An	-----
Move to/from		d=USP s			2	4																					0100 1110	0110 0AAA	An→USP	
User SP(A7)																														
MOVEA	W	d=An s=	2	4	2	4	2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	8	0011 AAA0	01ee eeee	s→An	-----
Move Address	L	d=An s=	2	4	2	4	2	12	2	12	2	14	4	16	4	18	4	16	6	20	4	16	4	18	6	12	0010 AAA0	01ee eeee		
MOVEM	W	s=Xn d					4	8+4n			4	8+4n	6	12+4n	6	14+4n	6	12+4n	8	16+4n							0100 1000	10EE EEEE	Xn→d	-----
Move Multiple																											a7~a0	d7~d0†		
Registers		d=Xn s					4	12+4n	4	12+4n			6	16+4n	6	18+4n	6	16+4n	8	20+4n	6	16+4n	6	18+4n			0100 1100	10ee eeee	s→Xn **	
																											a7~a0	d7~d0		
	L	s=Xn d					4	8+8n			4	8+8n	6	12+8n	6	14+8n	6	12+8n	8	16+8n							0100 1000	11EE EEEE	Xn→d	
																											a7~a0	d7~d0†		
		d=Xn s					4	12+8n	4	12+8n			6	16+8n	6	18+8n	6	16+8n	8	20+8n	6	16+8n	6	18+8n			0100 1100	11ee eeee	s→Xn**	
																											a7~a0	d7~d0		
MOVEP	W	s=Dn d=											4	16													0000 DDD1	1000 1AAA	Dn→d by bytes	-----
Move		s=d(An) d=	4	16																							0000 DDD1	0000 1AAA	s→Dn by bytes	
Peripheral	L	s=Dn d=											4	24													0000 DDD1	1100 1AAA	Dn→d by bytes	
		s=d(An) d=	4	24																							0000 DDD1	0100 1AAA	s→Dn by bytes	
MOVEQ	L	s=Imm8 d=	2	4																							0111 DDD0	QQQQ QQQQ	#→Dn	-**00
Move Quick																														
MULS	W	d=Dn s=	2	<70			2	<74	2	<74	2	<76	4	<78	4	<80	4	<78	6	<82	4	<78	4	<80	4	<74	1100 DDD1	11ee eeee	Dn×s→Dn	-**00
Multiply Signed																														
MULU	W	d=Dn s=	2	<70			2	<74	2	<74	2	<76	4	<78	4	<80	4	<78	6	<82	4	<78	4	<80	4	<74	1100 DDD0	11ee eeee	Dn×s→Dn	-**00
Multiply Unsigned																														
NBCD	B	d=	2	6			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 1000	00EE EEEE	0−d10−X→d	*U*U*
Negate Digit																														
NEG	B/W	d=	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 0100	SSEE EEEE	0−d→d	*****
Negate Binary	L	d=	2	6			2	20	2	20	2	22	4	24	4	26	4	24	6	28										
NEGX	B/W	d=	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 0000	SSEE EEEE	0−d−X→d	*****
Negate Multiprecision	L	d=	2	6			2	20	2	20	2	22	4	24	4	26	4	24	6	28										
NOT	B/W	d=	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 0110	SSEE EEEE	~d→d	-**00
Logical Complement	L	d=	2	6			2	20	2	20	2	22	4	24	4	26	4	24	6	28										
OR	B/W	s=Dn d=					2	12	2	12	2	14	4	16	4	18	4	16	6	20							1000 DDD1	SSEE EEEE	d<or>Dn→d	-**00
Inclusive OR		d=Dn s=	2	4			2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	8	1000 DDD0	SSee eeee	Dn<or>s→Dn	
Logical	L	s=Dn d=					2	20	2	20	2	22	4	24	4	26	4	24	6	28							1000 DDD1	10EE EEEE	d<or>Dn→d	
		d=Dn s=	2	8			2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1000 DDD0	10ee eeee	Dn<or>s→Dn	
ORI	B/W	s=Imm d=	4	8			4	16	4	16	4	18	6	20	6	22	6	20	8	24					4	20	0000 0000	SSEE EEEE	d<or>#→d	-**00
OR Immediate	L	s=Imm d=	6	16			6	30	6	30	6	32	8	34	8	36	8	34	10	38										
PEA	L	s=					2	14					4	18	4	22	4	18	6	22	4	18	4	22			0100 1000	01ee eeee	s→−(SP)	-----
Push Effective Address																														
ROR, ROL	B/W	count=Dn d=	2	6+2n																							1110 rrrf	SS11 1DDD		-**00
Rotate		count=#1~8 d=	2	6+2n																										
without X	L	count=Dn d=	2	8+2n																							1110 QQQf	SS01 1DDD		
		count=#1~8 d=	2	8+2n																							1110 rrrf	1011 1DDD		
Memory	W	count=1 d=					2*	12	2*	12	2*	14	4*	16	4*	18	4*	16	6*	20							1110 QQQf	1001 1DDD		
																											1110 011f	11EE EEEE		

Note : Refer to "Condition Code Computations" as for condition Code.
* : Word only
< : Maximum value
#; Number of Program Bytes
~; Number of Clock Periods

** : The MPU goes through an extra null read cycle after a multiple read is done (The last EA+2).

Opcode Bit Pattern Key

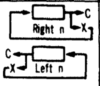
A; Address Register #
C; Test Condition
D; Data Register #
e; Source Effective Address
E; Destination Effective Address

f ; Direction; 0−Right, 1−Left
M; Destination EA Mode
P; Displacement
Q; Quick Immediate Data
r ; Source Register

R; Destination Register
S; Size; 00−Byte
01−Word
10−Long Word
11−Another Operation
V; Vector #

(In the MOVE Instruction
01−Byte
10−Long Word
11−Word)

(to be continued)

Mnemonic Operation	Size	Addr. Mode	Dn		An		(An)		(An)+		−(An)		d(An)		d(An.Xi)		Abs.W		Abs.L.		d(PC)		d(PC.Xi)		s=Immed d=SR/CC		Opcode Bit Pattern 1111 11 5432 1098	7654 3210	Boolean	Condition Codes XNZVC
			#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~				
ROXR,ROXL	B W	count:Dn d=	2	6+2n																							1110 rrrf	SS11 0DDD		***0*
Rotate		count:#1~8 d=	2	6+2n																							1110 QQQf	SS01 0DDD		
through X	L	count:Dn d=	2	8+2n																							1110 rrrf	1011 0DDD		
		count:#1~8 d=	2	8+2n																							1110 QQQf	1001 0DDD		
Memory	W	count:1 d=					2*	12	2*	12	2*	14	4*	16	4*	18	4*	16	6*	20							1110 010f	11EE EEEE		
SBCD	B	s=Dn d=	2	6																							1000 RRR1	0000 0rrr		
Subtract		s=(An) d=									2	18															1000 RRR1	0000 1rrr	d10 s10 X →d	*U*U*
digits																														
Scc	B	cc d=	2	6 4'			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0101 CCCC	11EE EEEE	d →MPU	-----
Set																													If cc true, 1's →d	
Conditionally																													Else, 0's →d	
SUB	B W	s=Dn d=			SUBA		2	12	2	12	2	14	4	16	4	18	4	16	6	20							1001 DDD1	SSEE EEEE	d·Dn →d	*****
Subtract		d=Dn s=	2	4	2*	4	2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	8	1001 DDD0	SSee eeee	Dn s →Dn	
Binary	L	s=Dn d=			SUBA		2	20	2	20	2	22	4	24	4	26	4	24	6	28							1001 DDD1	10EE EEEE	d·Dn →d	
		d=Dn s=	2	8	2	8	2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1001 DDD0	10ee eeee	Dn s →Dn	
SUBA	W	d=An s=	2	8	2	8	2	12	2	12	2	14	4	16	4	18	4	16	6	20	4	16	4	18	4	12	1001 AAA0	11ee eeee	An s →An	-----
Subtract	L	d=An s=	2	8	2	8	2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1001 AAA1	11ee eeee		
Address																														
SUBI	B W	s=Imm d=	4	8	SUBA		4	16	4	16	4	18	6	20	6	22	6	20	8	24							0000 0100	SSEE EEEE	d·# →d	*****
Subtract	L	s=Imm d=	6	16	SUBA		6	28	6	28	6	30	8	32	8	34	8	32	10	36										
Immediate																														
SUBQ	B W	s=Imm3 d=	2	4	2*	4	2	12	2	12	2	14	4	16	4	18	4	16	6	20							0101 QQQ1	SSEE EEEE	d # →d	*****
Subtract	L	s=Imm3 d=	2	8	2	8	2	16	2	16	2	22	4	24	4	26	4	24	6	28										
Quick																														
SUBX	B W	s=Dn d=	2	4																							1001 RRR1	SS00 0rrr	d s X →d	*****
Subtract		s=(An) d=									2	18															1001 RRR1	SS00 1rrr		
Multiprecision	L	s=Dn d=	2	8																							1001 RRR1	1000 0rrr		
		s=(An) d=									2	30															1001 RRR1	1000 1rrr		
SWAP	W	d=	2	4																							0100 1000	0100 0DDD	Dn(31:16)←→	-**00
Swap Regis-																													Dn(15:0)	
ter Halves																														
TAS	B	d=	2	4			2	14	2	14	2	16	4	18	4	20	4	18	6	22							0100 1010	11EE EEEE	test d →cc	-**00
Test and Set																													1 →bit 7 of d	
Operand																														
TST	B W	d=	2	4			2	8	2	8	2	10	4	12	4	14	4	12	6	16							0100 1010	SSEE EEEE	test d →cc	-**00
Test	L	d=	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20										
UNLK					2	12																					0100 1110	0101 1AAA	An →SP,	-----
Unlink																													(SP)+ →An	
Bcc	B	disp=																						bra taken	2	10	0110 CCCC	PPPP PPPP	if cc true,	-----
Branch	W	disp																						bra not taken	2	8'			PC+disp →PC	
Conditionally																								bra taken	4	10				
																								bra not taken	4	14'				
BRA	B	disp=																							2	10	0110 0000	PPPP PPPP	PC+disp →PC	-----
Branch	W	disp=																							4	10				
Always																														
BSR	B	disp=																							2	20	0110 0001	PPPP PPPP	PC →(SP),	-----
Branch																													PC+disp →PC	
to Subroutine	W	disp=																							4	20				
DBcc	W	disp=Imm			**cc**		**Counter**		**Branch**																		0101 CCCC	1100 1DDD	If cc false,	-----
Decrement				10	false		≠1		yes																				Dn−1 →Dn & if	
Counter, &		counter=	4	12'	true		≠1		no																				Dn≠−1, PC+disp →PC	
Branch Until				14'	false		expired		no																				Else, NOP	
Condition																														
True or																														
Count=1																														
JMP		d=					2	8					4	10	4	14	4	10	6	12	4	10	4	14			0100 1110	11EE EEEE	d →PC	-----
Jump to																														
JSR		d=					2	16					4	18	4	22	4	18	6	20	4	18	4	22			0100 1110	10EE EEEE	PC →(SP), d →PC	-----
Jump to																														
Subroutine																														
NOP			2	4																							0100 1110	0111 0001	none	-----
No Operation																														
RESET			2	132																							0100 1110	0111 0000	assert RESET pin	-----
Reset Exter-																														
nal Devices																														
RTE			2	20																							0100 1110	0111 0011	(SP)+ →SR,	*****
Return from																													(SP)+ →PC	
Exception																														
RTR			2	20																							0100 1110	0111 0111	(SP)+ →CC,	*****
Return from																													(SP)+ →PC	
Subroutine/																														
Restore CC																														
RTS			2	16																							0100 1110	0111 0101	(SP)+ →PC	-----
Return from																														
Subroutine																														
STOP																									4	4	0100 1110	0111 0010	# →SR, Wait for	*****
Load SR/Stop																													Interrupt	
TRAP			2	34																							0100 1110	0100 VVVV	PC →−(SSP),	-----
Trap																													SR → (SSP),	
																													(Vector) →PC	
TRAPV			2	34	Trap taken																						0100 1110	0111 0110	If V=1, then PC →	-----
Trap if				4	Trap not																								−(SSP), SR → (SSP)	
Overflow Set					taken																								(TRAPV vector) →PC	
																													else, NOP	

Note : Refer to "Condition Code Computations" as for condition Code.
* Word only
<: Maximum value
#; Number of Program Bytes
~; Number of Clock Periods

Opcode Bit Pattern Key

A; Address Register #	f; Direction; 0−Right, 1−Left	R; Destination Register	
C; Test Condition	M; Destination EA Mode	S; Size; 00−Byte	(In the MOVE Instruction)
D; Data Register #	P; Displacement	01−Word	01−Byte
e; Source Effective Address	Q; Quick Immediate Data	10−Long Word	10−Long Word
E; Destination Effective Address	r; Source Register	11−Another Operation	11−Word
		V; Vector #	

■ INSTRUCTION FORMAT SUMMARY

This provides a summary of the first word in each instruction of the instruction set. Table 26 is an operation code (op-code) map which illustrates how bits 15 through 12 are used to specify the operations. The remaining paragraph groups the instructions according to the op-code map.

where, Size; Byte = 00 Sz; Word = 0
Word = 01 Long Word = 1
Long Word = 10

Table 26 Operation Code Map

Bits 15 thru 12	Operation
0000	Bit Manipulation/MOVEP/Immediate
0001	Move Byte
0010	Move Long
0011	Move Word
0100	Miscellaneous
0101	ADDQ/SUBQ/S_{CC}/DB_{CC}
0110	B_{CC}
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(Unassigned)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate
1111	(Unassigned)

(1) BIT MANIPULATION, MOVE PERIPHERAL, IMMEDIATE INSTRUCTIONS

Dynamic Bit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register			1	Type		Effective Address					

Static Bit

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	Type		Effective Address					

Bit Type Codes: TST = 00, CHG = 01, CLR = 10, SET = 11

MOVEP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register			Op-Mode			0	0	1	Register		

Op-Mode; Word to Reg = 100, Long to Reg = 101, Word to Mem = 110, Long to Mem = 111

OR Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	Size		Effective Address					

AND Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	Size		Effective Address					

SUB Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	Size		Effective Address					

ADD Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	Size		Effective Address					

EOR Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	Size		Effective Address					

CMP Immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	Size		Effective Address					

(2) MOVE BYTE INSTRUCTION

MOVE Byte

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	Destination						Source					
				Register			Mode			Mode			Register		

(3) MOVE LONG INSTRUCTION

MOVE Long

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	Destination						Source					
				Register			Mode			Mode			Register		

(4) MOVE WORD INSTRUCTION

MOVE Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	Destination						Source					
				Register			Mode			Mode			Register		

(5) MISCELLANEOUS INSTRUCTIONS

NEGX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	Size		Effective Address					

MOVE from SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	Effective Address					

CLR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	Size		Effective Address					

NEG

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	Size		Effective Address					

MOVE to CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	Effective Address					

NOT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	Size		Effective Address					

MOVE to SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	Effective Address					

NBCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	Effective Address					

PEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	Effective Address					

SWAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	Register		

MOVEM Registers to EA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	1	Sz	Effective Address					

EXTW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	1	0	0	0	0	Register		

EXTL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	1	1	0	0	0	Register		

TST

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	Size		Effective Address					

TAS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	Effective Address					

MOVEM EA to Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1	Sz	Effective Address					

TRAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	Vector			

LINK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	Register		

UNLK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	Register		

MOVE to USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	0	Register		

MOVE from USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	1	Register		

RESET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

NOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

STOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0

RTE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

RTS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

TRAPV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

RTR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

JSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	Effective Address					

JMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	Effective Address					

CHK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Register			1	1	0	Effective Address					

LEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Register			1	1	1	Effective Address					

(6) ADD QUICK, SUBTRACT QUICK, SET CONDITIONALLY, DECREMENT INSTRUCTIONS

ADDQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Data			0	Size		Effective Address					

SUBQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Data			1	Size		Effective Address					

S_{CC}

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Condition				1	1	Effective Address					

DB_{CC}

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Condition				1	1	0	0	1	Register		

(7) BRANCH CONDITIONALLY, BRANCH TO SUBROUTINE INSTRUCTION

B_{CC}

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	Condition				8 bit Displacement							

BSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8 bit Displacement							

(8) MOVE QUICK INSTRUCTION

MOVEQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	Register			0	Data							

(9) OR, DIVIDE, SUBTRACT DECIMAL INSTRUCTIONS

OR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register			Op-Mode			Effective Address					

Op-Mode			
B	W	L	
000	001	010	Dn ∨ EA → Dn
100	101	110	EA ∨ Dn → EA

DIVU

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register			0	1	1	Effective Address					

DIVS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register			1	1	1	Effective Address					

SBCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Destination Register			1	0	0	0	0	R/M	Source Register		

R/M (register/memory): register — register = 0, memory — memory = 1

(10) SUBTRACT, SUBTRACT EXTENDED INSTRUCTIONS

SUB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Register			Op-Mode			Effective Address					

Op-Mode			
B	W	L	
000	001	010	Dn—EA → Dn
100	101	110	EA—Dn → EA
—	011	111	An—EA → An

SUBX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Destination Register			1	Size		0	0	R/M	Source Register		

R/M (register/memory): register — register = 0, memory — memory = 1

(11) COMPARE, EXCLUSIVE OR INSTRUCTIONS

CMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	Register			Op-Mode			Effective Address					

Op-Mode			
B	W	L	
000	001	010	Dn—EA
—	011	111	An—EA

CMPM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	Register			1	Size		0	0	1	Register		

EOR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	Register			1	Size		Effective Address					

(12) AND, MULTIPLY, ADD DECIMAL, EXCHANGE INSTRUCTIONS

AND

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register			Op-Mode			Effective Address					

Op-Mode			
B	W	L	
000	001	010	Dn ∧ EA → Dn
100	101	110	EA ∧ Dn → EA

MULU

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register			0	1	1	Effective Address					

MULS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register			1	1	1	Effective Address					

ABCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Destination Register			1	0	0	0	0	R/M	Source Register		

R/M (register/memory): register — register = 0, memory — memory = 1

EXGD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Data Register			1	0	1	0	0	0	Data Register		

EXGA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Address Register			1	0	1	0	0	1	Address Register		

EXGM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Data Register			1	1	0	0	0	1	Address Register		

(13) ADD, ADD EXTENDED INSTRUCTIONS

ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register			Op-Mode			Effective Address					

Op-Mode			
B	W	L	
000	001	010	Dn + EA → Dn
100	101	110	EA + Dn → EA
—	011	111	An + EA → An

ADDX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Destination Register			1	Size		0	0	R/M	Source Register		

R/M (register/memory): register — register = 0, memory — memory = 1

(14) SHIFT/ROTATE INSTRUCTIONS

Data Register Shifts

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Count/Register			d	Size		i/r	Type		Register		

Memory Shifts

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	Type		d	1	1	Effective Address					

Shift Type Codes: AS = 00, LS = 01, ROX = 10, RO = 11
d (direction): Right = 0, Left = 1
i/r (count source): Immediate Count = 0, Register Count = 1

■ INSTRUCTION EXECUTION TIMES

The following paragraphs contain listings of the instruction execution times in terms of external clock (CLK) periods. In this timing data, it is assumed that both memory read and write cycle times are four clock periods. Any wait states caused by a longer memory cycle must be added to the total instruction time. The number of bus read and write cycles for each instruction is also included with the timing data. This data is enclosed in parenthesis following the execution periods and is shown as: (r/w) where r is the number of read cycles and w is the number of write cycles.

(NOTE) The number of periods includes instruction fetch and all applicable operand fetches and stores.

● EFFECTIVE ADDRESS OPERAND CALCULATION TIMING

Table 27 lists the number of clock periods required to compute an instruction's effective address. It includes fetching of any extension words, the address computation, and fetching of the memory operand. The number of bus read and write cycles is shown in parenthesis as (r/w). Note there are no write cycles involved in processing the effective address.

● MOVE INSTRUCTION CLOCK PERIODS

Table 28 and 29 indicate the number of clock periods for the move instruction. This data includes instruction fetch, operand reads, and operand writes. The number of bus read and write cycles is shown in parenthesis as: (r/w).

● STANDARD INSTRUCTION CLOCK PERIODS

The number of clock periods shown in Table 30 indicates the time required to perform the operations, store the results, and read the next instruction. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table 30 the headings have the following meanings: An = address register operand, Dn = data register operand, ea = an operand specified by an effective address, and M = memory effective address operand.

● IMMEDIATE INSTRUCTION CLOCK PERIODS

The number of clock periods shown in Table 31 includes the time to fetch immediate operands, perform the operations, store the results, and read the next operation. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table 31, the headings have the following meanings: # = immediate operand, Dn = data register operand, An = address register operand, M = memory operand, CCR = condition code register, and SR = status register.

● SINGLE OPERAND INSTRUCTION CLOCK PERIODS

Table 32 indicates the number of clock periods for the single operand instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 27 Effective Address Calculation Timing

Addressing Mode		Byte, Word	Long
	Register		
Dn	Data Register Direct	0(0/0)	0(0/0)
An	Address Register Direct	0(0/0)	0(0/0)
	Memory		
An@	Address Register Indirect	4(1/0)	8(2/0)
An@ +	Address Register Indirect with Postincrement	4(1/0)	8(2/0)
An@ -	Address Register Indirect with Predecrement	6(1/0)	10(2/0)
An@(d)	Address Register Indirect with Displacement	8(2/0)	12(3/0)
An@(d, ix)*	Address Register Indirect with Index	10(2/0)	14(3/0)
xxx.W	Absolute Short	8(2/0)	12(3/0)
xxx. L	Absolute Long	12(3/0)	16(4/0)
PC@(d)	Program Counter with Displacement	8(2/0)	12(3/0)
PC@(d, ix)*	Program Counter with Index	10(2/0)	14(3/0)
#xxx	Immediate	4(1/0)	8(2/0)

* The size of the index register (ix) does not affect execution time.

Table 28 Move Byte and Word Instruction Clock Periods

Source	Destination								
	Dn	An	An@	An@ +	An@ -	An@(d)	An@(d, ix)*	xxx. W	xxx. L
Dn	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
An	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
An@	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
An@+	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
An@-	10(2/0)	10(2/0)	14(2/1)	14(2/1)	14(2/1)	18(3/1)	20(3/1)	18(3/1)	22(4/1)
An@(d)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
An@(d, ix)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
xxx. W	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
xxx. L	16(4/0)	16(4/0)	20(4/1)	20(4/1)	20(4/1)	24(5/1)	26(5/1)	24(5/1)	28(6/1)
PC@(d)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
PC@(d, ix)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
#xxx	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)

* The size of the index register (ix) does not affect execution time.

Table 29 Move Long Instruction Clock Periods

Source	Destination								
	Dn	An	An@	An@ +	An@ -	An@(d)	An@(d, ix)*	xxx. W	xxx. L
Dn	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
An	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
An@	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
An@+	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
An@-	14(3/0)	14(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)
An@(d)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
An@(d, ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
xxx. W	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
xxx. L	20(5/0)	20(5/0)	28(5/2)	28(5/2)	28(5/2)	32(6/2)	34(6/2)	32(6/2)	36(7/2)
PC@(d)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
PC@(d, ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
#xxx	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)

* The size of the index register (ix) does not affect execution time.

Table 30 Standard Instruction Clock Periods

Instruction	Size	op < ea >, An	op < ea >, Dn	op Dn, < M >
ADD	Byte, Word	8(1/0) +	4(1/0) +	8(1/1) +
	Long	6(1/0) + **	6(1/0) + **	12(1/2) +
AND	Byte, Word	–	4(1/0) +	8(1/1) +
	Long	–	6(1/0) + **	12(1/2) +
CMP	Byte, Word	6(1/0) +	4(1/0) +	–
	Long	6(1/0) +	6(1/0) +	–
DIVS	–	–	158(1/0) + *	–
DIVU	–	–	140(1/0) + *	–
EOR	Byte, Word	–	4(1/0) ***	8(1/1) +
	Long	–	8(1/0) ***	12(1/2) +
MULS	–	–	70(1/0) + *	–
MULU	–	–	70(1/0) + *	–
OR	Byte, Word	–	4(1/0) +	8(1/1) +
	Long	–	6(1/0) + **	12(1/2) +
SUB	Byte, Word	8(1/0) +	4(1/0) +	8(1/1) +
	Long	6(1/0) + **	6(1/0) + **	12(1/2) +

+ add effective address calculation time
* indicates maximum value
** total of 8 clock periods for instruction if the effective address is register direct
*** only available effective address mode is data register direct

Table 31 Immediation Instruction Clock Periods

Instruction	Size	op #, Dn	op #, An	op #, M	op #, CCR/SR
ADDI	Byte, Word	8(2/0)	–	12(2/1) +	–
	Long	16(3/0)	–	20(3/2) +	–
ADDQ	Byte, Word	4(1/0)	8(1/0)*	8(1/1) +	–
	Long	8(1/0)	8(1/0)	12(1/2) +	–
ANDI	Byte, Word	8(2/0)	–	12(2/1) +	20(3/0)
	Long	16(3/0)	–	20(3/1) +	–
CMPI	Byte, Word	8(2/0)	8(2/0)	8(2/0) +	–
	Long	14(3/0)	14(3/0)	12(3/0) +	–
EORI	Byte, Word	8(2/0)	–	12(2/1) +	20(3/0)
	Long	16(3/0)	–	20(3/2) +	–
MOVEQ	Long	4(1/0)	–	–	–
ORI	Byte, Word	8(2/0)	–	12(2/1) +	20(3/0)
	Long	16(3/0)	–	20(3/2) +	–
SUBI	Byte, Word	8(2/0)	–	12(2/1) +	–
	Long	16(3/0)	–	20(3/2) +	–
SUBQ	Byte, Word	4(1/0)	8(1/0)*	8(1/1) +	–
	Long	8(1/0)	8(1/0)	12(1/2) +	–

+ add effective address calculation time
* word only

Table 32 Single Operand Instruction Clock Periods

Instruction	Size	Register	Memory
CLR	Byte, Word	4(1/0)	8(1/1) +
	Long	6(1/0)	12(1/2) +
NBCD	Byte	6(1/0)	8(1/1) +
NEG	Byte, Word	4(1/0)	8(1/1) +
	Long	6(1/0)	12(1/2) +
NEGX	Byte, Word	4(1/0)	8(1/1) +
	Long	6(1/0)	12(1/2) +
NOT	Byte, Word	4(1/0)	8(1/1) +
	Long	6(1/0)	12(1/2) +
Scc	Byte, False	4(1/0)	8(1/1) +
	Byte, True	6(1/0)	8(1/1) +
TAS	Byte	4(1/0)	10(1/1) +
TST	Byte, Word	4(1/0)	4(1/0) +
	Long	4(1/0)	4(1/0) +

+ add effective address calculation time

• SHIFT/ROTATE INSTRUCTION CLOCK PERIODS

Table 33 indicates the number of clock periods for the shift and rotate instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

• BIT MANIPULATION INSTRUCTION CLOCK PERIODS

Table 34 indicates the number of clock periods required for the bit manipulation instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

• CONDITIONAL INSTRUCTION CLOCK PERIODS

Table 35 indicates the number of clock periods required for the conditional instructions. The number of bus read and write cycles is indicated in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

• JMP, JSR, LEA, PEA, MOVEM INSTRUCTION CLOCK PERIODS

Table 36 indicates the number of clock periods required for the jump, jump to subroutine, load effective address, push effective address, and move multiple registers instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w).

Table 33 Shift/Rotate Instruction Clock Periods

Instruction	Size	Register	Memory
ASR, ASL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–
LSR, LSL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–
ROR, ROL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–
ROXR, ROXL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–

Table 34 Bit Manipulation Instruction Clock Periods

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	–	8(1/1) +	–	12(2/1) +
	Long	8(1/0)*	–	12(2/0)*	–
BCLR	Byte	–	8(1/1) +	–	12(2/1) +
	Long	10(1/0)*	–	14(2/0)*	–
BSET	Byte	–	8(1/1) +	–	12(2/1) +
	Long	8(1/0)*	–	12(2/0)*	–
BTST	Byte	–	4(1/0) +	–	8(2/0) +
	Long	6(1/0)	–	10(2/0)	–

+ add effective address calculation time
* indicates maximum value

Table 35 Conditional Instruction Clock Periods

Instruction	Displacement	Trap or Branch Taken	Trap of Branch Not Taken
B_{CC}	Byte	10(2/0)	8(1/0)
	Word	10(2/0)	12(2/0)
BRA	Byte	10(2/0)	–
	Word	10(2/0)	–
BSR	Byte	18(2/2)	–
	Word	18(2/2)	–
DB_{CC}	CC true	–	12(2/0)
	CC false	10(2/0)	14(3/0)
CHK	–	40(5/3) + *	10(1/0) +
TRAP	–	34(4/3)	–
TRAPV	–	34(5/3)	4(1/0)

+ add effective address calculation time
* indicates maximum value

Table 36 JMP, JSR, LEA, PEA, MOMEM Instruction Clock Periods

Instr	Size	An@	An@ +	An@ -	An@(d)	An@(d, ix) *	xxx.W	xxx. L	PC@(d)	PC@(d, ix) *
JMP	–	8(2/0)	–	–	10(2/0)	14(3/0)	10(2/0)	12(3/0)	10(2/0)	14(3/0)
JSR	–	16(2/2)	–	–	18(2/2)	22(2/2)	18(2/2)	20(3/2)	18(2/2)	22(2/2)
LEA	–	4(1/0)	–	–	8(2/0)	12(2/0)	8(2/0)	12(3/0)	8(2/0)	12(2/0)
PEA	–	12(1/2)	–	–	16(2/2)	20(2/2)	16(2/2)	20(3/2)	16(2/2)	20(2/2)
MOVEM	Word	12+4n (3+n/0)	12+4n (3+n/0)	– –	16+4n (4+n/0)	18+4n (4+n/0)	16+4n (4+n/0)	20+4n (5+n/0)	16+4n (4+n/0)	18+4n (4+n/0)
M → R	Long	12+8n (3+2n/0)	12+8n (3+2n/0)	– –	16+8n (4+2n/0)	18+8n (4+2n/0)	16+8n (4+2n/0)	20+8n (5+2n/0)	16+8n (4+2n/0)	18+8n (4+2n/0)
MOVEM	Word	8+4n (2/n)	– –	8+4n (2/n)	12+4n (3/n)	14+4n (3/n)	12+4n (3/n)	16+4n (4/n)	– –	– –
R → M	Long	8+8n (2/2n)	– –	8+8n (2/2n)	12+8n (3/2n)	14+8n (3/2n)	12+8n (3/2n)	16+8n (4/2n)	– –	– –

n is the number of registers to move
* is the size of the index register (ix) does not affect the instruction's execution time

● MULTI-PRECISION INSTRUCTION CLOCK PERIODS

Table 37 indicates the number of clock periods for the multi-precision instructions. The number of clock periods includes the time to fetch both operands, perform the operations, store the results, and read the next instructions. The number of read and write cycles is shown in parenthesis as: (r/w).

In Table 37, the headings have the following meanings: Dn = data register operand and M = memory operand.

Table 37 Multi-Precision Instruction Clock Periods

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte, Word	4(1/0)	18(3/1)
	Long	8(1/0)	30(5/2)
CMPM	Byte, Word	–	12(3/0)
	Long	–	20(5/0)
SUBX	Byte, Word	4(1/0)	18(3/1)
	Long	8(1/0)	30(5/2)
ABCD	Byte	6(1/0)	18(3/1)
SBCD	Byte	6(1/0)	18(3/1)

● MISCELLANEOUS INSTRUCTION CLOCK PERIODS

Table 38 indicates the number of clock periods for the following miscellaneous instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods plus the number of read and write cycles must be added to those of the effective address calculation where indicated.

● EXCEPTION PROCESSING CLOCK PERIODS

Table 39 indicates the number of clock periods for exception processing. The number of clock periods includes the time for all stacking, the vector fetch, and the fetch of the first instruction of the handler routine. The number of bus read and write cycles is shown in parenthesis as: (r/w).

Table 38 Miscellaneous Instruction Clock Periods

Instruction	Size	Register	Memory	Register → Memory	Memory → Register
MOVE from SR	–	6(1/0)	8(1/1) +	–	–
MOVE to CCR	–	12(2/0)	12(2/0) +	–	–
MOVE to SR	–	12(2/0)	12(2/0) +	–	–
MOVEP	Word	–	–	16(2/2)	16(4/0)
	Long	–	–	24(2/4)	24(6/0)
EXG	–	6(1/0)	–	–	–
EXT	Word	4(1/0)	–	–	–
	Long	4(1/0)	–	–	–
LINK	–	16(2/2)	–	–	–
MOVE from USP	–	4(1/0)	–	–	–
MOVE to USP	–	4(1/0)	–	–	–
NOP	–	4(1/0)	–	–	–
RESET	–	132(1/0)	–	–	–
RTE	–	20(5/0)	–	–	–
RTR	–	20(5/0	–	–	–
RTS	–	16(4/0)	–	–	–
STOP	–	4(0/0)	–	–	–
SWAP	–	4(1/0)	–	–	–
UNLK	–	12(3/0)	–	–	–

+ add effective address calculation time

Table 39 Exception Processing Clock Periods

Exception	Periods
Reset	34(6/0)
Address Error	50(4/7)
Bus Error	50(4/7)
Interrupt	44(5/3)*
Illegal Instruction	34(4/3)
Privileged Instruction	34(4/3)
Trace	34(4/3)

* The interrupt acknowledge bus cycle is assumed to take four external clock periods.

■ APPENDIX

● THE 68000S MASK SET

We implement the specification for HD68000-10/-12 and two corrections on the 68000S mask set. One of these corrections is the bus arbitration logic, and the other is a change to correct a RTE/RTR microcode problem.

(1) Bus Arbitration Logic

The problem occurs when bus grant acknowledge ($\overline{BGACK}$) is asserted for **only one clock cycle** while bus request ($\overline{BR}$) is negated. IF $\overline{BR}$ is asserted one clock cycle after $\overline{BGACK}$ is negated, the processor asserts bus grant ($\overline{BG}$) and address strobe ($\overline{AS}$) at the same time (Refer to Figure 58). This, in turn, may cause external DMA logic to run a bus cycle at the same time as the processor cycle, only when those paticular timings are all satisfied. If the DMAC HD68450 is used, this problem can be avoided. Because the HD68450 negates $\overline{BR}$ by one clock after the assertion of $\overline{BGACK}$.

For the 68000S mask set, an internal hardware change is implemented and a timing specification (t_{BGKBR}) is added.

If $\overline{BR}$ and $\overline{BGACK}$ meet the asynchronous set-up time t_{ASI} #47, then t_{BGKBR} can be ignored. If $\overline{BR}$ and $\overline{BGACK}$ are asserted asynchronously with respect to the clock, then $\overline{BGACK}$ has to be asserted before $\overline{BR}$ is negated.

Table 40 t_{BGKBR} Specification

Number	Item	Symbol	Test Condition	4MHz Version HD68000-4 HD68000Y4		6MHz Version HD68000-6 HD68000Y6		8MHz Version HD68000-8 HD68000Y8		10MHz Version HD68000-10 HD68000Y10		12.5MHz Version HD68000-12 HD68000Y12		Unit
				min	max	min	max	min	max	min	max	min	max	
57	$\overline{BGACK}$ "Low" to $\overline{BR}$ "High"	t_{BGKBR}	Fig. 57	30	–	25	–	20	–	20	–	20	–	ns

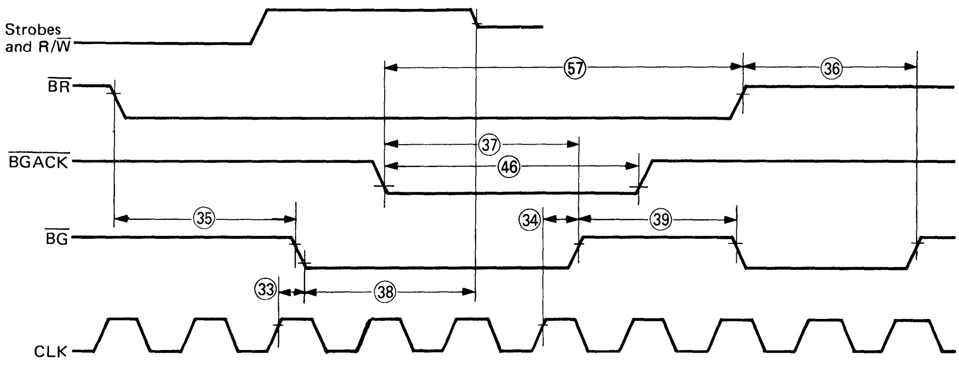


Figure 57 AC Electrical Waveforms – Bus Arbitration

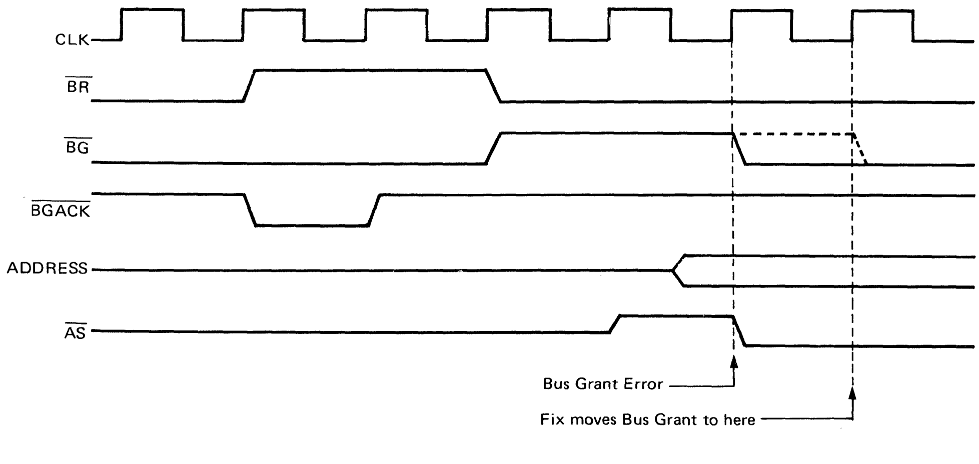
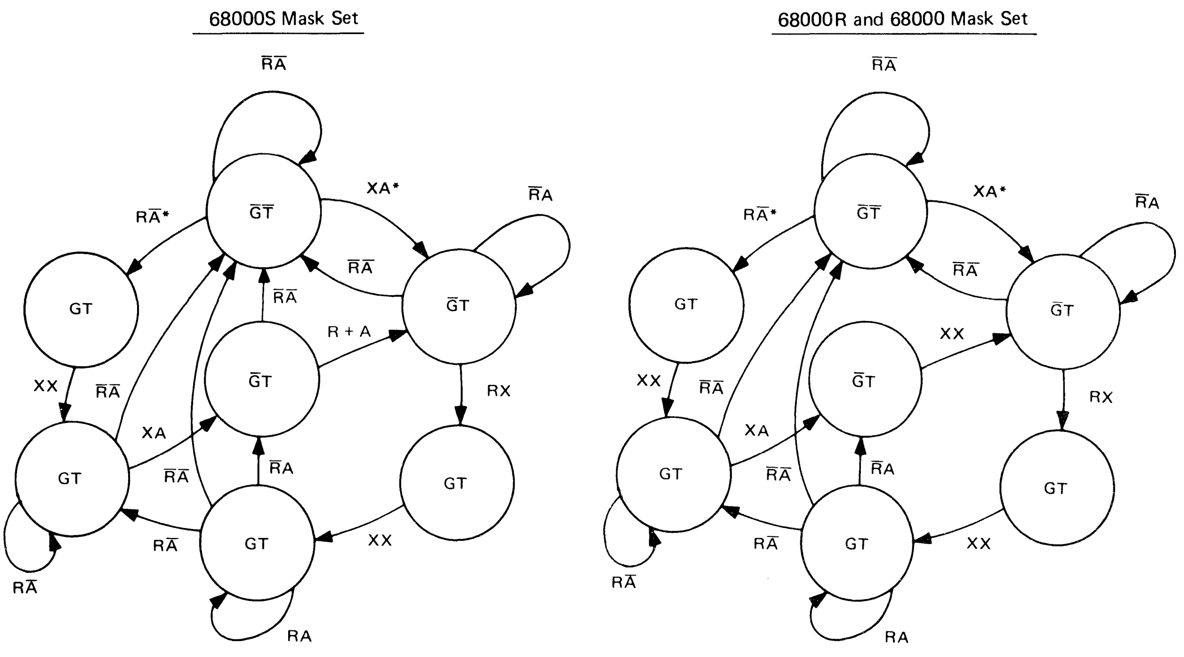


Figure 58 Bus Arbitration Timing Diagram Error Sequence



R = Bus Request Internal
A = Bus Grant Acknowledge Internal
G = Bus Grant
T = Three State Control to Bus Control Logic
X = Don't Care

* State machine will not change state if bus is in S0. Refer to BUS ARBITRATION CONTROL for additional information.

R = Bus Request Internal
A = Bus Grant Acknowledge Internal
G = Bus Grant
T = Three State Control to Bus Control Logic
X = Don't Care

* State machine will not change state if bus is in S0. Refer to BUS ARBITRATION CONTROL for additional information.

Figure 59 State Diagram of HD68000 Bus Arbitration Unit

To Avoid this problem on 68000R mask set, users are recommended to choose one of the followings.

1) Negate $\overline{\text{BR}}$ more than one clock after the assertion of $\overline{\text{BGACK}}$.
2) Avoid the assertion of $\overline{\text{BGACK}}$ for one clock cycle.
3) Reassert $\overline{\text{BR}}$ more than two clocks later than the negation of $\overline{\text{BGACK}}$.
4) Use HD68450 as DMA controllers.

(2) RTE/RTR Microcode Problem

The error in the microcode only affects the RTR and the RTE instructions. These two instructions execute correctly provided there is no bus error.

If there is a bus error on the 2nd, 3rd, or 4th bus cycle of RTR or RTE, the program counter is lost. The program counter loads the stack pointer +2 which is the same address as the access. The results is the program counter containing the stack pointer. This problem can occur on all HD68000 mask sets previous to 68000S.

The fix inhibits the loading of the program counter during this instruction until the 4th bus cycle.

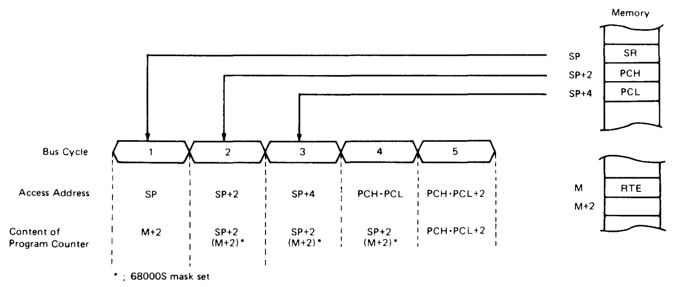


Figure 60 RTE Instruction Bus Cycle

HITACHI

A World Leader in Technology

Hitachi America, Ltd.
Semiconductor and IC Sales and Service Division
2210 O'Toole Avenue, San Jose, CA 95131
1-408-942-1500

HITACHI #U60

Printed in U.S.A.