F-67 (166)

ARITHMETIC PROCESSOR 166
INSTRUCTION MANUAL

VOLUME 1

PDP-6

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

F-67(166)

PDP-6 ARITHMETIC PROCESSOR 166
INSTRUCTION MANUAL

VOLUME 1

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

COPY NO.

This manual contains proprietary information. It is provided to the customers of Digital Equipment Corporation to help them properly use and maintain DEC equipment. Revealing the contents to any person or organization for any other purpose is prohibited.

PREFACE

This instruction manual is published in two volumes to aid personnel in the operation and maintenance of the Arithmetic Processor Type 166 and four of the more common PDP-6 input-output devices: Paper Tape Reader Type 760, Paper Tape Punch Type 761, Teletype Keyboard-Printer Type 626, and Card Reader Type 461. Maintenance information for the in-out devices is confined primarily to those portions of DEC manufacture; separate manuals for the devices themselves are furnished with the system.

The first three chapters present a general description of the system and its operation. Chapter 1 lists the operating specifications and describes the physical and electrical characteristics of the system. Chapter 2 provides a general description of system organization at the block diagram level, explaining what the system does rather than describing the circuit hardware involved in the various functions. This chapter also describes the number system and instruction formats used in the Type 166 Processor. Chapter 3 explains the use of all controls and indicators on the operator control panels and in-out devices, and outlines basic operating procedures.

The next five chapters present a complete, detailed description of the system logic. Chapter 4, Drawing Conventions and Flow Charts, discusses PDP-6 documentation and describes the symbols and terminology used in the logic drawings and flow charts. This chapter also escorts the reader through the flow charts in sequence, so that he may better understand the flow of operations in the processor, and discusses in detail some sequences that neither appear as coherent hardware units in the logic drawings nor are obvious from the flow charts. The next two chapters in this group describe the hardware for the main control sequence and logical and arithmetic processing; the final two describe the processor interfaces with the memory bus and the in-out bus Also included in the last chapter are the control units for four common in-out devices. The reader is strongly advised not to embark upon any logic chapter in this or any other PDP-6 system manual without first gaining a thorough understanding of the material presented in Chapter 4.

Chapter 9 contains information useful in maintaining the system, including a discussion of maintenance operation, maintenance programs, and preventive and corrective maintenance.

Following Chapter 9 are appendixes on engineering drawings and spares, a glossary, and several convenient tables. All logic drawings and flow charts referred to in the text are in Volume 2; all other figures are interleaved with the text.

FOREWORD

PDP-6 is a general-purpose, digital computing system consisting of processors, memories, and input-output devices, each of which has independent internal timing. Processors in a system may share memories and input-output equipment; the memories themselves may have different speeds.

A central processor, usually the Type 166 Arithmetic Processor, performs arithmetic and logic operations and governs the movement of information between memory and peripheral devices. The Type 166 includes an executive system that allows a number of programs, each restricted to a definite area in core, to share processor time. The central processor uses two busses for system intercommunication, one to the memory system, the other to its input-output devices. A system may contain any number of central processors, each with a memory bus and an in-out bus. The memory buses permit the memory complex to accommodate several processors— as many as four may address a single memory module. In order to deposit or retrieve information, the processor supplies an address and requests a memory cycle. Upon accepting the request, the addressed memory times its own cycle and furnishes the appropriate response to the processor.

Through the in-out bus the processor controls all information transfers to and from the peripheral equipment. A priority interrupt system in the processor allows a device to signal when it needs service so that the processor is free during the actual input-output time. One of the units that may be connected to the bus is the Type 167, an autonomous drum processor which supplies direct memory access for high-speed devices such as drums, discs, magnetic tape, and displays. In addition to its in-out bus connection to the central processor, the drum processor is itself connected to the memory system via its own memory bus, and has its own smaller scale in-out bus through which it may govern up to three input-output control units. Although the central processor must provide initial conditions and commands, the drum processor then operates independently, so large blocks of information may be transferred between an in-out device and a memory without reducing central processor efficiency.

All PDP-6 memories store words of 36 bits but may be of different sizes and speeds. Core memories usually have core banks of 8192 or 16,384 words. Cycle times for reading from and writing back into memory are typically 2 and 5 μsec, although in each case access time is much shorter: when reading, the processor need wait only until data is available; when writing, only until data is accepted. A fast flip-flop memory, with access time less than 1/2 μsec, is normally used instead of the bottom 16 locations in core.

The instruction format allows the basic instructions to address one of 262,144 locations in memory for an operand, one of 15 index registers for modifying the memory address, and one of 16 accumulators for a second operand. Instruction results may be stored in an accumulator, in memory, or in both. In-out instructions govern the transfer of data in both directions over the in-out bus, the transfer of control information, including priority interrupt channel assignments, to the peripheral equipment, and the gathering of status information from that equipment. In addition to addressing a memory location and an index register, an in-out instruction may address one of 128 devices, two of which are the priority interrupt system and the processor itself.

For further information on the overall system, refer to PDP-6 Programming (DEC publication K-06), which also describes system software and discusses programming for the processor and most in-out devices. Maintenance documentation for the system is provided by a series of manuals. This one discusses system maintenance for the Type 166 Arithmetic Processor and several common in-out devices, others cover the several types of memories that may be used in a PDP-6 memory system, and still others treat the drum equipment, magnetic tape equipment, DECtape, and other in-out devices. A separate circuit manual discusses circuit maintenance and describes most standard circuits including all those used in the equipment described in the present manual and all logic circuits used in the memories. Descriptions of specialized circuits, such as those associated with the core stack, reading and writing on magnetic tape, and the like, are included in the appropriate system manuals.

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

The Type 166 is a general-purpose central processor that performs all of the arithmetic, logical, executive, and internal data transmission operations in a PDP-6 system. It also controls all transfers of data between memory and peripheral equipment, although in many cases it may provide control merely by supplying system commands and initial conditions to an in-out processor. It contains two bus interfaces, one for connection to memory, the other to the input-output system.

Except for certain control information held permanently in the processor, the state of the processor resides entirely in memory. The only information carried over by the processor from one instruction to the next is the program count, flags, and information for a user mode which allows a number of programs, each restricted to a definite area in core, to share computer time. Besides operating on a stored program, the processor must retrieve all operands for every instruction, and all data and results of computations are stored at the completion of an instruction. Thus the arithmetic registers in the processor contain information only during actual processing and the registers used for address modification are the same as those used for computations within a single instruction. The accumulators, 15 of which double as index registers, actually occupy the bottom 16 memory locations and are usually contained in a fast memory. Most basic instructions have three addresses which select an accumulator, a memory location (which may be another accumulator), and an index register for memory address modification. All instructions may use multiple-level indirect addressing and some may use a single address to call two adjacent accumulators for processing double-length operands. With a single instruction, the processor is capable of performing a full-word or half-word transfer, a block transfer, or the manipulation of a character (byte) of variable size. The processor includes hardware for performing Boolean functions, shift operations, both fixed- and floating-point arithmetic, jumps, logical and arithmetic comparisons, and a variety of modification and testing instructions.

In addition to standard instruction operations, the hardware also includes a program-assignable priority interrupt system through which an external device or an internal condition can interrupt the normal program sequence; a number of flags that allow checking of various conditions and facilitate double-precision arithmetic; and memory protection and relocation registers that allow an executive routine to assign a specific area in core to each user program.

1.1 OPERATING SPECIFICATIONS

All timing in the Type 166 is completely asynchronous, and all processing is done in parallel except for a few extremely fast serial functions, such as carry propagation in the main arithmetic register and in dc adders that relocate memory addresses. Information handled by the processor has the following characteristics:

Word Length	36 bits
Instruction Format	
Basic	Instruction code, 9 bits Accumulator address, 4 bits Indirect, 1 bit Index register address, 4 bits Memory address, 18 bits
Input-Output	Instruction code, 6 bits Device code, 7 bits Indirect, 1 bit Index register address, 4 bits Memory address, 18 bits
Internal Number System	Binary
Negative Representation	2's complement
Number Format	
Fixed Point	Sign, 1 bit; magnitude, 35 bits
Floating Point	Sign, 1 bit; exponent, 8 bits; fraction, 27 bits

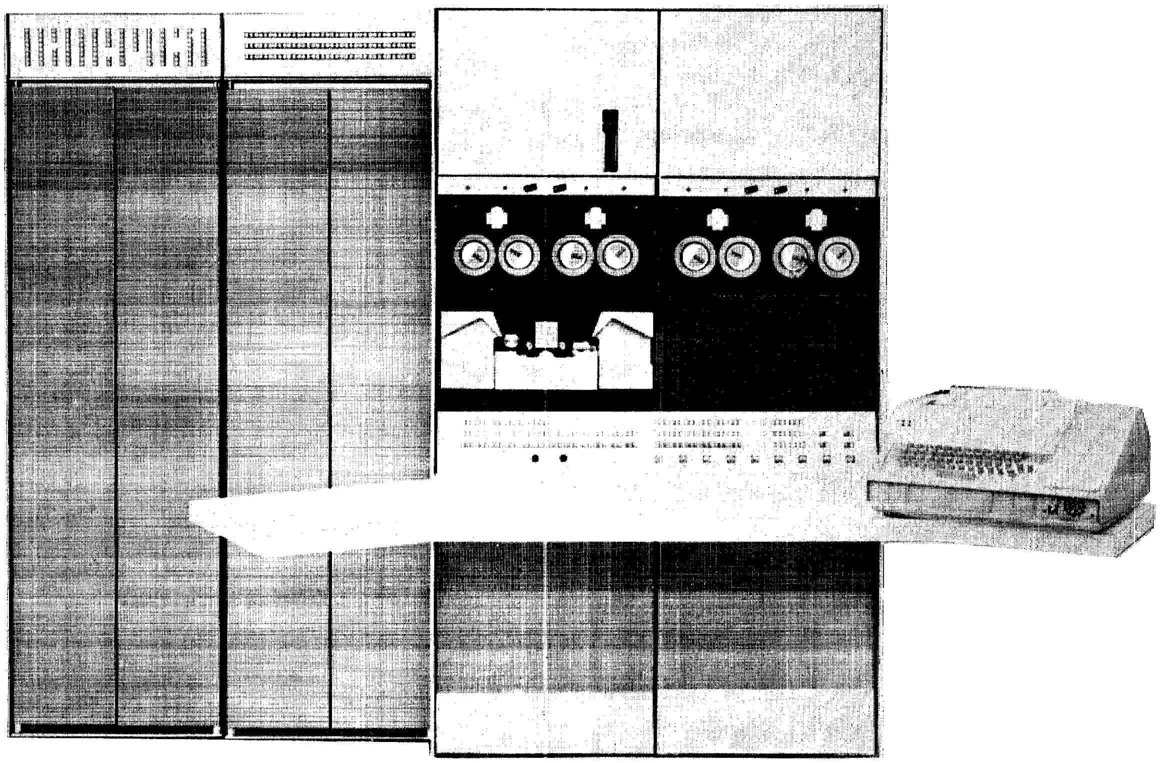
The time required for execution of any particular instruction varies tremendously because of the completely asynchronous operation. The basic operations, such as addition in the arithmetic register or a sequence of shifts controlled by the shift counter, are performed by built-in hardware subroutines. These are called whenever necessary either from the main instruction sequence or by special sequences such as byte manipulation, block transfer, floating add-subtract, divide, etc., which are in turn entered from the main instruction sequence. Even at the level of individual events, the execution time may vary; for example in the basic addition or subtraction subroutine, the complement function and the parital addition each require 100 nanoseconds but carry propagation, which is serial, depends upon the number of carries needed. Similarly on a larger scale, multiplication and division are performed by a series of additions and subtractions and the time required for such a major sequence depends upon the number of times it must call various subsequences. Most processor control functions involved in the retrieval and setup of instructions, and retrieval and storage of operands take a negligible amount of time when compared to memory access time. Exact instruction execution times may be determined from the flow charts included in Chapter 4. For each memory access, the processor must first check for memory protection and relocation and then wait until the addressed memory is free; the time required for access once the memory is free depends upon the type of memory.

The processor must set up all transfers of control information and data to and from the peripheral equipment; but since a device can signal the processor by means of the priority interrupt system when it requires service, no processor time need be lost in waiting, and processor and peripheral equipment can operate in parallel. Every transfer over the I/O bus does, however, require 2.5 microseconds. The four I/O devices included in this manual have the following operating specifications.

Paper Tape Reader	400 8-bit characters per second
Paper Tape Punch	63.3 8-bit characters per second
Keyboard-Printer	10 8-bit characters per second
Card Reader	200 80-column cards per minute

1.2 PHYSICAL CHARACTERISTICS

Most DEC equipment is housed in steel bays with aluminum control panels. The arithmetic processor with its console uses four such bays bolted together. The front of each bay can accommodate up to twelve 19 inch by 5-1/4 inch panels lettered A to N from top to bottom (skipping G and I). Bays 1 and 2, which house the bulk of the processor logic, each have an indicator panel at the top with the remainder of the bay occupied by eleven standard logic panels mounted behind double doors. Each mounting panel can hold up to 25 DEC system plug-in modules numbered from left to right when viewed from the front. At the center of bays 3 and 4, which hold the console, is the main operator control panel. Usually, a tape reader is mounted in the left console bay just above the control panel and a paper tape punch at the top (the front panel of the drawer containing the punch has an opening for removing fan-folded tape). At the top of bay 4 behind a metal cover are the marginal check controls and an indicator panel for the standard in-out equipment. The remaining space above the control panel may be used for DECtapes, displays, or other equipment. The space below the console table can hold up to eight logic mounting panels, two of which are used for the arithmetic processor; the remainder may be used to hold the control logic for some of the in-out equipment. Inside the double doors at the back of each bay is an inner plenum door, on which are mounted the required power supplies and power control panels.



Arithmetic Processor Type 166

Physical dimensions are as follows:

Arithmetic Processor

Height	69-1/2 inches
Width	100 inches
Depth	60 inches (75 inches with rear plenum doors open)
Weight	1300 pounds including tape reader and punch

Keyboard-Printer, Teletype Model 35 KSR

Height	38-1/2 inches
Width	20 inches
Depth	24 inches
Weight	151 pounds

Card Reader, Burroughs B122

Height	50 inches
Width	48 inches
Depth	29 inches
Weight	200 pounds

Intake fans at the bottom of every bay cool the logic modules by blowing air out between them. All equipment described in this manual can operate in an ambient temperature range from 50° to 100°F. However, if the installation includes temperature sensitive equipment such as magnetic tape, air conditioning is required. The floor should be capable of supporting approximately 150 pounds per square foot.

1.3 ELECTRICAL CHARACTERISTICS

All PDP-6 equipment uses standard line power at 105 to 125 vac, 60 cycles, single phase. All power cables use Hubbell Twist-Lok connectors; both cable and connector are rated at 30 amperes. The arithmetic processor, console and console-mounted standard in-out equipment together use two lines and two power controls. The main power control is usually a Type 829 or Type 835; it provides ac to all of the power supplies for the processor logic and any in-out control logic mounted below the console operator panel. The dc voltages required by the logic are +10 and -15 volts. Some power supplies provide both, others provide only the negative voltage. In some cases, two -15 volt supplies may be connected in series to provide - 30 volts to solenoid drivers for in-out equipment. One -15 volt line turns on a secondary power control (Type 811 or 834) that provides ac to the motors for the reader, punch, and keyboard-printer. For the punch, ac is fed through a Type 823 Power Control (mounted directly on the punch) that allows the processor logic to control application of punch motor power. Another -15 volt signal is applied to external power controls (usually Type 811 or 834) via the in-out bus to turn on the peripheral equipment. Still another -15 volt signal turns on the power controls (usually Type 836) in the memories. This last dc turnon signal is not sent via the memory bus; instead it is included in a small bus that also carries marginal check voltages from a variable power supply located in the console.

Current consumption of the equipment described in this manual is as follows:

Arithmetic Processor, including console and console-mounted in-out logic	25 amperes, 1900 watts Turnon surge, 40 amperes
Tape Reader	1.8 amperes, 150 watts Turnon surge, 2.8 amperes
Tape Punch	1.85 amperes, 65 watts Turnon surge, 9 amperes
Keyboard-Printer	2.6 amperes, 140 watts Turnon surge, 7 amperes
Card Reader	1.5 amperes, 145 watts Turnon surge, 7 amperes

All PDP-6 logic is solid state; transistors and diodes operate on static logic levels of 0 and - 3 vdc (tolerances are 0 to -.3 volts and - 2.5 to - 3.5 volts). Most logic modules include an internal supply to derive the negative logic level from the -15 volt input. PDP-6 logic uses pulse timing almost exclusively. Pulses are of either polarity depending upon gate input requirements. Pulse amplitude is 2.5 volts from ground with tolerances of +2.3 to +3.0 volts and - 2.3 to - 3.5 volts. Pulses at inverter outputs may be from ground to - 3 volts or vice versa. Pulse widths may be 1 microsecond or 400, 70, 40, or 25 nanoseconds depending upon module type and application. Occasionally, an input may be triggered by a level transition instead of a pulse.

CHAPTER 2

SYSTEM FUNCTION

The logical configuration of the Arithmetic Processor Type 166 is shown in Figure 2-1. Large blocks at the top and bottom represent the buses that connect the processor to the input-output equipment and the memory; the figure shows all connections to these buses, both data and control (each connection is labeled with the number of physical lines required). Between the buses is a block diagram of the processor showing all registers, with transfers among them represented by lines connecting the register blocks. Each block is labeled with both the name of the register and the number of bits. The registers vary considerably in size. Data registers have 36 bits, those that handle only addresses have 18. Registers that handle floating-point exponents have 9 bits; those that control memory protection and relocation have 8; 7-bit registers govern the requesting and granting of program sequence breaks through the priority interrupt system. The figure does not show the control lines within the processor, but all control pulses for each register are written beside the corresponding block. For an explanation of signal names, refer to the discussion of signal notation in Chapter 4.

The heart of the arithmetic processor is a set of three full-size registers which handle all data transfers and in which are performed all logical and arithmetic operations. These are arithmetic register AR, multiplier-quotient register MQ, and memory buffer MB. All transfers between processor and memory are made through MB, transfers between processor and peripheral equipment are made via AR. At the console, the operator may communicate with the system through a register of data switches for sending information in via AR and a register of memory indicators for displaying memory words via MB. MB takes part in all data transfers, but in logical and arithmetic operations it usually plays a passive role by holding an operand which is combined with the operand in AR, the result appearing in AR. MQ serves primarily as an extension of AR for handling double-length operands.

The processor performs a program by executing instructions retrieved from consecutive memory locations as counted by the program counter PC, although the program may change its own sequence by changing the address in PC. To gain access to memory for retrieval or storage, the processor requests a memory cycle and supplies an address over the bus from memory address register MA. This register also serves as a control link to the operator in that the system receives addresses via MA from the address switch register on the console. When a word is retrieved at MB as an instruction, its left half passes to the instruction register IR which controls further retrieval of index registers and accumulators, and which is decoded to govern the actual execution of the instruction. The right half in MB is the memory operand address, which may be modified in AR by the contents of an index register.

The operands are brought from memory to AR, MB, and sometimes MQ, for whatever operations are necessary for the execution of the instruction. In some of the more complicated operations, these full-size registers are aided by the shift counter SC and the floating-exponent register FE. SC is used for subsidiary computations such as the calculation of the exponent in floating-point arithmetic, and it also controls the count of any operation performed by the repetition of basic steps in the three main registers. FE is used only for temporary storage of preliminary results while SC is controlling the remainder of the operation.

Besides the registers that enter into the regular execution of the program and its instructions, the processor contains an executive system and a priority interrupt system. The first contains two registers for memory protection and relocation. When the processor is in executive mode, all instructions and all memory are available to the program. In user mode, a number of programs share processor time with each program restricted to a specific area in core, and certain instructions are illegal. All programs are written using the lower addresses but these are not supplied directly to memory from MA. In requests for memory access, the address in MA is compared with the contents of the memory protection register PR. The number in this register defines the size of the block available to the program and prevents it from addressing any location outside its assigned area. The address is then changed to one within the assigned area by adding a constant contained in the relocation register RLR to the address in MA.

The priority interrupt system allows peripheral devices and certain conditions internal to the processor to interrupt the normal program sequence. There are seven interrupt channels through which sequence breaks are allowed on a priority basis as governed by three control registers. The first register allows the program to turn individual channels on and off; the second synchronizes break requests to internal processor timing and assigns the break to the highest priority channel that has been recognized; the third holds the break and prevents further interruption by lower priority channels. A break is executed by performing the instruction in a particular memory location associated with each channel. The assignment of channels to devices is entirely under program control; the program may assign several devices to a single channel or give a device no assignment. One of the devices to which the program may assign a channel is the processor itself. For this purpose the processor has an I/O interface containing a number of flags that allow internal conditions to interrupt the sequence; the flags may be sensed and controlled by the program. Through this interface, the processor may also bring information in from the console DATA switches or supply memory protection and relocation information to the executive system.

Timing for all operations in the processor is supplied by asynchronous pulse chains. Processor operation is initiated by means of a special key cycle that supplies timing for events associated with operator intervention at the console and provides entry into the main sequence. When the processor is running, timing is supplied by the main sequence which is repeated for each instruction. The main sequence uses a hierarchy of other sequences—built-in hardware subroutines—which can be called directly by the main sequence or by any sequence of higher rank. Thus the processor operates using many levels of nested sequences; each sequence stops upon calling a lower ranked sequence and restarts upon return from it (although the restart need not be at the point of departure). For example, in a block transfer, the main sequence calls the block transfer subroutine which in turn calls others to perform the necessary arithmetic and obtain memory access.

2.1 PROGRAMMING

The first 16 locations in memory function as accumulators, index registers, or ordinary memory locations. Their particular functions are determined entirely by the processor under program control, but they differ from the remainder of the memory system only in that they are usually contained in a fast memory. All 16 locations may be used as accumulators or ordinary memory, but only locations 1 to 17 may be addressed as index registers because a zero index register address specifies no indexing. Since ordinary memory addresses are 18 bits, only the information contained in the right half of an index register is actually used for address modification.

In systems that include a fast memory, it replaces the first 16 core locations (which normally hold a readin loader) for normal processor operations—operations in the reading area can be initiated only from the console; and once an instruction has been taken from outside this area, it again becomes inaccessible to the program. In all systems locations 40 and 41 are used for programmed operators, 42 to 57 are used by the priority interrupt system—a programmer should be wary of using these locations for other purposes.

The logic descriptions contained in this manual assume that the reader is completely familiar with the processor instructions, all of which are described in detail in PDP-6 Programming (K-06). That manual describes the instructions in terms of elements available to the program, i.e., by their effect on accumulators, memory locations, flags, and control registers. For convenience Table 4-1 lists the mnemonic and octal codes for all instructions. The remainder of this section describes the Type 166 number system, instruction format, and flags.

a Number System

For arithmetic computations the hardware is capable of handling numbers in two formats, fixed point and floating point. Both formats use the full 36-bit word; bits are numbered 0 to 35 from left to right. In all numbers, bit 0 represents the sign, 0 for positive, 1 for negative. In floating point, bits 1-8 represent an exponent, bits 9-35 represent a fraction. In fixed point, bits 1-35 represent magnitude, which is usually interpreted as a full fraction with the binary point between sign and magnitude or as an integer with the binary point to the right of bit 35. Of course, the assumed position of the point has no effect on the processor and a program may adopt any consistent point convention. However, the fixed-point hardware does include special provisions to facilitate handling of integers in multiplication and division because these operations make use of double-length numbers.

In ordinary arithmetic, the negative of a number is usually formed merely by changing the sign. This notation is inconvenient for a machine so hardware arithmetic represents negatives by 1's and 2's complements. If x is an n-digit binary number, its 2's complement is $2^n - x$ and its 1's complement is $(2^n - 1) - x$ or equivalently $(2^n - x) - 1$. Subtracting a number from $2^n - 1$ (i.e., from all ones) is precisely equivalent to performing the logical complement, i.e., changing all zeros to ones and all ones to zeros. Therefore, to form the 1's complement, the processor uses the logical complement—usually referred to merely as the complement—and to form the 2's complement it complements and adds one to the result.

In 1's complement notation, one can read a negative number by attaching significance to the zeros instead of the ones. For 2's complement notation, this simple interpretation is not possible because adding one to the logical complement changes at least the final bit and in fact changes bits as far as the carry propagates. Thus in 2's complement notation, one can read a negative number by attaching significance to the rightmost 1 and attaching significance to the zeros to the left of it. A 2's complement system has the following characteristics. A number all ones represents -1. All even numbers both positive and negative end in 0. In a negative integer, ones may be discarded at the left. In a negative fraction, zeros may be discarded at the right; as long as only zeros are discarded, the number remains in 2's complement form because it still has a 1 which possesses significance. However if a portion including the rightmost 1 is discarded, the remaining part of the fraction is now a 1's complement.

In the Type 166, the 2's complement is used to represent negatives for both fixed and floating numbers. In a positive fixed-point number, the sign bit is 0 and bits 1-35 represent magnitude in normal binary fashion. In a negative, the sign is 1 and the remainder of the word contains the 2's complement of the magnitude of the corresponding positive number. Since 0 is considered one of the positive numbers, the magnitude of the largest positive number is one less than the magnitude of the largest negative number. Fixed-point integers thus have a range from -2^{35} to $2^{35} - 1$; for fractions, the range is -1 to $1 - 2^{-35}$.

The floating-point hardware interprets a computer word as containing an 8-bit exponent and a 27-bit fraction. For a positive number, the sign is 0, as before; but the contents of bits 9-35 are now interpreted only as a binary fraction and the contents of bits 1-8 are interpreted as an integral exponent in excess 128 (200_8) code, i.e., exponents from -128 to +127 are represented by the binary equivalents of 0 to 255. Floating-point zero and negatives are represented in exactly the same way as fixed point: zero by a word containing all zeros, a negative by the 2's complement. The negative thus has 1 for its sign and the 2's complement of the fraction, but since every fraction must contain a 1 unless the entire number is 0 (see below), it has the 1's complement of the exponent code in bits 1-8. Since the exponent is in excess 128 code, an actual exponent x is represented in a positive number by x + 128, in a negative number by 127 - x. The program, however, need not concern itself with these representations because the hardware compensates automatically. For example, for the instruction that scales the exponent without affecting the fraction, the hardware interprets the scale factor in standard 2's complement form but produces the correct 1's complement result for the exponent.

In all floating-point operations, the hardware assumes that all nonzero operands are normalized and always normalizes a nonzero result. Floating-point numbers are considered to be normalized if the magnitude of the fraction is greater than or equal to 1/2 and less than 1. The test for normalization is thus that either the sign bit differs from bit 9 or bits 9-35 contain $400\,000\,000_8$, the latter being required for the special case of the fraction -1/2, in which bits 9 and 0 are equal. Floating-point numbers thus have a fractional range in magnitude from 1/2 to $1-2^{-27}$ and an exponent range of -128 to +127. Note that the signed fractional part -1 (i.e., a 1 in bit 0 and all zeros in bits 9-35) satisfies the test for normalization but the hardware always changes it to -1/2 and adjusts the exponent appropriately. The hardware may not give the correct result if the program supplies an operand that is not normalized or that has a zero fraction with a nonzero exponent.

The characteristics of 2's complement notation require additional precautions in floating-point operations and fixed-point fractional multiplication because these have double-length results. The programmer must remember that discarding the low-order part of a double-length negative leaves the high-order part in correct 2's complement form only if the low-order part is null. In floating point, the programmer may request rounding, which automatically restores the high-order part to 2's complement form if it is negative.

b Instruction Format

All but the input-output instructions and programmed operators use a basic format with bit assignment as follows:

0-8		Instruction code
9-12		Accumulator address
13	I	Indirect bit
14-17	X	Index register address
18-35	Y	Memory address

Bits 0-8 determine which operations are executed for the instruction. Bits 9-12 and 14-17 each address the first 16 memory locations which serve as accumulators and index registers. On some occasions, bits 9-12 are used for control purposes instead of addressing an accumulator, for example to address flags. The effective address E of an instruction depends on the values of I, X, and Y. The contents of index register X (zero X specifies no indexing) are added to Y to produce an address. If I is 0, this address is used as the effective address; if I is 1, this address is indirect and is used to retrieve another word. The new word is processed in exactly the same manner as above, i.e., X and Y are used to determine the effective address if I is 0; otherwise, they are used to retrieve another word. The process continues until a word is found in which I is 0. This calculation using I, X, and Y is carried out for all instructions even when E is to be used as an operand or control information instead of a memory address.

IOT instructions (designated by three ones in bits 0-2) have the following bit assignment:

0-2		111
3-9		Device code
10-12		Instruction code
13	I	Indirect bit
14-17	X	Index register address
18-35	Y	Memory address

Bits 3-9 address an I/O device out of a possible 128, bits 10-12 specify one of eight IOT instructions; the processor and the priority interrupt system are considered devices. As in the basic format, I, X, and Y are used to calculate E, which is used as an address in some cases, as control information in others.

A programmed operator is designated by three zeros in bits 0-2. Whenever such an operator appears in the program, the processor calculates an effective address from bits 13-35 of the instruction word in the usual manner but it does no further decoding; instead it stores the contents of the instruction register in the left half of location 40 and the calculated effective address in the right half, and then executes the instruction contained in location 41 (which is usually a JSR to an appropriate subroutine).

c Program Flags

The processor contains a number of flags that may be sensed by the program. Flags that are set automatically, e.g., by error conditions, usually cannot be set by the program; whereas, flags that allow the program to enable specific operations can always be both set and cleared. Some flags are governed primarily by jump instructions but most are contained in the processor I/O interface and are governed by IOT instructions. Any flag listed as being able to cause a priority interrupt does so on the channel assigned to the processor provided the priority interrupt system is active.

AR CRY0, AR CRY1

These flags are set by carries from the corresponding bits in AR. They are useful primarily for double-precision arithmetic and in correcting a result that has overflowed.

AROV
AROV ENABLE

The overflow flag may be set by arithmetic operations in a variety of instructions. It indicates a loss of information, an incorrect result of a computation, or failure of the processor to perform a computation. Setting OV causes a priority interrupt if the enable flag has been set by the program.

PC CHG
PC CHG ENABLE

The PC change flag is set when the program sequence is changed by a skip or jump instruction. Setting PC CHG causes a priority interrupt if the enable flag has been set by the program.

PDL OV

The flag is set and triggers a priority interrupt when a pushdown or pullout instruction has gone outside of the core area assigned to the pushdown list.

NON EXIST MEM

This flag is set and triggers a priority interrupt when the memory system fails to respond to a memory request.

CLOCK
CLOCK ENABLE

The clock flag is set every sixtieth of a second by a signal from the main power control. It causes a priority interrupt if the enable flag has been set by the program.

USER

As a flag, the sole function of USER is to indicate to the executive routine whether a user program was interrupted, either by a priority interrupt or UUO (the executive routine must service all priority interrupts) or by the trapping of an illegal instruction (instructions that are illegal during a user program are a JRST that attempts to dismiss an interrupt or halt the computer, and any IOT). The execution of a JSR during a PI cycle or following a UUO or the trapping of an illegal instruction clears USER.

As a control flip-flop, USER implements the restrictions on user programs. Thus in order to restrict the operation of a user program, the executive routine must set USER when it transfers control to the program. If the sole purpose of an interrupt is to service a block IOT and there is no overflow, USER stays set and control automatically reverts to the user program after the IOT.

ILLEG OP

An attempt by a user program to address a location outside of its restricted area in core sets this flag causing a priority interrupt. At the time of the interrupt, PC may point either to the location of the instruction which tried to use the address or to the location following this instruction.

Some subroutine-calling jump instructions store what is referred to in the logic as "miscellaneous bits" in the left half of the location that receives the program count. In returning from the subroutine, the program may use a jump that restores the bits to their original states. Included in the miscellaneous bits are the two carry flags, overflow, PC change, user, and a control bit that is used in a special case for returning from a priority interrupt. The four byte manipulation instructions that load or deposit a character require two main sequences for their execution, and a priority interrupt can occur between them. The first part fetches and, if necessary increments the pointer; the second operates on the byte. If the program jumps to a subroutine for an interrupt that occurs between the two parts, bit 4 is set in the PC store location. Then in the subroutine, the program may determine whether a character operation was interrupted; and upon the return, the stored bit ensures that the interrupted instruction, which must be restarted, will not reincrement the pointer.

2.2 MAIN SEQUENCE

This section is devoted primarily to the manner in which the processor main sequence executes an instruction and sequences the program, but it also treats the control elements that allow entry into the main sequence from the console and the executive system which controls the sharing of processor time by user programs.

a Console Control

Operator control over the processor is exercised through two types of logical inputs associated with the keys and switches. Inputs from the switches are control levels that may provide data or addresses for use by the processor or gates to govern specific processor events. The keys are momentary contact switches that trigger specific events or initiate sequences although the level output of a key may also be used as a gate for events associated with the key action. The complete effect upon the computer of all keys and switches is described in detail in Chapter 3, Operation; we are concerned here only with the way in which the keys affect processor operation, in particular the main sequence.

The logic associated with the console keys consists primarily of a key cycle time chain and a control flip-flop RUN. Normal processor operation is initiated by triggering the main sequence and setting RUN—the 1 state of this flip-flop allows the completion of each main sequence to trigger the next so that the processor executes one instruction after another. Whenever RUN is cleared either from the console or by the program, operation ceases at the end of the current main sequence. The stop keys can, of course, affect the computer while it is running, but most keys that initiate events cannot; only the initiating keys trigger the key cycle. For those key functions that make use of the main sequence, the key cycle performs the necessary preliminary operations, such as transferring information in from the console data and address switches; but for those functions that do not use the main sequence, the key cycle controls the entire operation.

The operator may place the processor in normal operation by means of the START, READ IN, and INSTRUCTION CONTINUE keys. For these functions, the key cycle sets RUN and triggers the main sequence. The INSTRUCTION STOP key halts the processor at the end of the current main sequence by clearing RUN. The processor may also be stopped at the end of any memory access by means of the MEMORY STOP key, which disables the return from the memory subroutine to the waiting sequence. In this case, the processor is still "running" and normal operation may be resumed through the MEMORY CONTINUE key which simulate a memory subroutine return. The operator may also deposit information in the memory location addressed by the ADDRESS switches or examine the contents of that location while the processor is running. For these two console functions, a single key cycle is merely inserted between two main sequences. For the remaining functions, the processor cannot be running, i.e., RUN must be 0. The EXECUTE key causes the processor to execute as an instruction the word contained in the DATA switches. For this instruction, the key cycle triggers the main sequence but does not set RUN, so the processor stops when the instruction is complete. There are also two keys that allow the operator to examine or deposit information into a sequence of consecutive memory locations without addressing them individually. Each such examine or deposit requires a key cycle and these functions cannot be performed unless the processor is stopped.

For maintenance purposes, the console has a REPEAT switch. When this switch is on, any key function can be repeated at a rate determined by a pair of speed controls. The logic enables this by having the key cycle retrigger itself through a delay whose interval is determined by the speed setting.

b Instruction Execution

Most instructions are executed by the five cycles that comprise the main sequence: instruction, address, fetch, execute, and store. Each main sequence begins when the instruction cycle requests memory access to retrieve an instruction from the location specified by the program counter. Upon receiving the instruction, the processor enters the address cycle and performs the effective address calculation as outlined in 2.1b. If an address is indirect, a new address word is retrieved from memory and the cycle begins again. After repeating the cycle as many times as is necessary to produce the effective address, the processor goes on to the fetch cycle to retrieve the necessary operands. If an accumulator is specified, it is retrieved first and sent to AR. If the instruction uses a double-length operand, a second word is fetched from the next consecutive accumulator (with location 0 being taken as following location 17) and sent to MQ. In some instructions, an extra word must be retrieved from the memory location addressed by either the right or left half of the addressed accumulator. This type of operand is also sent to MQ. Finally the processor fetches the memory operand as specified by the effective address and leaves it in MB. This last fetch is skipped if E is to be used as control information, an operand, or a jump address.

After fetching the operands, the processor enters the execute cycle in which it performs whatever logical, arithmetic, or control functions are necessary to carry out the instruction. This cycle also increments the program counter by one so that it points to the next instruction in normal sequence. If a jump or skip is being performed, PC is changed following the count. Finally the processor enters the store cycle to deposit the result, which is usually contained in AR. For most instructions, the result may be deposited in an accumulator, in memory or in both as specified by the instruction; for a double-length result, AR and MQ may be stored in consecutive accumulators. The processor then returns to a new instruction cycle.

Although most instructions are performed by the sequence outlined above, there are many that are performed by variations of it. The more complicated instructions are performed by special sequences that are entered from the execute cycle and usually return to the store cycle. Sometimes a special sequence handles the storage itself and returns directly to the instruction cycle. Other instructions must first fetch and operate on a pointer that provides information necessary for the retrieval of the true operand; such instructions require in effect two main sequences. A block transfer repeats the fetch and execute cycles once for every word in the block. Whenever the execute cycle occurs more than once for a single instruction, the incrementing of the program counter is inhibited in all but the final occurrence. In this way, PC points to the next instruction only when the current one is bound to be completed before any interruption can occur.

The actual form of the sequence and the operations carried out in it are determined entirely by the instruction code as decoded from the instruction register. The codes are divided into eightclasses according to the configuration of bits 0–2. If these bits contain 111, the instruction is in the special IOT format and IOT control decodes bits 10–12 to determine which of eight instructions is specified. If bits 0–2 are 000, the instruction is taken to be a programmed operator—there is no further decoding and the processor enters a special sequence from which a subroutine must properly interpret the remainder of the code (and of the instruction word for that matter). In the other six classes the remaining six bits are decoded by the hardware, primarily by the logic associated with IR. They may be decoded in a variety of ways depending upon the instruction class. Occasionally, single bits are used to represent specific operations, such as specifying the left or right half in a half-word transfer or whether fixed-point multiply is to interpret the operand as an integer or a fraction. In other cases, groups of bits are decoded; for example in the Boolean class four bits determine which of 16 Boolean functions is specified, the other two determine the mode of execution. In some cases, all six bits are decoded to a single control level for an individual instruction that has no modes.

There are some instruction codes that are not used and are executed as no-ops; the unused octal codes are those for which no mnemonic is listed in Table A4-1. Since most instruction codes are divided into sets of bits that are decoded in different ways, it is possible for some combinations of mode and instruction to have no effect on the state of the computer and these may be considered as no-ops. An obvious example is a full-word transfer that does not change the operand and is performed in the self mode.

The way in which instructions are executed is also influenced by the requirements of the priority interrupt system and the executive system. The interrupt channels are strobed at the beginning of every instruction and address cycle; and if a request is discovered, the processor honors it by entering a special PI cycle in which it executes the instruction in the location corresponding to the channel being serviced. For a PI cycle, the processor starts a new main sequence and executes it in the normal fashion except that the address supplied to MA for instruction retrieval comes from a channel address encoder in the PI system rather than from PC, and the strobe is disabled so that the PI cycle cannot itself be interrupted. The instruction executed in a PI cycle must either do an I/O data transfer or transfer control to a subroutine for further service. If the data transfer requires no further service, the processor automatically returns to the interrupted instruction; if further action is required, a second PI cycle is executed so that control can be transferred to a subroutine before honoring any other interrupt. If control is transferred to a subroutine, the interrupt is "held" so that the processor may again be interrupted but only on a channel of priority higher than the one being held; the subroutine is responsible for releasing the interrupt upon completion.

The executive system restricts processor operation in order to permit time sharing by several programs. When running restricted (user mode), each program must operate within the area of core assigned to it; an attempt to use an address outside of the assigned area causes a flag to be set and immediately initiates an interrupt on the processor channel (the location to which PC points depends upon the time within the main sequence that the illegal memory request was made). The execution of a programmed operator (UUO) is unrestricted, but the locations used by UUOs (40 and 41) are inaccessible to user programs; UUOs executed by user programs always transfer control to the (unrestricted) routine responsible for overall system operation. Besides restricting addresses, the user mode traps (as if they were UUOs) attempt to halt the processor, dismiss an interrupt channel, or operate an I/O device. Instructions executed in PI cycles are unrestricted even if the interrupted program was running in user mode.

c Executive System

The executive system includes the 8-bit memory protection and relocation registers PR and RLR, nets that monitor user instructions, and the user flag. These logic elements allow the processor to be run in a restricted mode to permit time sharing of several user programs. A program that runs unrestricted (the executive routine) must be responsible for overall system operation. The executive routine is responsible for scheduling user programs (assigning core areas, entering user mode, and transferring control to the current program, interrupting when its time is up), for servicing all interrupts and UUOs, for servicing all I/O needs of user programs, and for taking action when it receives control because a user program attempted to use an illegal address or instruction or gave up control through a UUO.

Following power turnon, the processor is automatically in executive (unrestricted) mode, and when it is not running, the operator may place it in executive mode by pressing the I/O RESET key (this action also clears the I/O equipment). During a priority interrupt cycle, the processor runs unrestricted; but if a user program is interrupted, the user flag (which normally implements the user restrictions) remains set. Thus unless one instruction suffices to service an interrupt, the executive routine must within a PI cycle clear the user flag to return the processor to the executive mode and transfer control to one of its subroutines. Similarly, since all UUOs are under executive routine control, the instruction in location 41 must be a JSR, which stores and clears the user flag. The executive routine enters a user program by means of a jump which sets the user flag (JRST with a 1 in bit 12). The return to a user program after an interrupt or UUO may be made by means of a restoring JRST (a 1 in bit 11). This instruction restores all other flags to their original states and can set the flag but can never clear it. This prevents a user program from leaving user mode as a result of an incorrect restoring JRST.

Each user program is assigned a block in core whose first location is an integral multiple of 2000 octal (since the executive routine must use locations 40 to 57 to service UUOs and interrupts, 2000 is the lowest first address available for a user block); the block size is also an integral multiple of 2000. A user program is restricted to addresses from zero to one less than its block size; if it attempts to use an address equal to or greater than its block size, the illegal operation flag is set and an interrupt occurs immediately on the processor channel. To assign a core area to a program, the executive routine uses a processor DATAO, which loads PR and RLR, respectively, from bits 0-7 and 18-25 of the data word. Each time the memory subroutine is called during a user program, the executive system tests for an illegal address by checking that the address does not exceed C(PR) x 2000 + 1777; the size of the block is equal to [C(PR) + 1] x 2000. At the same time the user address is relocated by adding the block starting address to it; i.e., the address sent out on the memory bus is equal to C(RLR) x 2000 + C(MA). Addresses 0 to 17 are never relocated, so all programs have access to fast memory (note that this means that no user program ever uses the first 16 core locations in its assigned block).

The user flag implements the restrictions on a user program by enabling the relocation and protection circuits and enabling the nets that monitor user instructions. A user program may not use a JRST with a 1 in bit 9 or 10 (an attempt to dismiss an interrupt channel or halt the processor) nor any IOT. These instructions are trapped by having their IR decoder outputs drive the UUO command line when the processor is in user mode. As mentioned above, UUOs are unrestricted, i.e., unrelocated location 41 is executed. Thus in user mode, an illegal instruction is executed as if it were a UUO and thereby returns control to the executive routine.

2.3 ARITHMETIC LOGIC

The arithmetic part of the processor includes the three full-size registers AR, MB, and MQ, the two 9-bit registers SC and FE, the time chains that execute the special sequence instructions and subroutines, and a subroutine interface through which connections are made from the special time chains to the gating for the three main registers. Included in the AR part of the logic are four flags, AROV, AR CRY0, AR CRY1, and PC CHG. The states of these flags are stored as miscellaneous bits and may be restored by a JRST; they may also be sensed for a jump and cleared by a JFCL.

Transfers of full words or half words may be made between MB and AR, transfers of full words between MB and MQ. MB may also receive PC, IR, or the miscellaneous bits for storage in a UUO and in certain jumps. The two halves of a word can also be interchanged (swapped) in MB. Although the AND function of MB and AR can be formed in MB, it usually plays a passive roll in logical and arithmetic operations by holding an operand which is combined with an operand in AR. Associated with AR is a myriad of gates that implement the clearing or setting of individual bits in a word according to a mask, the formation of the complement, OR, AND, and exclusive OR logic functions, and the shifting of bits left or right. There is also a carry function which can be triggered at any point in the register and produces an arithmetic carry to the left; i.e., it complements the first bit, complements the second if the first changes from 1 to 0, and ripples to the left in this manner until it complements a 0 bit. If this carry chain is triggered only at the register LSB, it adds 1 to the number represented by the contents of the register. Some instructions use the left and right halves of a word to hold a word count and an address; in order to allow indexing of both half words simultaneously, the carry chain can be triggered at AR17 and AR35. Although this is used as two simultaneous index functions, there is no break in the carry chain and an overflow from the right half can carry into the left: hence the pair of index functions effectively adds 1000001 to AR. The above listed functions are the only ones that can be performed directly—all others are executed by combinations of them. If following an exclusive OR, the carry function is triggered at a number of places in the register (the particular places being determined according to the addition algorithm by the previous configuration of the words in MB and AR), it generates the algebraic sum in AR of the numbers originally in MB and AR. Negation (which always means arithmetic negation) is performed by complementing and adding 1. In subtraction, the number in MB must be subtracted from that in AR: for fixed point, the processor performs subtraction by complementing AR, then adding and complementing the result; for floating point, MB and AR are switched and the subtrahend in AR is then negated so the result can be produced merely by adding. Multiplication or division is a sequence of shifts with additions or subtractions interspersed.

The third register, MQ, is used occasionally for temporary storage and there is a special case in character operations where AR and MQ are shifted in parallel for control purposes, but MQ serves primarily as a right extension of AR for handling double-length operands. For actions on a pair of accumulators, the two registers are joined end to end and the double-length operand may be shifted in either direction. Moreover, the opposite ends of both registers may be joined to form a ring and the contents rotated in either direction. In multiplication, the multiplicand comes from AC and the multiplier is either C(E) or E, but when performing the actual arithmetic operation, MB holds the multiplicand and the multiplier in MQ controls the formation of partial products in AR. As bits of the multiplier are used and shifted out of MQ, the low-order bits of the double-length product are shifted in. In division, MQ holds the low-order half of the double-length dividend and as bits are shifted out to AR for use by the division steps, bits of the quotient are shifted in at the least significant end. At the completion of the computation, MQ contains the quotient and AR the remainder, but the divide subroutine then switches their positions so the quotient can be stored in AC.

In floating-point operations, the exponent is first calculated in SC, whose gating provides addition and indexing. In floating scale, the only operations performed are on the exponents. For other floating-point instructions, the exponent is calculated in SC and then stored in FE while SC is used to count the steps in the fixed-point part. Following computations, the exponent is transferred back to SC in case it must be changed while normalizing the result, and finally from SC it is inserted in the exponent part of AR. SC is also used to calculate the position portion of a pointer for a character operation that increments, and from SC the new position is inserted in the pointer in AR.

In addition to the registers, the arithmetic logic also includes the time chains and many control nets for executing the special sequences and subroutines. The basic subroutines, which can be called from any higher level, are the AR subroutine group (which includes fixed-point addition, subtraction, negation, and indexing in either direction), the SC addition subroutine, and the SC shift-count subroutine which simultaneously counts SC and shifts AR and/or MQ (for shift instructions both registers are shifted even though for a single operand only AR contains information). For fixed add and subtract, the execute cycle calls the AR subroutines directly. For other instructions, the processor switches from the execute cycle to a special sequence which calls the lower rank subroutines and which usually returns to the store cycle.

The sequences for character operations, block transfer, shift operations, and floating scale call only the basic subroutines (including the memory subroutine). Floating multiply and divide begin by calling the exponent calculate subroutine, then the multiply or divide subroutine whichever is appropriate, and both terminate by entering the normalize return subroutine which also follows the floating add-subtract sequence. The fixed multiply sequence calls only the multiply subroutine; fixed divide does not make use of an intermediate special sequence but instead enters directly into the divide subroutine.

2.4 MEMORY INTERFACE

The interface that connects the processor to the memory bus includes the memory address register MA, memory buffer MB, user mode registers PR and RLR, memory indicator register MI, and the control logic for the memory subroutine. A processor cycle or special sequence gains access to memory by triggering the memory subroutine, which has entries for read, write, and read-pause-write which must later be followed by a read-write restart. The calling sequence must also supply an address to MA, and if information is to be written, a word to MB. If the processor is in executive mode, the subroutine places the appropriate request levels on the bus immediately, but for user mode there is a delay while the address in MA is compared with PR. An illegal address causes the processor to go to the end of the current main sequence and sets the illegal operation flag requesting an interrupt on the processor channel.

While the comparison against PR is being made, the outputs of RLR and the more significant MA bits are applied to a set of dc adders whose outputs represent the sum of the two registers. If the address in MA is legal, memory control puts the relocated address on the bus (low-order bits are supplied directly from MA, high-order bits from the relocation adders). In the address as received by memory, MA34 supplies the least-significant bit of the address within a single memory and bit 35 is used as the LSB to select the bank. In this way, consecutive addresses are interleaved—all odd addresses in one bank, all even in another. A switch at the memory allows the operator to disable this feature when using a 16K bank (with 8K banks, addresses must be interleaved).

The processor memory subroutine requests a memory cycle by calling memory as a subroutine, and it must wait until the addressed memory accepts the request, which does not occur until the memory is free and this processor has priority. The processor restarts upon receipt of an acknowledgement signal from memory. If the request is for a write cycle, the processor need wait only until the memory accepts the word in its own buffer; but for a read cycle, it must wait until it receives the information read from the memory location. If the request is made to fast memory, the write takes slightly longer than the read because there is no buffer. If the request is not acknowledged within a considerable time compared to a memory cycle, the nonexistent memory flag is set, requesting an interrupt on the processor channel. Following the acknowledgement signal, the memory subroutine sends a restart pulse to the waiting sequence unless the MEMORY STOP key is on. To restart the processor after a memory stop, the operator must simulate the return to the waiting sequence by pressing the MEMORY CONTINUE key.

If the address in MA is the same as that in the console ADDRESS switches or the operator is examining or depositing information from the console, the contents of the memory buffer are displayed in the memory indicators. On a read MI displays the information read, on a write it displays the information to be written.

2.5 INPUT-OUTPUT SYSTEM

At the processor end of the I/O bus is the in-out transfer control logic that times the transfer of data, initial conditions, and status over the bus by sending command signals (also over the bus) to the device control units. Two of the devices on the bus are the priority interrupt system located in the processor, and the processor itself whose I/O interface contains a number of flags through which internal processor conditions can request priority interrupts and which allow the processor to check its own internal status with IOT instructions.

When the code 111 appears in bits 0-2 of the instruction register, the processor IOT control decodes bits 10-12 to determine the specific IOT instruction. Upon reaching the execute cycle, the processor switches to a special IOT sequence that times the instruction operations and generates the necessary command signals. Only four types of command signals are sent out on the bus; these are for DATAI, DATAO, CONO, and STATUS, of which the first three correspond to individual IOT instructions. BLKI or BLKO requires signals on the bus only after conversion to a DATAI or DATAO. CONI, CONSZ, and CONSO bring conditions in and the latter two then perform tests; all three generate the STATUS command and affect the peripheral equipment in exactly the same way.

While IOT control is generating the command signals, the device code from IR bits 3-9 is supplied over the bus to enable a gate in the device with that assigned code; signals are sent to all devices but only the selected device can respond. Data or initial conditions are supplied from AR over the bus to the selected device; data or status is supplied from the device over the bus to AR. Among the initial conditions that CONO may supply to a device is a priority interrupt assignment; CONO assigns a channel from 1 to 7 (zero is no assignment); and whenever the device requires service, it requests an interrupt by sending a request signal to the PI system on the bus line corresponding to its assigned channel. Every device except the PI system itself can receive at least one PI assignment.

The PI lines go to the priority interrupt system which contains three 7-bit registers, PIO, PIR, and PIH, to control the seven channels. A given channel is governed by one flip-flop from each register. The PIO flip-flop turns the channel on or off. The PIR flip-flop synchronizes the request to the processor main sequence and in conjunction with the remaining PIR flip-flops and a priority chain, generates an internal request signal for the channel that has priority. The PIH flip-flop holds a break on the selected channel. There are also three control flip-flops for the interrupt system, one that activates it, another that places the processor in a PI cycle, and a third that detects overflow from a block IOT performed in a PI cycle. By checking status, the program can determine whether the system is active and which channels are on. The processor strobes the PI lines at the beginning of every instruction and address cycle, and synchronizes a request signal from any PI line provided that the corresponding channel is on. If a PIR is on (a CONO can set the PIR for a channel even if the channel is off), the processor enters a PI cycle and starts a new main sequence which honors the request by performing the instruction in a particular memory location associated with the channel (if several PIR flip-flops are set, the lowest numbered channel has priority). To retrieve the instruction, MA receives the address from an encoder in the PI system. The encoder outputs are connected to MA in such a way that the channel number is doubled and added to 40. Thus for channel n, the processor executes the instruction in location 40 + 2n (the PI system uses locations 42 to 57, two for each channel). This instruction should be either a JSR to an appropriate subroutine or a block IOT to handle a data transfer. If it is a BLKI or BLKO and there is no overflow, the processor returns immediately to the interrupted program (another priority interrupt can occur before any instruction in the interrupted program is actually executed). If there is overflow, the processor goes into a second PI cycle in which it performs the instruction in location 41 + 2n, which should be a JSR to an appropriate subroutine. An instruction in a PI cycle should be either a BLKI, BLKO, or JSR; other instructions can be executed but they usually would have unfortunate consequences for the program and could even hang up the processor. If there is a jump to a subroutine, the break is held by setting the PIH flip-flop for the channel. This disables part of the priority chain so that the break routine can be interrupted only by a channel of higher priority. At the completion of the subroutine, the program should dismiss the channel so as to reenable all lower priority channels as well as the channel on which the break occurred.

Since a BLT may require considerable execution time, the PI request lines are also strobed following each word processed in the block. Whenever a request is discovered, the current source and destination addresses are stored in the accumulator and the partial block is terminated. The processor then begins a new main sequence as if to restart the block transfer, but is interrupted instead.

The I/O interface for the processor contains the flags discussed in 2.1c except for the four flags associated with AR and the user flag in the executive system. However two of the AR flags, overflow and PC change, can be set by the CONO that controls the interface flags and supplies it with a PI assignment, and these AR flags plus the user flag can be sensed as processor status (although the user flag is meaningless as status since it is 0 by definition whenever an IOT can be executed). The flag set by an illegal user address, a pushdown list overflow, or a request made to a nonexistent memory automatically requests an interrupt on the processor channel. Setting the clock, overflow, or PC change flag can cause an interrupt only if it has been enabled by a CONO. The program may also use data instructions for the processor: DATAO loads PR and RLR for a user program; DATAI brings in information from the console data switches.

Control units for other devices each contain a data buffer for transfers between the I/O bus and the device, an interface for control connections to the device, and an interface for control connections to the bus. The size of the buffer depends upon the device. It is 36 bits if full words can be transferred, but smaller if the transfers must be single characters. For an output device, the buffer can be loaded by DATAO; for an input device, DATAI gates the buffer onto the bus. The interface between the control unit and the bus includes a control register and a status register, which usually overlap and may be identical. For initial conditions, CONO can provide at least one PI assignment, place the device in operation, clear whatever error flags there may be, and often provide additional information such as determining the mode of operation or selecting an individual device from several that are connected to the same control unit. Usually all control bits can be examined as status and often additional status signals are supplied by the device. In most cases, the data instructions also perform certain control functions. For input, the loading of the buffer with information from the device usually sets a flag causing a priority interrupt. The processor responds with a DATAI that not only gates the buffer onto the bus but also clears the flag and initiates the retrieval of more information. For output, the transfer of information from the buffer to the device sets a flag causing an interrupt, and the processor responds with a DATAO that not only supplies new information but clears the flag and initiates the next transfer from buffer to device.

Included in this manual are the control units for four I/O devices. The paper tape reader has a 36-bit buffer but information may be retrieved in two modes. In alphanumeric mode, only one 8-bit line is read from the tape; in binary mode, the control unit accepts data from holes 1-6 only in lines in which hole 8 is punched, but it assembles six such characters into a 36-bit word. The punch handles only one character at a time but it still has two modes. In alphanumeric, it punches an 8-bit character; in binary, it punches a 6-bit character in holes 1-6, never punches hole 7, and always punches hole 8. The keyboard-printer is actually two independent devices with one PI assignment. For output, the processor prints single characters; for input, each character typed by the operator is placed separately on the bus. Characters typed at the keyboard are not printed unless the program sends them back out. With the card reader, only a CONO can initiate operations but only one CONO is required per card because once a card is started all 80 columns are read. The program can specify whether an interrupt shall be requested following each column or only when the buffer is full. In binary mode, all twelve holes of each column are read and three columns are assembled into a word. In alphanumeric mode, the Hollerith character in a column is translated into a 6-bit character and the control unit assembles six into a word. If an interrupt is requested but is not serviced before a new column is read, an error flag is set.

CHAPTER 3

OPERATION

This chapter discusses the normal operation of the arithmetic processor, reader, punch, Teletype, and card reader; some maintenance information is included, but the detailed discussion of operation for maintenance purposes is in Chapter 9. Although this chapter is relatively self-contained, it is recommended that the reader first familiarize himself with the functional organization of the equipment as presented in Chapter 2.

3.1 CONTROL PANELS

This section describes the function of the controls and indicators that are readily accessible to the operator; those mounted behind the doors of the bays are described in 9.1. All controls for normal operation of the processor, reader, and punch are on the main operator panel at the center of the console; this panel also contains most of the processor indicators. The panels at the top of bays 1 and 2 contain only indicators, most of which are for maintenance purposes. Indicators for the four in-out devices are on the upper part of the panel located behind the metal cover at the top of the right console bay (the lower part of this panel contains the marginal check controls, which are described in 9.1).

The name used in the logic drawings for a register or control level is listed in parentheses whenever it differs from the name engraved on the panel. When any indicator is lit, the associated flip-flop is in the 1 state or the associated function is asserted. Indicators for logic elements that retain their states over a considerable number of main sequences display useful information while the processor is running, but most indicators change too frequently and are therefore discussed in terms of the information they display when the processor has stopped. For maintenance purposes, the processor may be stopped from the console after every memory subroutine. Switches located inside the bay doors allow stopping after AR subroutines and single stepping through a shift-count. However, the discussion here is limited to stops at the end of a main sequence, i.e., at the completion of an instruction. This includes all programmed halts as well as the situation in which the operator latches down the INSTRUCTION STOP key to run a program at slow speed stopping after every instruction.

a Console Operator Panel

This panel contains indicators for most of the registers and control flip-flops that are of concern to the operator and contains all of the operating keys and switches. The switches supply continuous levels and all but the rotary speed controls and the console lock are 2-position toggles for which up is 1 or on. The keys are momentary contact levers that initiate or terminate operations, or produce an action only while held on.

Indicator Registers

INSTRUCTION (IR0-8) - Bits 0-8 of the instruction just completed. If the left three lights are all off, the instruction is a UUO and the remaining bits are defined by the program; if the left three are on, the instruction is an IOT and the remaining lights display the first six bits of the device code. Any other configuration of the first three bits indicates the basic format, for which the register contains the instruction code.

AC (IR9-12) - For instructions using the basic format, these four bits are usually an accumulator address, but for some instructions they are used for special purposes such as addressing flags. In an IOT instruction the left bit is the LSB of the device code; the remaining three bits specify one of the eight IOT instructions.

I (IR13) - This is the indirect bit, and it should always be off when the processor has stopped at the end of an instruction.

INDEX (IR14-17) - Contains the address of the last index register used in the instruction just completed. If the four lights are all off, there was no address modification in the final address cycle.

MEMORY (MI) - This 36-bit register displays the contents of the memory location associated with any console examine or deposit operation. The lights may also be used to display any desired location while the processor is running.

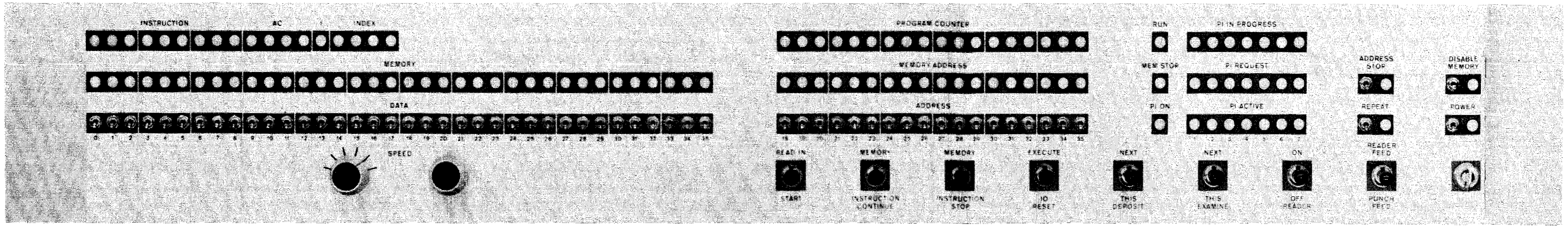


Figure 3-1 Console Operator Panel

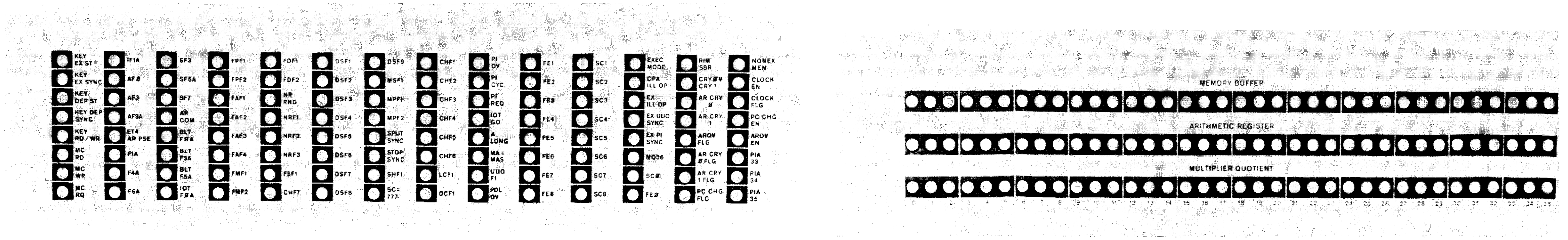


Figure 3-2 Bay Indicator Panels

PROGRAM COUNTER (PC) - This 18-bit register contains the address of the next instruction in the program.

MEMORY ADDRESS (MA) - On a programmed halt this 18-bit register indicates an address one greater than that of the location containing the halt instruction. On an instruction stop in slow speed operation, the register usually contains the address used for the last memory access. However, if there was no storage, either in the store cycle or in a subroutine, it contains the effective address, which may or may not be the address of the last memory access.

In addition to the above there are three processor registers located at the top of the in-out panel (c below).

Switch Registers

DATA (DS) - This register allows the operator to supply a 36-bit word to the processor. The operator may either deposit the word in memory or cause the processor to execute it as an instruction. The program may also read DATA with a DATAI for the processor.

ADDRESS (MAS) - By means of this 18-bit register the operator may specify address for use with the operating keys and switches. Whenever the memory subroutine gains access to the location specified by ADDRESS, the contents of that location are displayed by MEMORY. For a read request MEMORY displays the word read; for a write request the word written is displayed.

Control Indicators

RUN - Lit while the processor is running in normal operating mode, with each main sequence triggering the next. When the light goes off, the processor stops upon completion of the current instruction.

MEM STOP (MC STOP) - If this light goes on at the beginning of a memory subroutine, the processor stops after memory access is completed because the subroutine fails to send a restart pulse to the calling sequence. If RUN is also on, the processor can be restarted only by lifting the MEMORY CONTINUE key. If RUN is off, other keys may be used but only MEMORY CONTINUE restarts the interrupt key function.

PI ON (PI ACTIVE) - Indicates that interrupt requests can be granted by the priority interrupt system.

PI ACTIVE (PIO1-7) - These lights indicate which PI channels are on. The numerals below the lights specify the channels for these and the following two sets of indicators.

PI REQUEST (PIR1-7) - These lights indicate the channels on which requests have been synchronized. The program can force a request even if a channel is not on; for a request from any other source the REQUEST light can go on only if the corresponding ACTIVE light is on.

PI IN PROGRESS (PIH1-7) - These lights indicate the channels on which breaks are currently being held. Several lights may be on simultaneously, but while a given light is on, no higher-numbered light may go on; a lower-numbered channel can interrupt following the PI cycle(s), and the channel that is actually being serviced is the lowest-numbered one whose light is on. When a PROGRESS light goes on (following a jump to a routine for the break), the corresponding REQUEST goes off and cannot go on again until PROGRESS goes off.

If a break is serviced by a block IOT without overflow, the PIH flip-flop is set and cleared within a single PI cycle so REQUEST goes off without PROGRESS going on. If there is overflow, two PI cycles are required; at the end of the first, PROGRESS will not yet be on and REQUEST will still be on even though the break is being serviced. However, in this case the PI OV, PI CYC, and PI REQ lights at the top of bay 1 will all be on. The lights act in this way because PROGRESS can go on and remain on in a PI cycle only if the instruction performed is not an IOT. Thus a faulty program can hang up the processor in a PI cycle, and the only visual indication that the break is being held is that PI CYC and PI REQ will be on. For example if the channel location contains a CONO, the processor will repeat the instruction indefinitely with REQUEST on and PROGRESS off; PC will be static and will point to the next instruction in the program.

Operating Keys

In the right half of the operator panel is a row of eight 3-position switch levers, each of which is two logical keys. The momentary contact, up and down positions of a given lever are the on positions for the keys whose names are written above and below; the stable center position is off for both keys. The two levers at the right end of the row control the reader and punch, and these may be used at any time whether the processor is running or not. The other twelve keys affect the processor, and of these, two are stop keys, the others are initiating keys (i.e., they trigger the key cycle). Although special considerations for individual keys are given below, it is assumed throughout the discussion that the executive system is not in use, i.e., that both operator and program have access to all of memory and no operations are illegal. In order to use the keys properly when the executive system is in use, the operator must be fully aware of the special conditions imposed (a complete discussion of the relation of the keys to the programmed operation of the system is presented in 3.3b).

START - This key functions only if RUN is off. It places the processor in normal operation (lighting RUN) and causes the first instruction cycle to retrieve an instruction from the location specified by the ADDRESS switches.

READ IN - This key is exactly the same as START except that it also causes the processor to enter the readin mode, lighting the RIM SBR indicator at the top of bay 1. In this mode the fast memory is disabled, and any memory call with an address 17 or less is given access to the readin area, the normally inaccessible bottom 16 core locations. Whenever an instruction is retrieved from any location above 17, the processor leaves the readin mode.

INSTRUCTION STOP (INST STOP) - Turns off RUN, causing the processor to stop at the completion of the current instruction. This key has a catch that allows it to be left in the on position for single step operation. The turnon of the key triggers events that facilitate emergency stops (for details see 3.3c).

INSTRUCTION CONTINUE (INST CONT) - This key functions only if RUN is off. It causes the processor to resume normal operation (lighting RUN) beginning with the instruction in the location specified by PROGRAM COUNTER. By leaving INSTRUCTION STOP on, the operator can single step instructions by pressing INSTRUCTION CONTINUE. The latter key also has a catch so that by leaving both keys on and using the REPEAT switch, a program can be run at slow speed.

MEMORY STOP (MEM STOP) - This key has a catch that allows it to be left in the on position for single step operation. While the key is on, the MEM STOP light goes on at the beginning of every memory subroutine, causing the processor to stop at the completion of each memory access. During single step operation a call for read-pause-write in the fetch cycle generates only a read request so the processor does not hold memory during the stop. The subsequent restart then triggers a separate write cycle. This key is used only for maintenance purposes and the meaning of the lights depends upon where the stop occurs within the main sequence. However, MEMORY ADDRESS always displays the location to which access was made.

MEMORY CONTINUE (MEM CONT) - This key functions only if the MEM STOP light is on, and it then restarts whatever sequence was interrupted by the MEMORY STOP key (it also turns off the light). By leaving MEMORY STOP on, the operator can use MEMORY CONTINUE to single step by memory calls. The latter key also has a catch so that by leaving both keys on and using the REPEAT switch, a program can be run at slow speed from one memory call to the next.

EXECUTE (EXEC) - This key functions only if RUN is off. It causes the processor to execute the instruction contained in the DATA switches and stop immediately upon completing it. While the key is on, the normal program counting in the execute cycle is inhibited; thus PC cannot be affected unless a skip or jump is executed. A programmed skip always increments PC once for the normal program count and may increment it a second time for the skip; from the console a skip increments PC at most once. A programmed jump always increments PC before saving it so that it points to the next instruction in the program; when executed from the console, a jump loads PC normally but saves the count that is already in it.

IO RESET - This key functions only if RUN is off. It clears all flags, control flip-flops and control registers in the processor (placing it in executive mode) and in most equipment connected to the in-out bus.

DEPOSIT THIS (DEP) - Deposits the contents of DATA in the location specified by ADDRESS. The word deposited is displayed by MEMORY. If RUN is off during the deposit, the processor stops with the MA lights displaying the address of the affected location. This key should be used while the processor is running only if there is no chance of a program halt occurring (3.3b).

DEPOSIT NEXT (DEP NXT) - This key functions only if RUN is off, It deposits the contents of DATA in the location whose address is one greater than that specified by MEMORY ADDRESS, and the word deposited is displayed by MEMORY. At the completion of the operation MA contains the address of the affected location.

EXAMINE THIS (EX) - Causes MEMORY to display the contents of the location specified by ADDRESS. If RUN is off during the operation, the processor stops with the MA lights displaying the address of the examined location. This key should be used while the processor is running only if there is no chance of a program halt occurring (3.3b).

EXAMINE NEXT (EX NXT) - This key functions only if RUN is off. It causes MEMORY to display the contents of the location whose address is one greater than that specified by MEMORY ADDRESS. At the completion of the operation MA contains the address of the examined location.

READER ON - Turns on the reader motor, energizes the brake, and triggers a PI request on the reader channel.

READER OFF - Turns off the reader motor, releases the brake, and triggers a PI request on the reader channel.

READER FEED - Feeds tape through the reader while held on (provided the reader is on).

PUNCH FEED - While this key is held on, the punch generates blank tape, i.e., tape with only feedholes punched.

Operating Switches

The first four switches are toggles located at the right end of the operator panel, and associated with each is an indicator that lights while the switch is on.

POWER - This switch applies power to the processor and the control units for reader, punch, and Teletype, and makes power available to all external units (memories, peripheral equipment) whose local power controls are in remote. Almost every unit has its own power switch, which if left on, allows the unit to come on with system power. Exceptions include the reader, which must be turned on and off at the processor console, and the punch, which is turned on by the logic and goes off automatically whenever it is not called for 5 sec. After turning POWER on, wait a few seconds to allow the power clear to terminate and memory power to come on.

ADDRESS STOP (ADDR STOP) - While this switch is on, a memory stop occurs whenever access is made to the location specified by the ADDRESS switches. At the stop the MEM STOP light is on, and MEMORY displays the word read or written. Throughout the time that the switch remains on, any fetch cycle call for read-pause-write generates only a read request, so the processor does not hold memory following the stop. The subsequent restart then triggers a separate write cycle.

DISABLE MEMORY (MEM DISABLE) - While this switch is off, the failure of a memory to respond within 100 μsec to a request for access turns on the NONEX MEM light on the bay 1 indicator panel, causing a PI request on the processor channel. If the switch is on, such failure causes the processor to hang up in the memory subroutine. The operator can free the processor by pressing INSTRUCTION STOP and then I/O RESET.

REPEAT - Causes the sequence initiated by an operating key to be repeated as long as the key is held on. The sequence is iterated at a rate determined by the SPEED switches.

SPEED - These switches allow the operator to vary the repeat interval from 3.4 μsec to 8 sec in six overlapping ranges. They include a 5-position rotary range switch and a potentiometer knob for fine control within each range.

Console Disable - In the lower right corner of the panel is a key-locked switch. Turning the key clockwise disables all operating keys and switches on the panel (except those for the reader and punch) so no one can interfere with the operation of the processor.

b Bay Indicator Panels

Figure 3-2 shows the indicator panels at the tops of bays 1 and 2. Bay 2 displays the three main full-word registers: memory buffer MB, arithmetic register AR, and multiplier-quotient register MQ. Since the results of an instruction are stored in memory, these registers are useful primarily for single-step maintenance operation, and their contents at an instruction stop depend entirely upon the instruction just performed. If the instruction requires storage, MB always contains the last word stored. AR contains the word stored in an accumulator (if any), and for a double-length result MQ contains the word stored in a second accumulator.

On bay 1 are the indicators for flip-flops and control levels. Indicators for the flags described in 2.1c are at the right end of the panel. At the top of the third column from the right is the EXEC MODE light, which is driven from the 0 output of the user flag and is thus lit when the executive routine is running (or the executive system is not in use). The remaining flags are as listed in the text, although the names engraved on the panel are in many cases abbreviated from those used in the logic drawings. PDL OV is at the bottom of the sixth column from the right. The second light in the third column, CPA ILL OP, is the illegal operation flag, which indicates that a user program has attempted to address a location outside of its assigned core area and should not be confused with EX ILL OP just below. The latter flip-flop inhibits relocation when a UUO or an illegal user instruction is trapped, and the light is always off at an instruction stop. In the second column are the AR overflow and carry flags and the PC change flag. The carry flags should not be confused with the carry flip-flops above them: the flip-flops detect carries in AR and their states are transferred to the flags only in those instructions wherein the information is relevant to the program. The remaining flags are in the right column.

Besides the flags, the indicator panel also includes shift counter SC, floating-exponent register FE, several important control levels, and a multitude of flip-flops that govern the sequencing of the various processor cycles, special sequences, and subroutines. The following indicators are of importance to the operator in normal operation (unless otherwise specified, the meaning of a light is given for an instruction stop):

KEY EX SYNC, KEY EX ST - If the operator presses EXAMINE THIS while the processor is running and it stops with both of these lights on, the desired key function has not been performed. If only the sync light is on, the operation may have been performed incorrectly (see 3.3b). If the key is pressed while RUN is off, the start light does not go on at all, but the sync light goes on and remains on until some other initiating key is pressed.

KEY DEP SYNC, KEY DEP ST - If the operator presses DEPOSIT THIS while the processor is running and it stops with both of these lights on, the desired key function has not been performed. If only the sync light is on, the operation may have been performed incorrectly (see 3.3b). If the key is pressed while RUN is off, the start light does not go on at all, but the sync light goes on and remains on until some other initiating key is pressed.

CHF7 - If this light (bottom, fifth column from left) is on following a PI cycle that executes a BLKI or BLKO, an interrupt has occurred between the two parts of a character operation. Following a JSR in a PI cycle, the light will be off even if a character operation was interrupted. If CHF7 is on following a JRST, the instruction is returning from a break and the processor is about to restart an interrupted character operation.

SPLIT SYNC - Indicates that if there was a read-pause-write call during the preceding fetch cycle, it triggered only a read request, and the subsequent restart triggered a separate write cycle.

STOP SYNC - Indicates that the preceding fetch cycle triggered a read-pause-write memory cycle.

PI OV - Indicates that a BLKI or BLKO performed in a PI cycle has overflowed.

PI CYC - This light goes on when a PI request is honored, but is still on at the completion of an instruction only if a second PI cycle is required for the interrupt (i.e., the instruction was a BLKI or BLKO that overflowed).

PI REQ - At the completion of an instruction this light is on whenever PI OV and PI CYC are on. If those two lights are not on, PI REQ indicates that the PI system is active and a request that has been synchronized has not yet been honored. This can occur if a previous instruction activated the system and some requests were already waiting. If the system was already active, either a request was made by the program or synchronized by a BLT, or several requests were synchronized simultaneously and the processor has just finished servicing one of those with higher priority.

A LONG - Indicates that the address cycle of the preceding instruction used an index register for address modification or used an indirect address.

MA = MAS - Indicates that the number displayed by the MEMORY ADDRESS lights is identical to that contained in the ADDRESS switches. This light is on whenever an address stop occurs but may be on at other times as well.

EX PI SYNC - Indicates that the main sequence just completed was a PI cycle. This light remains on even when PI CYC goes off before the instruction is completed.

RIM SBR - Goes on when the operator presses the READ IN key and remains on until an instruction is retrieved from a location above 17. The light is always on if the system includes no fast memory, or if it is not in use.

PIA 33, 34, 35 - Indicate the PI channel assigned to the processor. If all three lights are off, no channel has been assigned.

At the completion of an instruction, SC and FE may have any configuration. Besides the flags and the control indicators discussed above, the following lights may be on at an instruction stop: MC WR, NRF2, NRF3, DSF7, MPF2, SC = 777, UUOF1, EX UUO SYNC (always on), MQ36, CRY0 ⊻ CRY1, AR CRY0, AR CRY1. None of the remaining lights should be on at an instruction stop. At a memory stop MC RD will be on if the memory cycle was used to retrieve information, and at least one other light will be on to indicate the point at which a time chain is stopped awaiting the return from the memory subroutine. For example if KEY RD/WR in the left column is on, the memory stop occurred in an examine or deposit operation initiated at the console. Refer to 9.1 for further information on the use of these indicators for maintenance purposes.

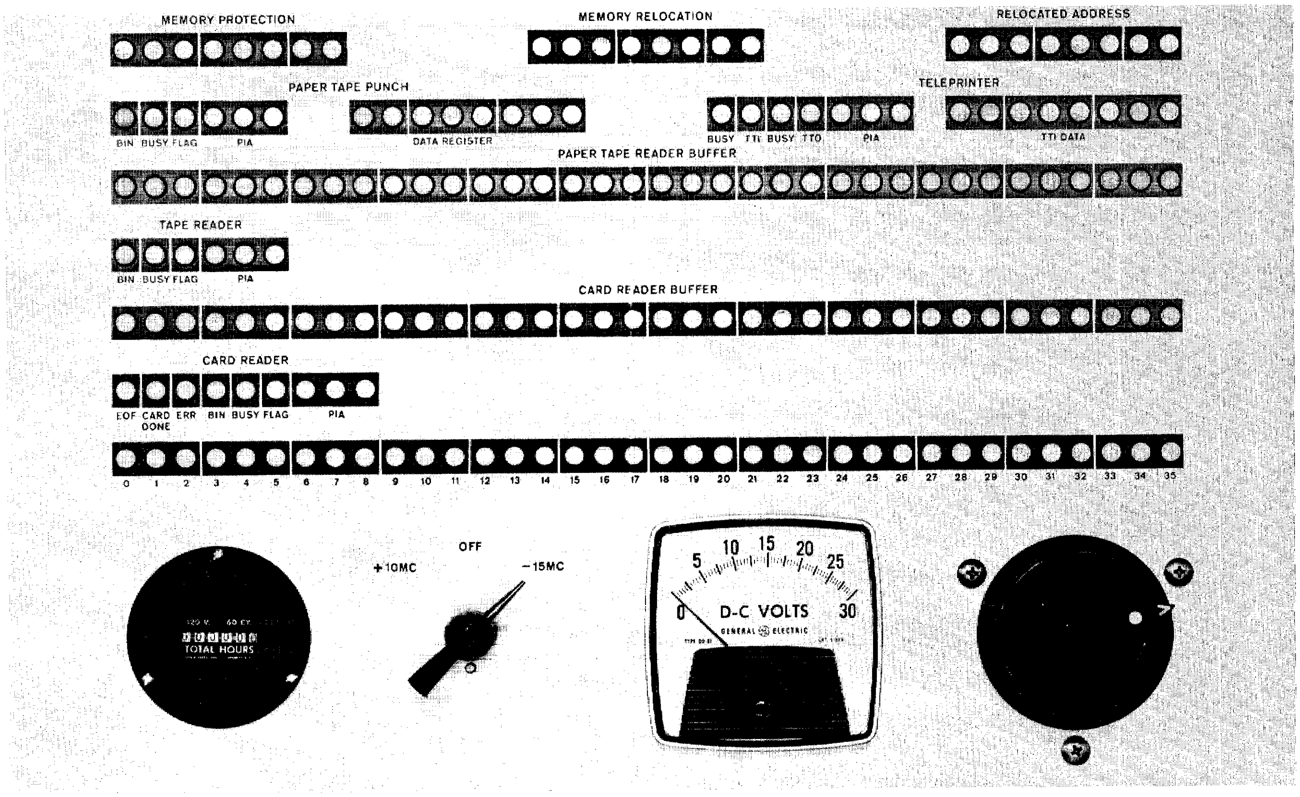


Figure 3-3 In-Out and Marginal Check Panel

c In-Out Indicator Panel

Figure 3-3 shows the panel that contains the in-out indicators and the marginal check controls (the latter are described in 9.1). At the top of the panel are three 8-bit indicator registers that are associated with the executive system rather than the in-out equipment. These are as follows:

MEMORY PROTECTION (PR) - Defines the size of the block in core available to a user program. The number of locations in the block is 2^{10} times the number one greater than that contained in the register. Each program must use addresses from zero to one less than the block size. If a user attempts access to an address greater than the number in PR followed by ten ones, i.e., to an octal address greater than C(PR) x 2000 + 1777, the processor skips the remainder of the current instruction, and the CPA ILL OP light at the top of bay 1 goes on causing a priority interrupt on the processor channel.

MEMORY RELOCATION (RLR) - Specifies the position of a user block in core. The address of the first location in the block is the number contained in RLR followed by ten zeros. Each user address other than for fast memory is relocated to the assigned block by adding C(RLR) x 2000 to the number displayed by MEMORY ADDRESS.

RELOCATED ADDRESS (RLA) - These lights display the most significant eight bits of a user address as it is placed on the memory bus (the least significant ten bits come directly from MA).

The bottom row of indicators on the panel displays the contents of the 36 data lines in the I/O bus. Since the bus is reset following every data transfer, the lights should always be off when the processor is stopped. The remaining lights are the buffers and some of the control and status bits for the reader, punch, keyboard-printer, and card reader. Each device has a 3-bit PIA register that contains the number of the PI channel assigned to it. Whenever the FLAG light for any device goes on, a PI request is made on the channel specified by the associated PIA (if all three PIA lights are off, there is no channel assignment). The lights labeled TELEPRINTER are actually for two distinct devices, keyboard and printer. Both share a common PI assignment but have duplicate control bits. The data buffer shown is actually for the keyboard; the printer buffer is not shown because it automatically clears as each character is being transmitted.

PAPER TAPE PUNCH -

DATA REGISTER (PTP1-8) - Contains the last character punched. The buffer bits are numbered 1 to 8 from right to left and correspond to a frame of tape viewed with the feed hole near the right edge. The buffer receives information from bus lines 28-35, with line 35 supplying the information for bit 1.

BIN (B) - While this light is on, any punch operation always punches hole 8, never punches hole 7, and punches hole 6-1 according to the information on bus lines 30-35. While the light is off, the information on lines 28-35 is punched.

BUSY - Indicates that the punch is in operation.

FLAG - Causes a PI request upon completion of a punch operation. FLAG goes off when the program supplies another character.

TELEPRINTER -

TTI DATA (TTI1-8) - Contains the last character received from the keyboard. The buffer bits are numbered 1 to 8 from right to left so that when a character is shifted in at the left, the first bit received ends up in buffer bit 1. The character is transferred to the processor over bus lines 28-35 with bit 1 on line 35.

BUSY, TTI (BUSY, FLAG) - This pair of lights (at the left) is for the keyboard. BUSY goes on when a key is struck. When the entire character is assembled in TTI DATA, BUSY goes off and TTI goes on, requesting an interrupt. TTI goes off when the program retrieves the character.

BUSY, TTO (BUSY, FLAG) - In the right pair of lights, BUSY is on while a character is being transmitted to the printer. When transmission is complete, BUSY goes off and TTO goes on, requesting an interrupt. TTO goes off when the program supplies a new character.

PAPER TAPE READER -

BUFFER (PTR0-35) - Contains data read from tape but no yet retrieved by the program. The 36 buffer bits are numbered to correspond to the bus lines. Characters of six or eight holes from tape are brought in at the right end of the buffer, with bit 35 receiving hole 1.

BIN (B) - When this light is on, each reader operation reads hole 6-1 of only those characters in which hole 8 is punched and assembles six such characters into a 36-bit word. When each character is brought into the buffer at the right, the previously read characters are shifted left. When BIN is off, each read operation retrieves a single character, sensing all eight holes.

BUSY, FLAG - When BUSY goes on, the reader goes into operation retrieving information in the manner specified by BIN. When the required number of characters is retrieved, BUSY goes off and FLAG goes on, requesting an interrupt. FLAG goes off when the program retrieves the information from the buffer. FLAG is also set when the operator turns the reader motor on or off.

CARD READER -

BUFFER (CR0-35) - The 36 bits of this buffer are numbered left to right to correspond to the bus lines. Six-bit characters from cards are brought in at the right end of the buffer over reader signal lines 1, 2, 4, 8, A, B, with buffer bit 35 receiving information from line 1. The program specifies whether information retrieved from a card is to be placed on the bus in units of one, two, or six characters. If more than one character is to be read per bus transfer, previously read characters are shifted left in the buffer as new ones come in.

BIN (B) - While this light is on, a card is read in binary mode wherein each column is read as two characters. The first character is from the lower half of the column (holes 4-9) with hole 9 on reader line 1; the second character is the upper half (holes 12, 11, 0, 1, 2, 3) with hole 3 on line 1. If the light is off, reading is in alphanumeric mode in which the reader converts the Hollerith character in a column to the Burroughs 6-bit code, and six columns are required to fill the buffer.

BUSY - This light goes on when the program requests that the reader begin a card cycle, and the light remains on until the entire card is read.

FLAG - Each time the buffer contains the amount of information specified by the program, this light goes on, requesting an interrupt. Retrieving the data turns FLAG off and clears the buffer, which is cleared automatically if the program does not respond to the request before the next column is read. FLAG may light after each column or only when the buffer is full; depending on whether BIN is off or on respectively, a single column is one or two characters, and a full buffer contains six or only three columns. FLAG also signals that the reader has finished an entire card regardless of the number of characters in the buffer, or that a card jam has occurred.

CARD DONE - Lit from the time the reader completes one card cycle until it starts another. It is possible for CARD DONE and BUSY to be lit at the same time. The program turns on BUSY to cause the reader to begin a card cycle, but CARD DONE remains on until the cycle actually starts.

EOF - When the card hopper is empty, pushing the END OF FILE button on the reader turns on this light. It also turns on FLAG, requesting an interrupt.

ERR (CREL) - Indicates a validity check or read check error in the reader.

3.2 OPERATION OF IN-OUT EQUIPMENT

This section describes the normal operation of the photoelectric perforated tape reader, paper reader, paper tape punch, keyboard-printer, and card reader. Information for other devices is included in their maintenance manuals and in the operator manual, PDP-6 Operation. DEC also supplies manufacturer manuals for all devices included in a PDP-6 system.

a Tape Reader

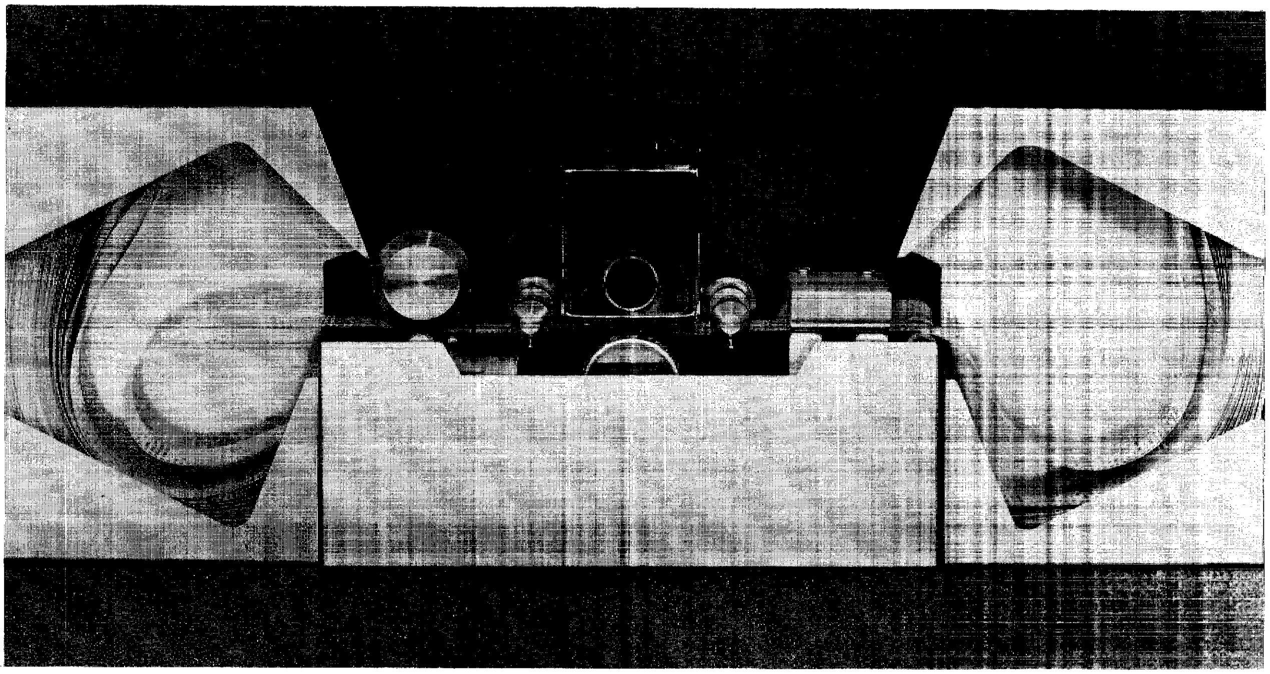


Figure 3-4 Paper Tape Reader Type 760

Before loading a tape in the reader, turn off the reader motor by pressing the READER OFF key. This releases the brake so that tape may be inserted, and it also requests an interrupt on the reader channel to inform the program that the reader is unavailable. Place the fanfold tape stack vertically in the bin at the right with the tape oriented so that the front end of the tape is nearest the read head and the feed holes are nearer the reader mounting panel, i.e., away from the operator. Take three or four folds of tape from the bin and slip the tape into the reader from the front so it is threaded as shown in Figure 3-4. Make sure that the part of the tape in the left bin is placed to correspond to the folds, otherwise it will not stack properly.

Once the tape is properly loaded lift the READER ON key to start the motor and energize the break. This also requests an interrupt to inform the program that the reader is on. When using the readin mode loader, always turn on the reader before starting the loader. The program makes use of the reader by sending signals to the clutch, which moves the tape past the sensing photocells. After the program has finished reading the tape, run out the remaining leader by lifting the READER FEED key, or turn the reader off so the tape may be slipped out directly.

b Tape Punch

The punch is located in a drawer at the top of the left console bay. The punch mechanism faces the right side of the drawer. Fanfold tape is fed from a box as shown in Figure 3-5. After punching, the tape moves into a storage bin from which the operator may remove it through a slot on the front of the drawer.

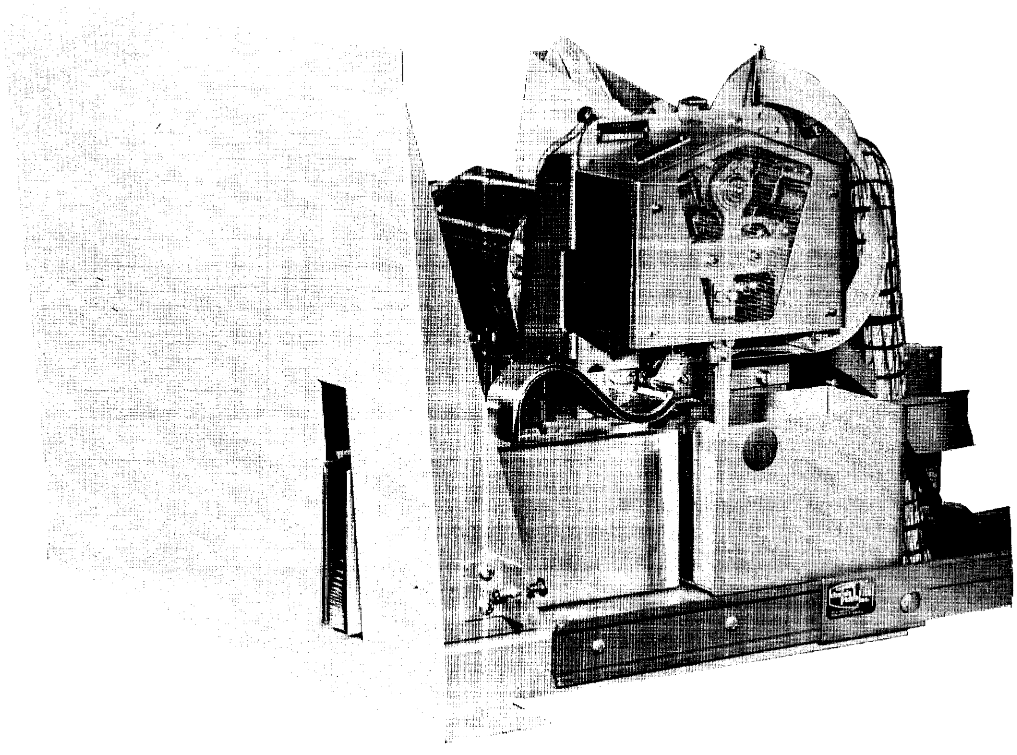


Figure 3-5 Paper Tape Punch Type 761

To load the punch, first empty the chad box below the punch mechanism. Then tear off the top of a box of fanfold tape (the top has a single flap; the bottom of the box has a small flap in the center as well as the flap that extends the full length of the box). Set the box in the frame at the right side of the punch and thread the tape through the mechanism as shown in Figure 3-5. The arrows on the tape should point in the direction of tape motion. If they point in the opposite direction, the box was opened at the wrong end; remove the box from the frame, seal up the bottom, open the top, and thread the tape correctly. After loading the tape, hold down the console PUNCH FEED key long enough to feed approximately 18 in. of leader. Make sure the tape is feeding and folding properly in the storage bin.

To remove a length of perforated tape from the bin, first hold down PUNCH FEED long enough to provide an adequate trailer at the end of the tape (and also leader at the beginning of the next length of tape). Remove the tape from the bin and tear it off at a fold within the area in which only feed holes are punched. Make sure that the tape left in the bin is stacked to correspond to the folds; otherwise, it will not stack properly as it is being punched. After removal, turn the tape stack over so the beginning of the tape is on top, and label it with both name and date.

c Teletype Keyboard-Printer

The teletypewriter (Figure 3-6) provides two-way communication between operator and computer. It is actually two independent devices, keyboard and printer, which may be operated simultaneously. The equipment operates at speeds up to ten characters per second, with 8-bit characters plus start and stop control signals transmitted serially. Located at the right front of the unit is a 2-position rotary switch, OFF/LINE. When this switch is set to LINE, the unit is on line and it goes on and off with system power.

The keyboard resembles that of a standard typewriter with four rows of keys and a space bar. Striking a key transmits a character to the Teletype control unit connected to the bus, but the character is printed or the function executed only if the processor sends it back to the printer. The line feed moves the carriage only vertically with a spacing of six lines to the inch. The return moves the carriage to the left margin but does not feed a line: to start a new line the operator must strike both return and line feed. Codes for the characters on the lower parts of the key tops can be transmitted merely by striking the keys. Codes for printable characters on the upper parts (punctuation, ampersand, percent sign) are transmitted by holding down the shift key when striking the character key. Control codes are transmitted by holding down the control key, CTRL, when striking the appropriate character key. Codes for all characters listed on the keyboard and some that are not can be transmitted to the computer, but codes for some of the control functions have no effect on the printer when sent back. Table A4-2 lists all codes, their ASCII assignments, and the key combinations required to transmit them. Because of recent changes in the code, there may be slight differences in the printing characters associated with certain key positions. In such cases alternate characters are listed in parentheses.

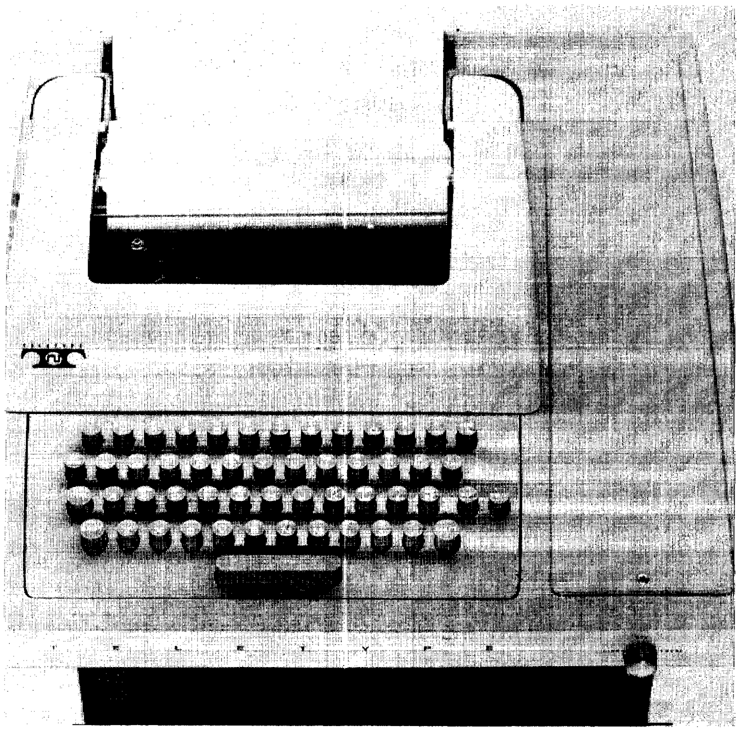


Figure 3-6 Keyboard-Printer Type 626

In line with the space bar below the keyboard are four red buttons. At the right end is the repeat button REPT. Pressing this button and striking any character key causes repeated transmission of the corresponding code so long as REPT is held down. Characters that require the shift key may also be repeated in this manner, but there is no repetition of control characters.

The red button on the left, BRK RLS, is not connected in the console teletypewriter. The remaining two buttons, LOC LF and LOC CR, are the local line feed and carriage return. These buttons affect the printer directly and do not transmit codes over the bus.

Paper installation, ribbon replacement, and the procedure for setting horizontal and vertical tabs are described below. All references are to figures in typing unit section 574-220-100 in Vol. 1 of Teletype Bulletin 281B (Technical Manual 35 Keyboard Send-Receive (KSR) and Receive-Only (RO) Teletypewriter Sets).

Paper

The unit has a sprocket feed and uses 8-1/2 x 11 fanfold form paper. The supply is held in a tray at the back of the unit, and printed forms can be torn off against the edge of the glass window in front of the platen. To replace the paper first remove the upper cover by pressing the cover release button on the right side. To free the remaining old paper for removal, lift the paper guides by pushing the handle marked PUSH at the right of the platen. To insert new paper from the tray, offer it up below the platen at the rear, lining up the holes at the edges of the paper with the sprockets, and press the local line feed button to draw the paper in under the platen.

Ribbon

Replace whenever it becomes worn or frayed or when the printing becomes too light. Disengage the old ribbon from the ribbon guides on either side of the type block, and remove the reels by lifting the spring clips on the reel spindles and pulling the reels off (the ribbon feed mechanism is called out in Figure 4). Remove the old tape from one of the reels and replace the empty reel on one side of the machine; install a new reel on the other side. Push down both reel spindle spring clips to secure the reels. Unwind the fresh ribbon from the inside of the supply reel, over the guide roller, through the two guides on either side of the type block, out around the other guide roller, and back onto the inside of the take-up reel. Engage the hook on the end of the ribbon over the point of the arrow in the hub. Wind a few turns of the ribbon and make sure that the reversing eyelet has been wound onto the spool. Make sure the ribbon is seated properly and feeds correctly in operation.

Tabs

The horizontal and vertical tabulator mechanisms are also called out in Figure 4. Each is a slotted wheel surrounded by a spring on which are mounted a number of tab stops. The horizontal tab mechanism is shown in detail in Figure 47. The slotted wheel is mounted on the spacing drum, and a tab can be set by inserting a tab stop in a groove where it catches the tabulator pawl when the type block carriage is in the desired position. With needle-nose pliers or equivalent, lift the tab stop out of the slot in the wheel against the spring tension. Slide the stop along the spring in the desired direction, and reinsert it into the slot at the new location. A stop may be removed from use by turning it so that it does not catch the pawl. Figure 49 shows the vertical tabulator mechanism. The slots in this disc allow vertical tabs at any desired line, but adjacent tabs must be at least 1 in. apart.

d Card Reader

The B122 Card Reader handles 200 cards per minute and has a hopper and stacker capacity of 500 cards. With a trivial change in the control unit logic, the processor can control the B124 Card Reader. Its operation is similar to that described here, but the maintenance information given in Chapter 9 applies only to the B122. The B124 handles 800 cards per minute and has a hopper and stacker capacity of 2000 cards. In both machines the cards are read lengthwise and sensed photoelectrically. In Figure 3-7, the hopper is at the right, the stacker at the left; in the center is a console that contains the operating buttons and indicators. Of the following four indicators, the first is white, the other three are red error indicators.

NOT READY Indicates one or more of the following:

START button has not been pushed
Hopper empty
Stacker full
Card jam (feed check)
Read check
Validity check (only when VALIDITY ON button is lit)

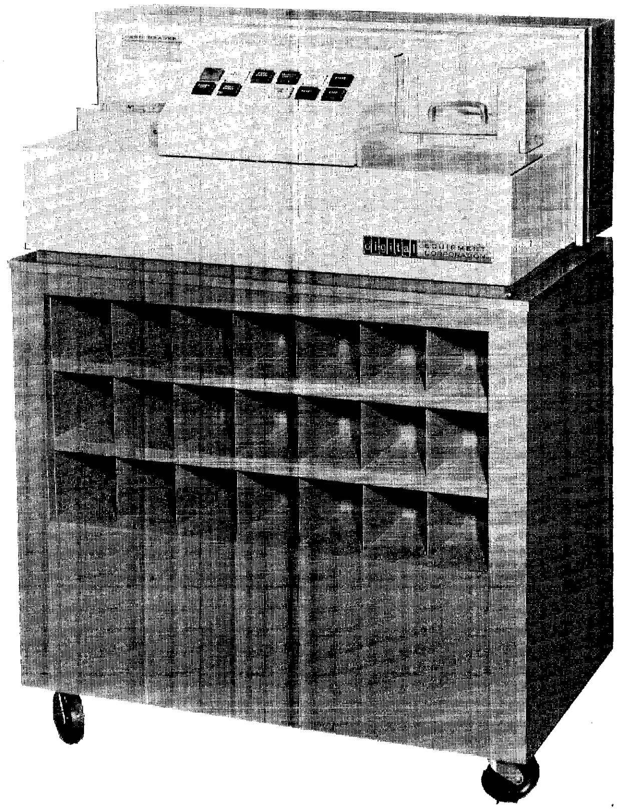


Figure 3-7 Card Reader Type 461

The reader cannot respond to the program while NOT READY is lit. The program may check a not-ready status bit to determine if the reader is available. This status bit is 1 on any of the above conditions and also when the reader power is off (the console cover is interlocked to turn power off when not in place).

READ CHECK - Indicates a NOT READY condition due to a malfunction in the read circuits, e.g., exciter lamps, solar cells, photo amplifiers. There is no read checking during any card cycle; otherwise, read checking is continuous. When this light is on, the ERR light on the processor in-out panel is also on and the reader error status bit is 1.

FEED CHECK - Indicates a NOT READY condition due to a jammed card or failure to select a card. The drive motors stop as soon as the light goes on. A signal to the processor turns on the FLAG light on the in-out panel, requesting a priority interrupt on the channel assigned to the reader; the signal also supplies a status bit to the program.

VALIDITY CHECK - This light functions only if the VALIDITY ON button is lit. It then indicates that an invalid punch combination has been read in alphanumeric mode. When this light is on, the ERR light on the processor in-out panel is also on and the reader error status bit is 1.

Located on the left side of the stacker is the main power switch that controls power to the reader auxiliary power supply. With this switch on, the reader may be turned on either at the reader console or by turning on system power at the processor console. The reader console contains seven buttons, three of which light; the first six below are momentary contact, the last is alternate action.

POWER ON - Green button which turns on the main power supply, the reader motors and, after a 3-sec delay, the reader control logic. This action is duplicated by turning on system power at the processor console. Button is illuminated when power is on.

POWER OFF - Turns off the main supply, but the auxiliary supply remains on.

START - Turns off the NOT READY indicator provided no other not-ready condition exists.

STOP - If the reader is in operation, this button turns on the NOT READY light and the reader stops when the current card runs out to the stacker.

RESET - Turns off the three red check lights: READ CHECK, FEED CHECK, VALIDITY CHECK.

END OF FILE - Pressing this white button when the hopper is empty, lights the button and turns on the EOF and FLAG lights on the processor in-out indicator panel, requesting a priority interrupt on the channel assigned to the reader; the button light goes out when cards are placed in the hopper. The signal generated by the button can be checked as a status bit by the program.

VALIDITY ON - This is an alternate-action yellow button that is lit when on. An invalid punch combination read in alphanumeric mode is sent to the reader control unit as an all-zero character; when the switch is on, and invalid punch also lights VALIDITY CHECK and stops the reader.

In addition to the above there are also interlocks in the hopper and stacker and a LOCAL RUN toggle switch under the cover; the interlocks generate the NOT READY condition when the hopper is empty or the stacker is full. Raising the toggle switch (local) causes the reader to feed cards continuously until the hopper is empty, and then stop with the FEED CHECK light on. With the switch in remote (down), card cycles can be started from the processor whenever the NOT READY light is off.

For operation off line, turn on the main power switch and press POWER ON; after 3 sec NOT READY should light. For normal operation on line, the main power switch is turned on at system power turnon and the reader is left on with NOT READY lit whenever it is not in use. Cards should be placed in the hopper face down with the 12 edge toward the operator. Place the plastic weight on top of the deck to prevent jamming as the last few cards are read. If any of the red check lights are on, push RESET. Push START to turn off NOT READY, and the reader is then available to the processor. STOP may be pushed at any time to generate a NOT READY condition, causing the reader to stop at the end of the current card. To continue, push START.

When reading is in alphanumeric mode, every column is checked for a valid Hollerith character. If an invalid punch is encountered and the VALIDITY ON button is lit, a VALIDITY CHECK error is indicated and the reader stops at the end of the current card. If a second attempt to read the card fails, check it for improper punches. Table A4-3 lists the Hollerith character codes and all invalid punch combinations (note that the Burroughs code is incompatible with that used by the IBM 029 Card Punch).

If the reader stops with READ CHECK lit, the self-checking circuits have detected a malfunction in the read circuitry. Usually this is either the failure of a lamp or solar cell and may be temporary. If both READ CHECK and FEED CHECK go on together, the reader is in need of adjustment. There is one section of the read circuitry in which a failure is not detected as a read check error. However, a failure in this particular part would cause incorrect timing resulting in a validity check error if reading is alphanumeric. If no invalid punches can be found on a card, but several attempts to read it result in a validity check error, it is likely to be a malfunction in this unchecked part of the read circuits. For particulars refer to the manual for the Burroughs B122 Card Reader.

When a card cycle begins, the card is first contacted by a knife at the bottom edge of the hopper and is pushed into the read station where the feed operation is taken over by rollers. The most probable point for a feed malfunction to occur is at the entry into the read station. If the card is bent, it may jam in the feedways; if the trailing edge had been damaged by frequent handling, the pickup knife may fail to move the card through the rollers. When a card fails to appear at the read station in the prescribed time, the FEED CHECK light goes on and the drive motors stop. Do not attempt to reread a worn or damaged card that has caused a feed check error, but put a duplicate in its place.

In the unlikely event that a card should jam inside the read station, no FEED CHECK is indicated, but no cards are processed either (if reading is in alphanumeric, a card stuck in the read station may produce a validity check error). To check for jammed cards, remove the head cover by lifting it at its base and pulling it out horizontally (an interlock removes power if it has not already been turned off). If necessary, the photocell head can be removed by moving the knurled-head sliding bolts to their vertical positions and squeezing them toward the center. The head can then be lifted straight up. Be sure when replacing the head that it is seated properly and the knurled-head bolts are fully engaged.

3.3 PROCESSOR OPERATING PROCEDURES

After turning on system power at the processor console, check the memories and peripheral equipment connected to the memory and in-out busses. In general all memories should go on with system power unless a single unit has been taken off line deliberately. Whether a particular peripheral device is on depends upon its own organization. Most in-out control units go on and off with system power; however, in some instances power supplies must be turned on and off independently of the processor.

a Read In

In order to allow initial information to be brought into memory, a readin loader is usually kept permanently in the part of core that is ordinarily inaccessible because of the fast memory. To use the readin loader, set the appropriate starting location in the ADDRESS switches and lift the READ IN key. This turns on the RIM SBR light at the top of bay 1, and while the light is on, any memory address from 0 to 17 provides access to core instead of fast memory. The light goes out and the processor leaves readin mode whenever an instruction is retrieved from any location above 17. Read in can be single stepped using the CONTINUE and STOP keys, but any other key takes the processor out of readin mode.

To deposit a loader in the bottom of core, the operator must make use of a small toggle switch labeled RIM MAINT, which is mounted behind the double doors on bay 2, on a bracket at the left end between mounting panels 2L and 2M. Putting this switch up holds the processor in readin mode regardless of any action taken at the console, so the operator may deposit the loader. Place the first word in the DATA switches, set the ADDRESS switches to the first location (0 is most convenient), and lift DEPOSIT THIS. MEMORY displays the word deposited. If the remaining words are in consecutive locations, they may be deposited in order by setting them in the DATA switches and pressing DEPOSIT NEXT for each. Although all words are displayed when deposited, it is a good idea to check the entire loader by going through it first pressing EXAMINE THIS, then EXAMINE NEXT. After the loader has been deposited, turn RIM MAINT down.

b Operating Keys

The operator should check material accompanying each program for information on halts, tape requirements, and so forth. Every program is begun by either START or READ IN. On a halt the operator should make note of the console lights, particularly PROGRAM COUNTER, and do whatever is requested in the program operating instructions; the operator may restart by pressing INSTRUCTION CONTINUE.

To debug programs, INSTRUCTION STOP may be latched on and the program single stepped using INSTRUCTION CONTINUE. Or with the REPEAT switch on, and both INSTRUCTION STOP and INSTRUCTION CONTINUE latched down, the program speed can be varied by the SPEED controls. By similar use of the MEMORY STOP and MEMORY CONTINUE keys, a program may be single stepped from one memory call to the next; low-speed operation can be effected by using REPEAT.

The keys for the reader and punch may be operated at any time whether the processor is running or not. The stop keys may also be pressed at any time, but ordinarily these are used only for single-step or low-speed operation (for special stop considerations see c below). The remaining ten keys use the key cycle to initiate some operation, if only a clear function as is the case with IO RESET. For seven keys, entry into the key cycle is gated by the 0 state of RUN so that inadvertent key manipulation can have no effect while a program is running. Even through RUN remains on throughout memory single stepping (unless of course a program halt should occur), the program can be restarted by pressing MEMORY CONTINUE; at a memory stop MEM STOP is on. Of the initiating keys, only EXAMINE THIS and DEPOSIT THIS have any effect while the program is actually running, i.e., there is neither an instruction stop nor a memory stop. Either of these keys inserts a key cycle between two instructions without stopping the processor. However, do not use these keys if there is any chance of a program halt occurring. The halt instruction stops the processor by clearing RUN at the beginning of the execute cycle, and it is thus possible for a key cycle to be triggered between the time RUN is cleared and the instruction is completed (of course the same caution holds for any wanton key manipulation while the processor is running). If a program halt should occur (RUN goes off) at the same time that EXAMINE THIS or DEPOSIT THIS is operated, check the corresponding pair of SYNC and START lights in the left column on the bay 1 indicator panel. If both lights are on, the corresponding key function was not performed; if neither light is on, the key function was performed prior to the halt instruction. If only a SYNC light is on, the key function was performed but there is no way of knowing whether it was executed during the halt or after, and thus the console lights are meaningless. If EXAMINE THIS or DEPOSIT THIS is pressed with RUN off, the SYNC light does not go out but the key function is performed.

There are also special precautions that must be observed while user programs are sharing processor time. While RUN is on, relocation and protection are inhibited during a key cycle so the operator may use EXAMINE THIS and DEPOSIT THIS with all of memory available to him. However, when the processor is stopped, as between instructions in single-step operation, the operator must make sure his actions at the console are compatible with the operating mode. In user mode any address supplied for a key function, including addresses in an instruction initiated by the EXECUTE key, must be smaller than the block size as indicated by the MEMORY PROTECTION lights at the top of the in-out panel, or a priority interrupt for an illegal address will occur. The operator should also understand that unless the address is for fast memory, it is relocated to the block specified by MEMORY RELOCATION (the RELOCATED ADDRESS is displayed at the right). Furthermore, an illegal instruction executed from the console will take the processor out of user mode. The operator must observe the lights at the top of bay 1 to determine what he can do. All addresses and instructions are legal if EXEC MODE is on. If this light is off but PI CYC is on (this can happen only between two PI cycles required for the same interrupt), there is also no relocation or protection. However, in this circumstance not all instructions are legal; an IOT may be executed from the console, but any other illegal user instruction or a UUO will return the processor to executive mode. If both EXEC MODE and PI CYC are off, the operator must observe all user restrictions. The operator can switch from either mode to the other by executing the appropriate instruction from the console. The switch may be made from user to executive mode by pressing IO RESET, but this also clears most of the in-out equipment including all PI assignments.

Care should be exercised in the use of the EXECUTE key whenever priority interrupts are allowed while a program is being single stepped. In addition to observing the user-executive restrictions associated with priority interrupts, the operator must be aware of the following. An interrupt has priority over any instruction including one executed from the console. If an interrupt request is waiting when EXECUTE is pressed, the processor performs the PI instruction instead of the one in DATA switches. If PI CYC is on (indicating that the preceding instruction was a block IOT that overflowed), a non-IOT executed from the console will be taken by the processor to be the jump to the break routine. This will turn off PI CYC, and the JSR to the subroutine will be skipped when the program is continued. Furthermore, the processor will "hold" the break—i.e., PI REQUEST goes off, PI IN PROGRESS goes on, and both the channel on which the interrupt was requested and all lower priority channels will remain disabled as the program continues.

CAUTION

Never under any circumstances press more than one initiating key at a time because the processor will try to perform both functions at once. Note that in low-speed operation one of the continue keys is always on, so although EXAMINE THIS or DEPOSIT THIS can be used while the processor is running, they cannot be used in low-speed operation.

c Emergency Stop

Ordinarily INSTRUCTION STOP is used for single step operation and maintenance procedures, but it can also be used for an emergency stop if the processor should get caught in a loop. For this purpose the turnon of the key triggers a pulse generator whose output triggers a one-shot that temporarily inhibits certain pulses in the instruction and address cycles; then the processor can stop at one of these points if a loop prevents its reaching the end of a main sequence. The pulse generator output also clears RUN in case a hardware malfunction should disable the normal clearing in the execute cycle. Both features apply only to the turnon of the key so that once it has been latched down it will not interfere with single step operation. Once RUN is clear, IO RESET can be used to clear the computer.

If the processor should hang up or be running without seeming to accomplish anything, do not call a DEC Technical Representative until certain routine checks have been made, as it is possible for an inept programmer to hang up the machine. Although it is recommended that only BLKI, BLKO, or JSR be used in a PI cycle, nonetheless the processor will perform a PI cycle correctly for any non-IOT instruction. But a condition IOT will cause the processor to hang up in the PI cycle. When this happens, the processor repeats the instruction over and over: PC is static, the PI REQUEST light for the channel is on but the PI IN PROGRESS light never goes on, and PI CYC and PI REQ at the top of bay 1 remain on indefinitely (the AC lights display the IOT instruction code).

If the program attempts to retrieve an instruction from a memory that is not connected to the bus (and the DISABLE MEMORY switch is off), the lack of any instruction retrieved is interpreted by the processor as a UUO. The attempt to address a nonexistent memory usually results in an interrupt on the processor channel, but if the JSR for the break should attempt to go to the same memory, the processor would go into a loop.

Both of the above loops include the complete main sequence, so pressing INSTRUCTION STOP will cause a stop at the completion of an instruction. There are other program failures, however, that never allow the processor to finish an instruction. If a program should include an XCT that executes itself or if a programmer puts a UUO in location 41, the processor goes into a loop that keeps jumping back to the instruction cycle without ever completing a main sequence. The key thus halts these in the address cycle. If DISABLE MEMORY is on and the program attempts any access to a memory that is not on line, the processor hangs up in the memory subroutine. Interrupting the time chain cannot affect this situation, but INSTRUCTION STOP clears RUN, and the machine can then be freed by IO RESET.

Hardware malfunctions can cause loops that the time chain inhibit cannot stop. For example if PI CYC fails to set, an interrupt request will cause a loop in which the processor keeps trying to honor the request without succeeding. If MQ0 refuses to clear in a block transfer, the processor will loop forever, returning on each step to the fetch cycle. For any loop that includes a memory call, do not free the processor merely by clearing RUN and press-IO RESET. If the processor is within the memory subroutine when the reset occurs, it will very likely hang up a memory while freeing the processor. For this situation it is preferable to press MEMORY STOP and then check the lights to be sure the processor has not stopped following the read part of a read-write access. If it has, hold on MEMORY STOP and press MEMORY CONTINUE, so that it will stop following the subsequent write. Once the processor has stopped with no chance of hanging up a memory, press INSTRUCTION STOP to clear RUN, and then press IO RESET.

CHAPTER 4

DRAWING CONVENTIONS AND FLOW CHARTS

Accompanying each PDP-6 is a complete set of drawings, consisting primarily of D-size flow charts, logic drawings (block schematics), and wiring diagrams. Every drawing is labeled with both a DEC drawing number and a type code. The drawing number is in four parts separated by dashes (e.g., D-166-4-EX): the first part is a letter indicating size; the second is the type number of the equipment (usually three digits); the third is the drawing serial number (see next paragraph); and the last is a number or a mnemonic letter code specifying the individual drawing (the code may end with a number, sometimes preceded by another dash, if more than one drawing is required to treat a section of the logic). If a drawing includes several sheets, both the sheet number and the number of sheets are written at the lower left of the drawing number. If a drawing is revised after being signed by the project engineer, a revision letter is written to the right. To the left of the number is a type code; some typical codes are block schematic BS, system diagram SD, flow diagram FD, timing diagram TD, interconnection diagram ID, cable diagram CD, wiring diagram WD, power wiring PW, module list ML, utilization module list UML, master drawing list MDL, cable list CL, wiring list WL (the last three are usually A size).

In general, the only drawings included in the manual for a given piece of equipment are the associated flow diagrams and logic drawings. The maintenance chapter of each manual does, however, describe the other types of engineering drawings and their use. Drawings in the manual are intended for instruction purposes only; personnel working at the machine should use the prints for the equipment rather than the figures for the manual. Drawings that are reduced to B size and printed in this and other manuals for the PDP-6 system are serial 0, corresponding to the standard production machine. Although every unit of a given type is assigned a different serial number, most of the prints accompanying the equipment have drawing serial 0. But if a particular unit differs in some way from standard, those drawings that reflect the difference have the same serial number as the lowest numbered machine that is so modified. Therefore, although each manual contains the drawings for that portion of the standard system that it describes, maintenance personnel should use the prints for work on the equipment because they show any variations peculiar to the installation.

All drawings included in a manual are assigned figure numbers by chapter. These numbers are also written on the prints in the lower right, above the drawing legend. To differentiate drawings associated with one manual from those of another, a code designating the manual appears in front of the figure number (the code may not be included on the figures reproduced in the manual). The letters "AP" indicate a figure for this manual, which includes not only the Arithmetic Processor Type 166 but also the control units for four in-out devices Types 461, 626, 760, and 761. Figures for the memory manual are prefixed "M" and show both the fast and core memories. Figure numbers on drawings for in-out devices described in separate publications are usually prefixed by the appropriate type number.

The complete system logic for the arithmetic processor and the common in-out devices is shown in a series of flow charts for Chapter 4 and a series of logic drawings for Chapters 5, 6, 7, and 8. This chapter describes the conventions and notation used in these drawings.

4.1 LOGIC DRAWINGS

The logic drawings are block diagrams that show the function of every logic element used in the computer. They also show the type of signal present at any module connector pin that carries a logic signal or some special voltage level. The standard power and ground pins (A to D on every module) are not shown. In addition to showing the function of every logic element, the drawings indentify every circuit by type and by physical location. Circuit type is always identified by the type number as given in the DEC module catalog. Below the type number is a location code made up of one digit, one letter, and one or two digits. For example, the location code 2F10 represents plugin unit connector 10 in mounting panel F in bay 2 (in the lettering on the logic drawings, each numeral "0" has a slash through it to distinguish it from letter "O"). Pin designations may be formed merely by adding the pin letter to the module location code, e.g., 2F10H.

The frame containing the arithmetic processor and the console includes bays 1 to 4. Each memory, or peripheral device requiring a major portion of a bay, is designated as bay 1. If an in-out device control unit requires only a few logic mounting panels, it has no bay number; the panels are designated A, B, ..., even though they may be mounted in any position. For the smaller block symbols, such as those representing single inverters and capacitor-diode gates, the circuit type number and location code are written near the symbol, and the inputs and outputs are labeled by connector pins. With all larger blocks, the circuit and location information are written inside the block. If several logic elements from the same module appear together in a drawing, they may be enclosed in a dashed line: the location and type number are then written only once within the module boundary and the pin letters are written just inside the boundary where the signal lines cross it.

Some modules have connectors on both front and back; pins on the rear connector are identified by the prefix "R." Some modules are double height, with two front connectors. The location of such a module is given by two panel letters, e.g., 2DE17, and the front connector pins are prefixed by the appropriate panel letter (pins on this module would be designated 2DE17-DT, 2DE17-EB). Only the upper connector receives the power lines, so pins A to D of the lower connector are available for logic signals. Such modules usually also have two rear connectors, the upper one identified by the prefix "R," the lower by "S." Thus for a double-height module mounted in 2DE, the R connector is at the rear of panel D, the S connector at the rear of panel E.

On the logic drawings, the type of signal present at a pin connection is shown by a triangle or diamond. In DEC convention, timing is provided by pulses whose polarities are shown by open and closed triangles. These polarities depend only upon input requirements and represent no logical difference. Similarly, gating levels are represented by open and closed diamonds that represent the assertion polarity that satisfies the gate; neither voltage level categorically represents 1 or 0, true or false. A given logic function may have different assertion levels in different places depending upon gate input requirements. For example, if a function has a negative assertion level, the function is considered true when the line corresponding to it is at −3 vdc; for ground assertion, the function is considered true when the line is at ground. Sometimes a line carrying a logic level is shown connected to the input of a pulse amplifier or capacitor-diode gate, which produces a pulse output. In these cases, the output is triggered by a level transition at the input. If the input is shown as a diamond, triggering occurs at the leading edge, i.e., the diamond shows the assertion polarity of the logic function immediately after the triggering transition. An event triggered on a trailing edge is indicated by a composite symbol with a diamond showing the assertion polarity of the level, and a triangle showing the opposite polarity required for the input pulse (i.e., the triangle indicates the direction of the transition when the logic level is negated). Sometimes a leading edge is shown by a composite diamond and triangle of the same polarity, but this is not necessary. Occasionally in the in-out equipment, a wide pulse is used to produce a delay by triggering events on the pulse trailing edge. The composite signal for this is a pair of triangles, the first showing the pulse polarity, the second the polarity of the triggering transition. Any nonstandard signal is shown merely by an arrow pointing in the direction of signal flow.

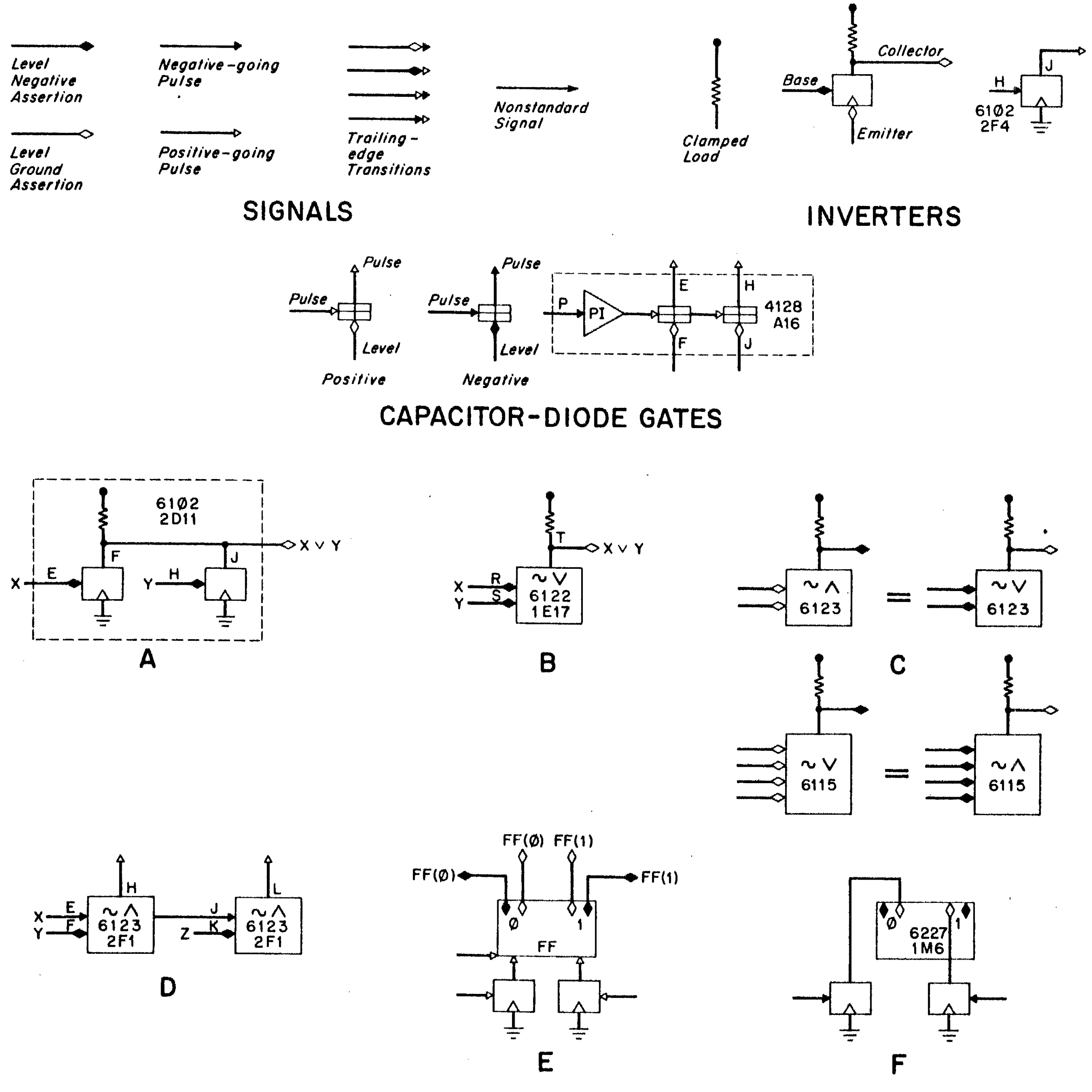


Figure 4-1 Logic Symbols

The upper part of Figure 4-1 shows the signal designations and the symbols for inverters and capacitor-diode gates. These are described fully in the introduction to the DEC module catalog. With the symbols in Figure 4-1 are examples of type numbers, location codes, and connector pin letters. The remaining lettered sections of Figure 4-1 show other conventions for the PDP-6 drawings.

The logic drawings show the function of every logic element in the simplest way consistent with the requirement that every pin connection be shown. Thus if two single inverters are connected to form an AND or OR gate, the individual inverters are shown in the drawing as indicated at A in Figure 4-1. However if the gate is produced by connections internal to a module, such as a pair of inverters with a common collector pin and internally grounded emitters, then it is shown by a block labeled for the appropriate logic function as shown at B. Blocks are used to represent inverter gates with as many as four inputs and diode gates with as many as eight (the in-out equipment uses capacitor-diode modules in which the gates have common pulse connections, but the individual gates are always shown). Since all such nets invert (a diode gate includes an inverter output) the blocks are always labeled "∿∧" or " ∿∨." The tilde (∿)in these labels has no actual connection with logical function and may be ignored when learning the logic. Because an assertion level may be inverted without affecting its truth value, all gates are AND (∧) or OR (∨) gates, and the tilde merely indicates that the output assertion level is of opposite polarity to the inputs. Each block also contains a location code and a type number; input and output pin connections are labeled in the usual way. Since the logical function of a gate depends both upon its logical configuration and the assertion polarities of its inputs, a given net may be used as either an AND or an OR gate—the equivalence is shown at C. Of course, a single level input may be replaced by a pulse in an AND gate, and all levels may be replaced by pulses in an OR gate. Inputs to a gate are generally at the left, outputs at the top, and a signal shown passing through a gate from left to right implies no logical change: D shows a pair of AND gates in which the pulse X, though labeled only once, is an input to both gates, whereas levels Y and Z each enter only one.

All other circuit elements except flip-flops appear as blocks that contain a mnemonic abbreviation of the circuit function. Some examples are delay DLY, pulse amplifier PA, pulse generator PG, clock CLK, bus driver BD, solenoid driver SD, majority gate MAJ, dc adder DCA.

Clocks, pulse amplifiers, pulse generators, and some one-shot delays have a pair of transformer-coupled pulse outputs (usually shown at the right of the block). When the input to such a circuit is triggered, a positive-going pulse appears at the positive output if the negative output is grounded, or a negative-going pulse appears at the negative output if the positive output is grounded. One-shot delays other than the 4303 have a logic level output that is asserted negative during the delay period. The symbol that represents a flip-flop is shown at E (the same type of symbol also represents the 4303 Integrating Delay and the 6131 DC Adder). In this rectangle, terminals S and T are drawn twice, showing the polarities associated with either state of the flip-flop. In normal convention the "0" is at the left and the 0-out terminal is represented by the left diamond in both pairs. Some flip-flops have a separate output terminal, represented by a fifth diamond, to drive an indicator.

The principal advantage in showing the two states at both assertion levels is that there is never any need to invert the name of a signal that appears as an input to a logic net: all logical conditions appear in the drawings with correct truth values. When a flip-flop output is used as the input to a logic net, the signal name indicates the enabling state of the flip-flop. To determine the physical source of the signal (the output terminal to which the signal line is connected), one must know both the signal name and the assertion level. For example, the signal FF(1) at negative assertion originates at the 1-out terminal of flip-flop FF; at ground assertion this signal actually originates at the 0-out terminal.

Two gatable inputs are shown at the bottom of the rectangle at E, with the 0-in terminal at the left. Direct pulse inputs (i.e., those that are not gated) are at the sides of the flip-flop: in the example a direct clear input is shown at the left. A flip-flop may also have a complement input, which is drawn at the bottom center. An unbuffered flip-flop may be set or cleared by grounding one of the flip-flop collectors; such a function may be represented in either of the ways shown at F.

Connections between flip-flops are shown in various ways on the logic drawings, always in the simplest way consistent with showing all pin connections. The clear line for an entire register is usually shown entering the lower left corner of the leftmost flip-flop, then out of the lower right corner on to the next flip-flop, and so on through the register. If the flip-flops in a counter have count inputs and outputs, the count pulse is usually shown going from right to left, entering the center of the right side of each flip-flop and leaving at the center of the left (e.g., see the program counter, Figure 5-11). If flip-flops from different modules are connected for shifting, all of the shift gates must be shown; however, if the flip-flops within a module are connected internally as a shift register, the shift signal is shown in the same way as a count (e.g., the shift register modules in Figure 8-7). In all processor registers, most transfer input gates for a given flip-flop are included on the module containing the flip-flop. These gates are therefore shown as small rectangles with logic symbols and pin connections but without location codes or type numbers (refer to AR, Figure 6-5). In many instances no pin numbers are included because the input connections are internal to the module: e.g., the AR, MB, and MQ flip-flops are on the same modules and the connections shown between them have no pin numbers. If these internal logic gates are similar to the regular inverter and diode gates, i.e., if all inputs to a given gate are of the same polarity and the output is inverted, the block is labeled with the appropriate symbol, either "∿∧" or "∿∨." However, in many cases there are nonstandard gates, e.g., one in which a ground level is gated by a negative pulse to produce a ground output. Such gates are labeled merely by logic function, "∧" or "∨," and no attempt is made to indicate signal polarities other than the diamonds and arrows used for inputs and outputs. In some cases pulse inputs to individual bits of a register are made through NPN emitter followers. Since these perform no logical function and do not even change signal polarity, each is shown on the logic drawing merely as a small circle at the pin to indicate that the signal at the flip-flop input terminal through other input gates is not available at that pin.

State changes in the 10-mc flip-flops take place more rapidly than the duration of the input pulses. To compensate for this, many gated inputs to these flip-flops are made through delay elements, which are not shown on the logic drawings. Since in many instances the outputs of a register flip-flop condition its inputs, the flip-flop state change is delayed until the termination of the input pulse to prevent logical race problems.

In addition to the many modules containing flip-flops, pulse amplifiers, etc., there is also a hybrid module, the subroutine card SBR, which includes three circuits each containing a flip-flop and a gated pulse amplifier. The flip-flop 1 output is an input to the AND gate at the PA input; the PA output, besides being available at a connector pin, is connected internally to the flip-flop clear input (see the SBR at the left in Figure 5-2). An SBR (or its equivalent) is used whenever a subroutine is called from any time chain. The same pulse that triggers the subroutine also sets the flip-flop in an SBR, enabling the input gate to the PA; the other input to the PA gate is the return pulse from the subroutine. At the completion of the subroutine, the return pulse triggers the PA, whose output both clears the SBR and restarts the time chain.

All logic drawings are laid out with rectangular map coordinates, numbered 1 to 8 from left to right and lettered A to D from top to bottom. Because a single drawing may contain a number of logic elements, coordinates are often included in figure references. For example, a reference to the circuit in "Figure 5-6B6" would mean the circuit located in block B6 of logic drawing 5-6.

4.2 SIGNAL NOTATION

All signal names in PDP-6 are mnemonics that indicate both the function of the signal and its source. Each register with associated logic and each control system, whether it occupies several drawings, one drawing, or only part of one, has a single mnemonic code of one to three letters, which appears in the drawing title and at the beginning of the name of any signal originating in this part of the logic. This source code may appear naturally as part of the signal name; if not, it is merely prefixed to the name. For example the arithmetic register AR and its associated transfer logic, flag logic, and AR subroutines require a number of drawings all with prefix code AR, and the pulse that shifts the contents of AR to the left is AR SH LT. On the other hand, the readin mode subroutine flip-flop, which is associated with read in and is shown on one of the drawings for the key logic, is designated KEY RIM SBR. All prefix codes and corresponding figures are listed at the left in Figure 4-2.

The name of a signal that transfers information from one register to another includes the names of the two registers with an arrow between them. The arrow invariably points to the left because the transfer logic is always associated with the receiving register and its name must therefore appear first in the signal designation. The name of a transfer signal specifies not only the registers and the direction of transfer, but also the type of transfer and the register bits involved if the signal acts on less than the entire register. Numerals representing register bits are merely appended to the register name; bit 8 in AR is AR8 and bits 0-7 in MB are MB0-7. Operations that affect only half of a full-word register are indicated by appending LT or RT to the register name. The state of a flip-flop is represented by a numeral in parentheses, e.g., the 1 state of bit 8 in AR is AR8(1). Since transfers in effect transfer states, the type of transfer is also indicated by a symbol in parentheses following the register name. For example the signal that transfers all zeros into the bits of AR, i.e., the signal that clears AR, is AR ←(0). This action of course is not usually referred to as a 0 transfer but rather as the clear function. The actual transfer of zeros from register A to register B is as follows: the transfer pulse clears a given bit of B if the corresponding bit in A is in the 0 state. The pulse that produces this effect is B←A(0). Since B now contains ones only in bit positions that originally contained ones and also correspond to ones in A, the 0 transfer therefore produces in B the AND function of A and B. Similarly, the transfer of ones from A to B, B←A(1), produces the inclusive OR function. If B is cleared before the transfer, then after the transfer A and B both contain the same information and the pair of pulses B ←(0) and B ←A(1) transfer the contents of A to B. The same effect would be produced by setting all bits in B and then transferring zeros. If both the zeros and ones of A are transferred to B simultaneously so that B bits corresponding to zeros in AR are cleared and those corresponding to ones in AR set, no prior setting or clearing is necessary. This is a jam transfer and is written B←A(J). Since the jam transfer occurs at a single point in time, it is possible to switch the contents of two registers: the outputs from A can provide the gating levels for the transfer into B while the B outputs gate the transfer into A. The signal that triggers both transfer pulses at once is labeled B(J)↔A(J).

There are other types of transfers, such as B(0)←A(1), which clears bits of B that correspond to ones in A, i.e., transforms ones into zeros. The pulse that complements a given bit in B if the corresponding bit in A is 1, produces in B the exclusive OR function of A and B, and is written B ←A(∀). Most other types of signals have fairly obvious names: MQ SH RT shifts the word in MQ to the right (a shift is a jam transfer from one bit to another in the same register); AR COM complements the word in AR. We have been discussing pulse signals so far but the names of gating levels are also quite descriptive: PC+1 INH inhibits incrementing the program counter; FC(E) PSE causes memory control to fetch the word in location E and then pause to wait for a restart for subsequent storage; SAC2 causes the store cycle to deposit a second accumulator. AC0 always refers to accumulator 0, but AC2 refers to the accumulator following the one addressed by the instruction (if the instruction AC address is 17, AC2 refers to accumulator 0). No more than two sources can have the same signal name, and any pair must have opposite polarities so that the source of every signal is uniquely identified by its name and the associated polarity symbol. Whenever two logically equivalent signals have the same polarity, they are differentiated by adding an extra letter or number to one of the signal names; for example, MR CLR and MR CLR A are equivalent pulses. If one logic signal produces an equivalent signal through a bus driver, the buffered signal is indicated by the letter "B" at the end of the signal name. If the outputs of a flip-flop are buffered externally to the flip-flop module, the buffered signals are indicated by a "B" between the flip-flop name and the state numeral.

The pulses in the time chains for the various main cycles and subroutines all have the same three-part format: first the prefix code naming the chain, then the letter "T," followed by a number or number and letter combination specifying the pulse. For example, FT3 is pulse number 3 in the fetch time chain; DST13 is pulse 13 in the divide subroutine time chain. These pulses are not always in exactly the order that one might expect, and the reader should always consult the flow charts to determine the proper sequence. For example the first three pulses in normalize return are NRT0.5, NRT0, NRT0.1. In the execute cycle, ET0 and ET0A are logically equivalent and together are the first pulse in the execute cycle—the two labels indicate separate but equivalent pulse lines. The next pulse is ET1 but this is followed by ET3. In character operations CHT8B follows CHT8 but precedes CHT8A. But in most cases a letter following the number in a pulse name indicates the next pulse in the chain, usually following return from a subroutine. Thus the first pulse in the floating multiply chain, FMT0, calls the exponent calculate subroutine, and the next pulse, triggered by the subroutine through an SBR, is FMT0A. The SBRs and most control flip-flops that govern the time chains also have similar three-part names in which the "T" is replaced by an "F."

4.3 INSTRUCTION DECODING

Figure 4-2 is a tree which shows the decoding of instructions from the instruction register. The output signals in the figure appear as the gating levels in the flow charts that are described in the following sections. The purpose of the tree is to allow the reader to gain familiarity with these logic gates and to correlate them with the instruction codes. No attempt is made here to give a detailed explanation of every logical function; for this the reader should use the tree in conjunction with the discussion of the decoding hardware in 5.3.

Any code placed in IR is converted into gating levels to govern events that must occur in the fetch, execute, and store cycles and various special sequences to execute the particular instruction. The decoding begins at the left in the figure with the three most significant IR bits, which are converted into signals representing the eight instruction classes. These primary outputs act as gates to enable decoding of the remaining bits. In some cases the output enables a second decoder for several more bits whose outputs in turn act as gates for further decoding. In others the first output represents a single instruction group and it gates the decoding for all remaining bits. For the former type, the line extends only part way across the figure and then generates a number of branches at a single position; for the latter the line extends the length of the tree, and branches appear at several positions. Most of the groups of two, three, or four bits shown together are decoded by binary-to-quarternary or binary-to-octal decoders.

Groups of bits are actually decoded into individual outputs only where signal names are shown for each bit configuration. Where only parenthetical items are listed, the coding as shown produces the mode or action listed, but the hardware does not decode the bit configuration into individual outputs.

When IR receives an instruction code in the instruction cycle, the first three bits are decoded to generate one of eight primary command levels shown at the left in the figure. At the top is the decoder output for a UUO, which corresponds to three zeros in bits 0-2. The decoder output is conditioned by a flip-flop to prevent the generation of the actual command level while IR is clear awaiting an instruction code. Other conditions also generate the command level to perform a UUO when a user program attempts an illegal instruction. At the bottom of the drawing is the decoder output for an IOT, corresponding to three ones. If it is not a user IOT, the command level is generated and causes the decoding of bits 10-12 into the eight IOT instructions. Some of these individual instructions are ORed to generate composite functions. At the same time IR bits 3-9 are placed on the I/O bus to select the device.

Between the top and bottom entries in the figure are the decoder outputs for the six instruction classes that use the basic format (these are not in numerical order, but are instead listed so that classes with common decoding are adjacent). The configuration 001 indicates the floating-point instructions and character operations. If the second octal digit in the code is 0, 1, or 2, there is no further decoding as these 24 codes are not used. If bits 3-5 contain 111, IR6-8 are decoded for two unused codes, the five character operations, and the single instruction floating scale. A 1 in bit 3 indicates the floating-point instructions and for these, bits 4 and 5 are decoded for the specific instruction, bits 7 and 8 for the mode. The logical condition for rounding (NR ROUND) is dependent upon bit 6, but is not dependent upon any command level because the gate is used only by the normalize return subroutine, which is called only by floating-point instructions.

The second basic instruction class corresponds to the primary command level IR 2XX, which represents a number of small instruction groups. Either 0 or 1 in the second octal digit specifies a full-word transfer for which bits 7 and 8 are decoded for the mode, and specific configurations of bits 5 and 6 generate the levels that control swapping and negating of the word. Second octal digits of 2 and 3 correspond to fixed multiply and divide, respectively. The 01 configuration of bits 3 and 4 (shown as 01-) generates a composite multiply-divide level to enable the net that determines the necessary fetch and store operations from bits 7 and 8. Further fetch and store operations are determined separately for multiply and divide from the state of bit 6, which specifies whether the operands are to be treated as integers or fractions. For division bit 6 is actually decoded into a further pair of control levels, but for multiplication the outputs of IR6 act as gates directly on the multiply sequence. The next three configurations of IR3-5 enable the decoding of IR6-8 for the shift operations, a group of miscellaneous instructions, and the pushdown and jump instructions (the absence of a signal name by any configuration of IR6-8 corresponds to an unused op code). The decoder output for JRST, like IOT, generates the true command level only if it is not executed as a UUO. For the JP group there are two additional control levels corresponding to the six instructions that actually execute a jump, and the three instructions that store the miscellaneous bits. The last group of instructions in IR 2XX are fixed addition and subtraction wherein IR6 determines the instruction, and bits 7 and 8 are decoded for the mode. These modes are equivalent to those for the Boolean instructions, which make up an entire class with the configuration 100 in IR0-2. IR BOOLE also enables the decoding of IR3-6 to determine which of the 16 instructions is specified.

Another single instruction group, the half-word transfers, is specified by 101 in the first three IR bits. For these instructions bit 3 determines which part of the destination register shall receive the half word. Bits 4 and 5 determine the effect on the nontransfer half: a 1 in either bit clears AR, and further decoding enabled by bit 3 determines whether that half shall then be set to all ones. A 1 in bit 6 causes the operand to be swapped before the transfer so that the destination register receives the half word from the opposite half of the source register. Bits 7 and 8 are decoded for the mode.

The remaining two IR0-2 configurations, 011 and 110, are for the arithmetic compare instructions and the logical compare instructions (ACBM). The former class includes the ACCP group in which an accumulator is compared against either C(E) or E, and the MEMAC group in which either AC or C(E) is compared against zero. For the arithmetic class, 00 in IR3-4 specifies the ACCP type, and a 1 in bit 5 indicates a direct comparison, i.e., against C(E); the other three configurations of bits 3 and 4 specify the types of MEMAC instructions, any of which generates the composite MEMAC command level. The state of IR5 determines whether the comparison tests AC for a jump or memory for a skip. In the ACBM group, bits 3 and 4 determine the action on the masked bits. A 1 in bit 5 specifies that the mask is C(E), and a 1 in bit 8 swaps the mask before the test. The remaining bits in the two classes determine the skip or jump condition as shown; for the logical comparison the condition is determined by bits 6 and 7, for the arithmetic comparison by bits 6-8. The level for ACCP or MEMAC appears in the term for bit 8 because this part of the condition is not used by the ACBM instructions; otherwise no instruction levels need appear because the test level generated is used only by these instructions.

4.4 FLOW CHARTS

The remaining figures in this chapter are flow charts of all operations that can be executed by the arithmetic processor and the four common in-out devices. These figures show every event, and in so far as possible, show the flow of operations in a manner that is equivalent to the actual gating and timing in the hardware (the terminology is from the logic drawings unless italicized). Certain intermediate pulses are shown only implicitly (for example ET1 is shown clearing AR, when actually it triggers AR ←(0), which in turn triggers ARLT ←(0) and ARRT ←(0), which together clear both halves of AR). If an event is prefixed by CFAC, the time pulse is routed via the subroutine interface to the register gating.

Each flow chart is based on a sequence of time pulses shown along a vertical line. Except for insignificant intervals, such as the delay across a pulse amplifier or inverter, time between pulses is shown by breaks in the line. Pulses always appear in ellipses and events in rectangles.

A pair of single horizontal lines breaking a vertical line indicates a delay; between the lines is listed the delay time, or the signal that must be asserted to continue the flow. A break shown by double lines indicates a subroutine call; the upper term identifies the subroutine, the lower term names the pulse returned by the subroutine to restart the calling sequence. A flow line that terminates with an arrow indicates that the flow continues with the pulse listed below the arrow (numbers in parentheses are figure references); a line that terminates in an ellipse indicates that the flow along this path ends with the events associated with the pulse. In a rectangle, a condition written to the left of a colon must be satisfied in order for the event written to the right to occur. A condition written on a line must be satisfied for flow to continue along the line. When several vertical lines branch from a horizontal line, the conditions are written above the vertical lines.

The key cycle, through which initial entry into processor operations must be made, is shown at the left in Figure 4-3. Key functions that require a sequence of events are shown in columns associated with the key time pulses. Some console operations, such as examine and deposit, require only the key cycle; others, such as start and read in, use the key cycle to provide entry to the main sequence. Figure 4-4 and 4-5 show the main sequence. The flow charts for the instruction, address, fetch, and store cycles show all possible events; the execute flow chart shows the main time chain for the cycle, including all exits to special instruction sequences and all operations that are not produced by any instruction, e.g., clearing AR CRY0 at the beginning of the cycle. The basic memory and AR subroutines can be called from the main sequence and are shown, respectively, in Figure 4-3 and at the right in Figure 4-8.

The remaining figures show the execution of instructions following the fetching of the operands. Those instructions that are timed by the execute cycle are shown in tables in Figure 4-6, 4-7, and at the left in 4-8; instructions, other than IOTs, that are executed by special sequences are shown in Figures 4-8 to 4-10. These special sequences may call the basic SC subroutines (Figure 4-8, right) and the special arithmetic subroutines shown in Figures 4-11 and 4-12. Figure 4-13 shows the IOT instructions, some of which require the entire execute cycle as well as the special IOT sequence. In-out operations that take place outside of the processor are shown in Figure 4-14; for each of these an IOT is required at the beginning or the end, but otherwise the sequence goes on independently of processor operations.

For every instruction, Figures 4-6 to 4-10 and 4-13 list the complete decoding, the instruction action, the initial registers, and the initial and final gates. The decoding is shown as performed by the hardware; whenever the decoding does not correspond exactly to the meanings of the bits in the instruction codes, further explanation is given in italics. The action is given in the programming sense (i.e., in terms of accumulators, memory locations, control registers, flags), but in the notation of the logic drawings: AC may mean either an accumulator or its contents, E may mean either an address or the addressed location; the meaning is evident from the context. The states of the processor registers are shown following the fetch cycle; MA always contains the effective address and is not shown. The initial gates are those that control the fetch and execute cycles. It should be understood that PC is incremented at ET0 whenever the PC inhibit is not shown as an initial gate, unless there is some special situation (e.g., the processor is in a priority interrupt cycle or the instruction is being executed from the console). For all instructions that are executed by the execute cycle, or return to it, the final gates (i.e., those that control the store cycle) are listed at the end of the sequence. In all cases, registers and gates are listed in terms of the instruction modes even though the modes for fixed multiply and divide are not decoded individually.

Since the complete timing for the execute cycle is defined in Figure 4-5, those instructions whose events are timed entirely by the execute pulses are shown in tables. In a table entry for ET3, double vertical lines indicate the call of an AR subroutine; the subroutine is at the left, the return pulse at the right. The pulses from ET6 to ET9 are omitted when the instruction does not use the second half of the cycle.

4.5 EXECUTE CYCLE FLOW

Every main cycle, subroutine, and special instruction sequence is explained with the description of the logic that generates the time chain and gating levels for it. However, for instructions that are performed by the execute time pulses it is impossible to consider the multitude of possible events in the description of the execute cycle. For most such instructions the reader can easily determine how the specific sequence of events produces the desired result by inspecting the tables in Figures 4-6, 4-7, and 4-8. A few of the less obvious sequences are, however, described here (in the following it is assumed that the reader is familiar with the action of each instruction).

In a half-word transfer (Figure 4-6, left) the positions of the source and destination words after the fetch cycle depend upon the mode; i.e., the fetch cycle always places an accumulator in AR and a word from memory in MB, and the mode specifies which of these is the source and which the destination. The execution of the instruction requires that the source word be in MB and the destination in AR, so ET0 switches MB and AR for the memory mode and transfers MB to AR for the self mode. If the transfer is to be from one half of the source to the opposite half of the destination, then ET1 swaps the two halves of the source in MB. If the instruction is to perform any operation on the unused half of the destination, ET1 also clears AR. Then ET4 jams the specified half word into the appropriate half of AR and completes the action on the other half by setting it, if required. At ET10 the result is transferred from AR to MB for either the memory or self mode, i.e., those modes that require the store cycle to deposit the result in location E. In most instructions the result is in AR at the end of the execute cycle and thus any mode requiring storage in E usually includes the AR to MB transfer at ET10. Exceptions include instructions such as EXCH in which the correct words are already in MB and AR at ET0.

In a full-word transfer, ET0 performs whatever transfer is necessary so that the word to be moved is in both MB and AR. Then if the halves are to be swapped, the swap is made by ET1 in MB; if the word is to be negated—which may happen for either MOVN or MOVM—the AR negate subroutine is called at ET3. By ET4 the result is in MB only for MOVS, so for this instruction MB and AR are switched. Of course only a transfer from MB to AR would be necessary, but the gate for the bidirectional transfer is required for other instructions (no instruction requires a transfer limited to one direction). The other instructions in Figure 4-6 are quite straightforward with the possible exception of the logical compare group, ACBM. At the end of the fetch cycle, the word to be used as the mask is in MB, and the word to be tested in AR. For convenience let us refer to the latter as the data word, and the AND function of it with the mask as the test word. ET0 completes the construction of the mask by swapping the two halves of MB, if this is necessary. ET1 then produces the test word by transferring zeros from AR to MB, and at the same time it adjusts the data word for the complement and set actions by complementing bits of AR corresponding to ones in MB (the exclusive OR function) or transferring ones from MB to AR. ET4 then moves the test word to AR and the data word to MB. At this point the clear action has not been handled and it would seem that the mask has been lost. However, any ones now in AR must be a subset of the ones in the original mask, and these correspond exactly to all the ones within the masked bit positions in the data word in MB. For the clear action the next three time pulses clear these bits by complementing AR, transferring zeros to MB, and complementing again to restore the test word. At ET9 the test is made according to the contents of AR, and the adjusted data word is transferred into AR at the same time so that it is available for subsequent storage in AC. The bidirectional transfer listed in the table is not necessary, but is available because of the requirements of other instructions. The store cycle deposits the data word in AC unless it has not been changed from the original.

In JRST (Figure 4-7), the second half of the execute cycle performs a series of operations that seems to be pointless, since the store cycle does not deposit anything. Pulses 5 and 6 transfer the current program address from PC to MB, and after the jump address has been transferred to PC, pulses 9 and 10 transfer MB to MA. If the program does continue, these operations are pointless, but the JRST can halt the processor and may be used for error halts in maintenance programs. When the processor stops, PC displays the jump address and MA displays an address one greater than that of the location that contained the JRST.

CHAPTER 5

MAIN SEQUENCE CONTROL

This chapter describes the processor response to console inputs and explains the way in which the main control elements operate on instructions. Of the five major cycles, four are described in detail here: retrieval of an instruction from memory and decoding it from the instruction register; calculation from memory and decoding it from the instruction register; calculation of the effective address; retrieval of the operands; and storage of the result. The execute cycle is discussed only to the extent of basic operations, and timing events required for the execution of individual instructions are described in other chapters. The many arithmetic and data subroutines are described in Chapter 6 with the arithmetic registers; Chapter 7 describes the memory subroutine and memory address and data transfers; Chapter 8 includes the decoding of IOT instructions and control over transfers between the processor and input-output equipment. The various test conditions that affect the program sequence are discussed here, but any arithmetic or logical operations necessary to produce them are described in Chapter 6.

5.1 CONSOLE CONTROL

In general, the console switches provide continuous control levels that gate certain operations within the processor; whereas the keys are momentary contact switches that allow the operator to trigger specific sequences of events or to stop the processor. The hardware that provides control and timing for the various console operations is shown in Figures 5-1 and 5-2 (flow chart, Figure 4-3).

When the operator turns on computer power at the console, the 4303 Integrating Delay in the lower right of Figure 5-1 comes on in the 1 state, turning on a clock that provides a string of master power clear pulses. These pulses clear RUN so the processor cannot begin operations inadvertently, and they also trigger the master clear and the master start (Figure 5-2, right). MR START clears the flags and the in-out system including all flip-flops in the priority interrupt system and the processor I/O interface; MR CLR clears the major control registers and flip-flops in the processor proper including all SBR flip-flops. The master clear also occurs at the beginning of every new operating sequence, i.e., at the beginning of every console operation other than memory continue (see below) and at the initiation of any new instruction (IT0 or any equivalent pulse). The master start is generated only by the power clear and when the operator specifically wishes to clear the I/O system or place the processor in the executive mode. The flags and several special control flip-flops are cleared only by MR START because they must carry information over from one main sequence to another.

The left half of Figure 5-1 shows the logic inputs from the keys and switches, and nets that generate a number of composite functions to control events common to two or more console operations. Several of the nets AND key functions with a state of the RUN flip-flop. For example, a net in A3 allows the input from the EXECUTE key to affect the processor only if it is not in operation. Since examine and deposit may be performed while the processor is running, these two operations are governed not by levels from the keys but rather by two pairs of sync and start flip-flops (upper right). A pulse in the key time chain, KT0A, normally clears all four flip-flops, but for examine or deposit the appropriate sync flip-flop is set instead. Then, if the processor is running, the corresponding start flip-flop is set at the beginning of the execute cycle. The sync flip-flops provide gates for the events in these operations; the start flip-flops control the insertion of the operation between two main sequences. For this purpose, two of the nets at the left generate the AND functions of RUN (1) with functions of the start flip-flop states. If at the end of a main sequence, either flip-flop has been set and the processor is to continue, the signal from the net in C5 causes the processor to enter the key cycle at the end of the main sequence. However, if neither flip-flop has been set, the signal from the net in C3 causes the processor to begin a new main sequence.

Pushing any key other than the two STOP keys generates the level KEY MANUAL through the net in the top center of the figure (the entire net is disabled during the power clear period). The turnon of this level triggers a pulse generator in the upper left of Figure 5-2 to start the key time chain. This chain is broken between KT0A and KT1 by the condition RUN (0) so that KT0A triggers the following triplet of pulses only if the processor is not running. There are, however, two gates that bypass the RUN condition: if the operator selects EXAMINE or DEPOSIT while the processor is running, the appropriate start flip-flop allows the final pulse in a store cycle to trigger KT1; furthermore memory continue goes directly from KT0A to KT1 because it is assumed that the processor is "running"—i.e., RUN is 1—but that the operator has previously stopped it at the end of a memory cycle by inhibiting the memory subroutine return. This is why memory continue is the only operation in which KT1 does not generate the master clear: somewhere an SBR is legitimately waiting for the subroutine completion. Memory continue triggers the MC restart at KT1.

For read in, KT1 sets the readin mode subroutine flip-flop shown at the lower left. When the processor is operating in readin mode, the normally unused core registers at the bottom of memory replace the 16 flip-flop registers of the fast memory. In all other respects, read in is exactly equivalent to start; thus the operator may keep a loading program stored permanently in the readin area of memory, and the processor switches to normal operating mode whenever an instruction is taken from any location beyond 17 (∿ MA18-31 = 0). For console operations other than read in, KT1 clears the flip-flop for normal operation, but the clear is inhibited by the CONTINUE keys so that READ IN can be single stepped (the clear net is in Figure 51D1).

A few of the functions produced by the key time pulses are shown on drawings associated with the appropriate function (such as operations involving PC) but most are shown in the center portion of Figure 5-2. Start, read in, examine, and deposit all clear MA and transfer an address into it from the console ADDRESS switches. Examine next and deposit next, instead of loading MA, increment it. The three operations that send data into the computer, deposit, deposit next, and execute, clear AR and load it from the DATA switches. Also shown in the key drawing is the gate through which the DATAI for the processor loads the DATA switches into AR. CPA indicates that the IOT has selected the processor and the assertion of the DATAI level triggers the transfer PA (see 8.3).

Following the various setup operations, start, read in, and instruction continue all generate KEY GO (upper right). This pulse places the processor in operation by setting RUN (Figure 5-2) and starting the instruction cycle. As long as RUN remains set, the completion of every store cycle triggers a new instruction cycle so that the processor performs one instruction after another in the program. Either the operator or the program may stop the computer at the end of a store cycle by clearing RUN at ET0. The executive program clears it by a 1 in bit 10 of a JRST; a user program cannot stop the processor. The operator clears it by pressing the INSTRUCTION STOP key, but the logic gate is bypassed by a pulse generator that is also connected to the key. The action through the PG is slow compared to the processor time chains, but this ensures that RUN will be cleared even if some malfunction should interrupt the normal procedure.

The other console functions do not place the processor in normal operation. Execute transfers AR to MB at KT3 and enters the instruction cycle at the point at which an instruction would normally be in MB after retrieval from memory but it does not set RUN, so the processor stops at the completion of this single instruction. In the four examine and deposit operations, KT3 requests a memory read or write as required and sets KEY RE/WR (Figure 5-2 B6, C1). To deposit information in memory, KEY WR also transfers AR to MB. Upon completion of the memory subroutine, the MC restart generates the read/write return clearing the SBR.

The upper right portion of Figure 5-2 shows the gates that generate KT4 for the repeat mode. If the REPEAT switch is on, KT4 triggers a repeat delay (upper left): the 4303 state change at the end of the delay period then retriggers the key time chain as long as any initiating key is held on. To use the memory repeat function, the processor must already be running when the operator presses the STOP key for memory; then KT4 is generated every time the memory stop flip-flop is set. To repeat start, instruction continue, or read in, the operator must hold on both the initiating key and the INSTRUCTION STOP key; then at the completion of each instruction, ST7 triggers KT4. Since execute does not start the processor, the operator need not use the STOP key to repeat it; in this case, KT3 triggers KT4. Similarly, for the four examine and deposit operations, the read/write return triggers KT4. Furthermore, examine or deposit may be repeated while the processor is running: if one of these operations interrupts the normal transition from store cycle to instruction cycle, the appropriate start flip-flop causes KT4 to trigger KEY GO. This pulse does not affect RUN, which is already 1, but it does retrigger the instruction cycle and clear the start flip-flops.

5.2 PROCESSOR CYCLES

There are five cycles in the main sequence: instruction, address, fetch, execute, and store. All but the execute cycle perform only operations that are common to all instructions or to groups of instructions, and these four are described in detail in this section. Instruction, fetch, and store can call only the memory subroutine; whereas the address cycle can call both the memory and the add subroutines.

The execute cycle performs some general control operations but is limited primarily to the specific operations necessary for the execution of individual instructions, including entry into special instruction sequences. For this reason, the description of the execute cycle included here is limited to timing, general control functions, the calling of the simple AR subroutines, and the entry and return for special sequences.

In the fetch, execute, and store cycles, the gating levels that govern the sequence of time pulses and the calling of subroutines are OR functions of instruction conditions. An input condition may be an entire instruction class or a single instruction, or it may be a single mode either within an instruction or in an instruction class. The reader is assumed familiar with the standard outputs from the instruction decoders (4.3) and only special conditions are discussed here.

a Instruction

The lower half of Figure 5-3 shows the time chain that controls the retrieval of instructions from memory (flow chart, Figure 4-4). When the operator starts the processor or causes examine or deposit to interrupt normal processor operations, KEY GO begins a new instruction cycle by triggering IT0. If the processor is in normal operation and no examine or deposit has been synced by key timing, the final pulse in the store cycle of an instruction starts the instruction cycle of the next one. IT0 generates the master clear, clears the memory address register, and after a slight delay sets IF1A, the SBR for the memory subroutine that will subsequently retrieve an instruction. IT0 also generates PI SYNC (8.2b) which strobes the priority interrupt system provided the processor is not already in a PI cycle. If the PIR strobe produces no request or if the processor is already in a PI cycle, the sequence continues to IT1 directly. If the strobe does produce a request, the sequence continues instead to IAT0, which triggers the master clear, places the processor in a PI cycle by setting PI CYC, and then triggers IT1, which calls the memory subroutine and supplies a memory address for instruction retrieval.

In a normal sequence, i.e., PI CYC(0), the address comes from the program counter; if the processor is in a PI cycle, however, the appropriate PI channel address is transferred into MA. Furthermore, if PI OV is 1 (this can occur only in the second consecutive PI cycle when a block IOT is completed), MA is incremented by 1 at the same time that the channel address is transferred in, so the processor performs the instruction in the second location associated with the channel. When the instruction is available in MB, the memory restart triggers IT1A, which transfers the instruction code and AC address from MB into IR. If the instruction was retrieved from a location above 17, IT1A also clears the readin mode flip-flop in case the processor has been in read in.

The instruction cycle may also be entered late for several situations. If a UUO appears in the program, UUO T2 starts a new instruction execution by setting IF1A and making a read request to retrieve the instruction in location 41; the memory restart then continues the chain automatically by triggering IT1A. After retrieval of the operand in the fetch cycle of the execute instruction, XCT T0 returns to IT1A so that the operand is executed as an instruction. Similarly, the console operation execute triggers IT1A after the contents of the DATA switches have been transferred to MB via AR.

b Address

The calculation of the effective address for an instruction is governed by the logic shown in the upper half of Figure 5-3 (flow chart, Figure 4-4). In a normal sequence, IT1A starts the address cycle by triggering AT0. However, the cycle is also triggered by CHT9 for the second part in a character operation: in this case, the processor has already handled the pointer and must now calculate the effective address for the operand as specified by the pointer. AT0 transfers the address portion of the instruction word from MB to AR, transfers the indirect bit and the index register address to IR, and clears MA in preparation for subsequent memory access.

AT0 also generates PI SYNC (8.2b) which strobes the priority interrupt system provided the processor is not already in a PI cycle. If the strobe discovers any request, PI SYNC generates IAT0 which returns the processor to the instruction cycle. If there is no request or if the processor is already in a PI cycle, the sequence continues the address cycle by triggering AT1 provided the operator has not pressed the INSTRUCTION STOP key within the preceding 100 μsec (IF1A being 0 guarantees that the return is to the address cycle rather than the instruction cycle.) If the instruction specifies no index register (IR14-17 = 0), AT1 jumps directly to AT4. However, if address modification is to be performed, AT1 triggers AT2, which transfers the index register address to MA, makes a memory read request, and sets A LONG. The return from memory triggers AT3, which sets AF3A and requests the add subroutine to add the contents of the index register to the address specified by the instruction. The subroutine return transfers the calculated address from AR back to MB and moves the cycle on to AT4, which clears the left half of AR. If the instruction specifies an indirect address (IR13 is 1), the cycle continues to AT5 which transfers the calculated address from MB back to MA and makes another read request to retrieve the new address. AT5 also sets A LONG, clears the indirect bit and index register portions of IR, and sets AF0. The last action allows the memory restart to trigger AT0, restarting the address cycle.

If the instruction specifies a direct address, the processor goes on from AT4 to the fetch cycle. At the completion of address calculation, MA, MQ, and ARLT are all clear; the effective address calculated for the instruction is contained in the right halves of both MB and AR. The contents of MBLT are equal to the last word retrieved from memory, provided the final address cycle did not index.

c Fetch

Figure 5-4 shows the logic that controls the retrieval from memory of the operands necessary for the execution of an instruction (flow chart, Figure 4-4). The lower part of the figure shows the generation of the control levels for the cycle. In all instruction groups, individual instructions or specific instruction modes which do not require an accumulator, the net in the lower left generates an inhibit for that function. The bottom two inputs, CH INC OP and CH ∿ INC OP, both apply to the first part of character operations and indicate whether or not the pointer shall be incremented after it is retrieved. The level IR 254-7 is asserted for the four included instruction codes: the first two are jumps; the third is XCT; the fourth is not used. All other inputs that generate FAC INH are standard instruction situations (5.3). There are a few instructions which, after retrieving the accumulator, require the retrieval either of a second accumulator or of a word addressed by half of the already retrieved accumulator. The three levels that govern these situations are shown in the center of the figure. The cycle fetches AC2 for any double word shift operation and for fixed point division, which uses a double-length dividend. POP and POPJ both pull out the word addressed by ACRT, which keeps the final address for the pushdown list. Fetching of the word addressed by ACLT is required by JRA and BLT: the former to restore AC as the return from a JSA which stored AC in E and saved E and PC in AC; the latter to retrieve a word for subsequent storage in the location addressed by ACRT.

Two levels govern the retrieval of a word according to the effective address, FC(E) and FC(E) PSE. The first reads and releases the memory so it can rewrite automatically. The second makes a read/write request: the memory subroutine thus pauses after fetching so that information can later be written into the same memory location (usually in the store cycle and controlled by the same level). For the first part of character operations, FC(E) retrieves the pointer if it is not to be incremented; whereas the pause is required if an incremented pointer is to be written back in. Similarly, in the second part, a character deposit requires the pause while a load does not. An IOT BLK also retrieves a pointer which is automatically indexed and thus requires the pause. The remaining inputs to both nets are all standard instruction situations, but the reader should take special note of two inputs to FC(E): IR MD FC(E) and IR FP. The first represents all fixed-point multiplication and division operations that do not use E as an operand; the second includes all floating instructions except floating scale. All of these instructions have modes that store the result in E; however, all require significantly greater execution time than other instructions and they request read to release the memory.

In the upper half of the figure are the delays and SBR flip-flops that provide the fetch time chain. The normal entry is from the address cycle at AT4 when no further deferring is required. If the address calculation requires no indexing or deferring at all, A LONG is 0 and the entry into FT0 is delayed; otherwise the entry is immediate. There is also an entry into FT0 for a block IOT: after the pointer has been indexed and stored, IOT T0A triggers the fetch cycle for the data instruction that follows (8.1). FT0 performs no outside operations but continues the proper fetch sequence. If the instruction requires no accumulator, the sequence goes directly to FT5. To fetch AC, the sequence goes to FT1, which transfers the AC address from IR to MA and requests a memory read. The memory restart triggers FT1A, which is also produced directly by BLT T6. In a block transfer, each word is retrieved by a fetch cycle; but the store and index operations are performed by the special BLT subroutine. The subroutine returns to FT1A bypassing FT1, because the instruction must fetch AC, which contains the initial source and destination addresses, only for the first transfer.

The sequence from FT1A and the operation performed by it depend upon whether additional AC operations are necessary. All sequences include a switch between MB and AR so that AC is in AR and E in MB. If there are no extra AC fetch operations, FT1A clears MA and switches MB and AR. To fetch a word addressed by AC, FT1A clears MA; and if the address is in the left half, swaps the halves in MB. It also triggers FT3, which loads MA with the selected half of AC, makes the MB-AR switch, and goes on to FT4. If a second accumulator is required, FT1A adds 1 to MA, switches MB and AR, and skips to FT4 which triggers the memory read for either type of extra AC fetch. FT4 also sends MB to MQ to save E. The memory subroutine returns to FT4A, which clears MA, and switches MB and MQ so that MQ contains the extra word fetched and MB again contains E. FT4A also triggers FT5, which results directly from FT1A if no additional AC operations are required. FT5 transfers E from MB to MA and then triggers FT6 or FT7 depending upon whether the instruction requires a memory read or read/write. The memory return then completes the cycle by triggering FT6A, which results directly from FT5 if the effective address is not used for memory access.

At the completion of the cycle, MA contains E; and AR and MB contain AC and C(E) if both words were fetched, although the AC halves are swapped if the cycle also fetched the word addressed by the left half. If there was no AC fetch, AC contains E with the left half clear; and if there was no C(E) fetch, MB contains E with the left either clear or in the same state as at the end of the address cycle (the latter case occurs only if no fetch operations were performed at all). MQ is clear unless an additional AC operation was required, in which case it contains the extra word.

d Execute

After fetching the operands, the processor performs the operations necessary for the execution of the instruction. Figure 5-5 shows the logic for the execute time chain, the flow chart is Figure 4-5. Only the simpler instructions are actually executed by the ET pulses; for the more complicated ones, the pulses trigger appropriate subroutines.

The lower portion of the figure shows the generation of three levels that control the execute sequence. Two are inhibit levels that break the chain for subroutines; reentry into the main sequence is made from the subroutine usually—but not always, at ET10 (XCT and UUO, for example, both return directly to the instruction cycle). The subroutines for all IOTs and for FSB start at ET4 and thus inhibit ET5. All other instructions requiring execution by subroutine inhibit ET4 although this does not mean that the subroutine entries are made at ET3. In fact, most of them are at ET0 but the execute chain continues to ET3. There is also a group of instructions (including fixed-point add and subtract and some of the data transmission and executive instructions) whose transfer and logical operations are triggered by the ET pulses but which require an AR subroutine. These instructions all generate the level AR SBR which inhibits ET4 but only for a pause: AR SBR causes ET3 to trigger the SBR at the left at the same time that it triggers an appropriate subroutine. The subroutine return then triggers ET4 to continue the chain. AR SBR includes several instructions which generate both inhibits: floating subtract requires the negate subroutine at ET3 and enters the floating add subroutine at ET3 and enters the floating add subroutine at ET4; both block IOTs index the pointer at ET3 and switch to an IOT subroutine.

Almost all logical and transfer operations triggered by ET pulses occur in the first half of the chain; most instructions using ET6 to ET9 affect the program counter. These instructions generate E LONG (right) which causes ET5 to go to ET6 instead of skipping to ET10.

In the execute cycle flow chart, the only events for which complete conditions are listed are general operations independent of specific instructions and operations involving PC, MA, and the flags. The many logical and transfer operations on AR and MB that actually execute the Boolean, data transmission, and other instructions occur almost exclusively at pulses 0, 1, and 4; to determine which events are required for a specific instruction, refer to the appropriate instruction flow chart. The following description of the execute cycle discusses only the sequencing of the cycle, entry to and return from subroutines, and the more general operations. For events involving individual registers, refer to the appropriate section of the logic in this and following chapters.

To begin the execute cycle, the final pulse from the fetch cycle FT6A, triggers ET0 and ET0A simultaneously and also triggers a delay for a subsequent ET1; ET3 follows automatically from ET1. The two 0 pulses are logically equivalent but two pulse lines are required because of the many events they trigger. Usually ET0 increments the program counter so that the next instruction will be taken from the next memory location in sequence. The only circumstances that inhibit PC+1 are those in which ET0 will occur again before the next instruction in normal sequence is to be performed. ET0 also clears the AR carry flip-flops for use during the cycle, synchronizes those console operations which may interrupt the normal sequence from store cycle to instruction cycle (5.1), and clears RUN for the halt instruction (JRST with a 1 in bit 10). The first three execute pulses also handle hold and dismiss operations for the priority interrupt system (8.2b), restoration of flags, and entry into user mode. ET0 provides subroutine entry for character operations, both fixed and floating multiplication and division, and floating addition; ET3 starts the subroutines for BLT, FSC, UUO, and all shift operations. XCT also stops the chain at ET3 in order to return to the instruction cycle.

ET4 follows immediately from ET3 only if there is no inhibit. For the instructions that generate AR SBR, ET3 triggers the appropriate AR subroutine and sets ET4 AR PSE. The subroutine return, ART3, clears this SBR and triggers ET4, which in turn continues to ET5 unless there is an entry to the FSB subroutine. If ET4 follows from ET3 without pause, ET5 follows immediately except for the in-out transfer instructions: for these ET4 triggers the IOT subroutine, but at its completion time pulse IOT T3 returns to the execute cycle by triggering ET5.

Most computational and data transmission instructions skip the second half of the execute cycle by having ET5 go directly to ET10. However, if E LONG is asserted, the chain continues through all the remaining pulses without interruption. ET7-9 perform the necessary program control operations (5.4); the final two pulses in the cycle regulate the flags (6.2e, 8.3). Since most instruction results that are to be deposited by the store cycle are produced in AR, most instructions switch AR and MB either at ET9 or ET10. ET10 sets up MB and triggers the store cycle not only for most instructions executed in the execute cycle but also for many of those executed by subroutine. The return for fixed-point division is at ET9 (provided the division could be performed); return is made at ET10 for character operations, all floating-point instructions (all of these except floating scale are made from the normalize return subroutine), fixed-point multiplication (which also returns via NRT6), and the block transfer routine if the block is complete.

e Store

Information resulting from the execution of an instruction is deposited in memory by the store cycle (logic, Figure 5-6; flow chart, Figure 4-5). The sequencing of the store time chain is controlled by four levels: FC(E)PSE from the fetch cycle, which gates in the write restart for any instruction that used a fetch and pause before execution; and three levels that gate in write requests and are generated by the store logic. SC(E) is generated by three types of instructions: those which merely clear location E and deposit information in it without having required a fetch; those which store C(E) in a location other than E; and those which require so much execution time that they requested only the read rather than the read and pause in the fetch cycle to free the memory during execution. SAC INH is generated by instructions that address no accumulator, by computational and data transmission instructions in the memory mode (that mode which stores the result only in memory), or by test instructions that have no result to store. Instructions that compare a memory word against zero store the word (which may be indexed) in both memory and an accumulator, but AC storage is inhibited if AC0 is addressed (the net in C6 decodes IR9-12 for no AC selection). The input BLT LAST inhibits AC storage in the cycle following completion of a block transfer. If a priority interrupt stops the block before completion, the current addresses are stored in AC so that the block may be continued after the break. Some arithmetic and shift instructions generate SAC2 to store MQ in a second accumulator. This event occurs for all double-length shifts, floating-point instructions that store the low-order halt, and fixed-point operations that have a double-length result.

To generate a write request for SC(E), ET10 triggers ST1; whereas, it triggers ST2 to provide a read/write restart for SC(E) PSE. Either time pulse sets SF3, and at the completion of the memory subroutine the return triggers ST3. If there is no storage according to E (or whatever address has replaced E during the execute cycle), ET10 triggers ST3 directly. If there is also to be storage in an accumulator, ST3 clears MA and ST5 loads the AC address into it from IR9-12, transfers AR to MB, and makes the write request. The memory restart triggers ST5A which goes on to ST6 if a second AC storage is necessary. In this case, ST6 increments MA to address the second AC, transfers MQ to MB, and calls another write. ST6 is also delayed slightly to generate ST6A which sets SF7, the SBR for the memory subroutine call. This delay is necessary to guarantee that the return from the previous subroutine cannot trigger both

ST5A and ST7. The return from this storage then triggers ST7 which also follows directly from ST3 if there is no AC storage at all, or from ST5A if there is no second AC storage. ST7 is the return time for the divide subroutine if the division could not be performed, and for a character operation that terminates after the first part (i.e., an instruction which operates not on a character but only on the pointer). Since all FP/CH codes inhibit ET4, those that are not used for instructions cause ET3 to trigger ST7; thus all unused codes (except the UUOs) are interpreted as no-ops. Furthermore, if at any time a user program attempts to address a protected area of memory, a priority interrupt is requested on the CPA channel and the sequence jumps directly to ST7. If RUN is 0, the processor stops at this point. Otherwise, ST7 returns to a new instruction cycle unless the operator has requested examine or deposit, in which case, it returns to the key cycle.

5.3 INSTRUCTION CONTROL

For each instruction to be performed, the 18-bit instruction register receives the instruction code, indirect bit, and AC and index register addresses from MB (Figure 5-7). Although information may be transferred into IR only from MB, the IOT time pulse T0A directly sets IR12, which changes a block IOT instruction into the corresponding data instruction after the pointer has been indexed. For effective address calculation, the indirect bit IR13 controls the repetition of the address cycle, and IR14-17 provides the address of an index register to be used in the calculation. Although there are 16 accumulators, only 15 of them can be used as index registers. A 0 address in IR14-17 indicates no index register selection and the address cycle performs no indexing. The generation of the appropriate address cycle gate, IR14-17=0, is shown in the lower right of Figure 5-8. For the fetch and store cycles, the AC address is supplied by IR9-12. For IOT instructions, IR0-2 are all ones, and IOT control decodes the instruction specified by IR10-12. A device is selected according to the code supplied by IR3-9; the bus drivers for the in-out selection lines are at the top of Figure 5-7.

The lower portion of the figure shows the pulse signals that control IR. MR CLR clears the entire register at the beginning of every main sequence. In every instruction cycle, IT1A transfers the instruction code and AC address (or instruction and device codes for an IOT) into IR0-12 from MB; at the beginning of every address cycle, AT0 transfers an indirect bit and index register address into IR13-17 from MB. If the calculated address is indirect, AT5 clears

IR14-17 in preparation for reloading at AT0 when the cycle repeats. When the first part of a character operation is completed, the processor returns to the address cycle to calculate the effective address for the character from the pointer. Thus CHT8A clears IR13-17 in preparation for loading at AT0, which is triggered by the final pulse in the first part of the character time chain.

The next three logic drawings, Figures 5-8 to 5-10 show the main decoding of IR0-8 to control the execution of individual instructions. In this section of the processor, the decoding is carried down in many cases to individual instructions and instruction modes. There are, however, a few command lines which represent instruction pairs whose differences are minor; trivial additional decoding for these is shown with the hardware that actually executes the instruction. For example, the IR multiply and divide outputs represent both integral and fractional operations; the floating command outputs represent both the standard instructions and those that round. Figure 5-8 shows the primary decoding into major classes, decoding into command levels for fixed-point multiply and divide and for all floating-point instructions, and final decoding for all single instructions without modes that correspond to unique 9-bit codes. All of the Figure 5-8 outputs, although representing functions at several different levels in the decoding hierarchy, have the prefix IR. Outputs from the groups decoded in Figures 5-9 and 5-10 have the appropriate group prefix. Figure 5-10 also shows the timing for SCT and UUO.

The binary-to-octal decoder in the upper left of Figure 5-8 performs the first stage in instruction decoding by determining which of the eight primary instruction classes is specified by IR0-2. The decoder gating input P is grounded so that the decoder is always on. All codes in the UUO class produce exactly the same operations and require no further decoding. In the IOT class, the eight instructions specified by IR10-12 are decoded in IOT control. Outputs 1 and 2, IR FP/CH and IR 2XX, gate other decoders shown in this figure; further control in the classes represented by outputs 3, 4, 5, and 6 is shown in Figures 5-9 and 5-10. The primary command level IR FP/CH includes the instructions for floating-point arithmetic and character operations, and instruction codes 100 to 131 which are not used. IR FP/CH ANDed with the condition 011 in IR3-5 (i.e., codes of the form 13X) gates the lower left decoder, which produces the command levels for FSC and the five character operations. At the right, the primary level gated by IR3(1) represents all floating-point instructions other than FSC: bits 4 and 5 are further decoded to determine the instruction, bits 7 and 8 to determine the mode. Bit 6 does not appear here but controls rounding at the normalize return subroutine.

The top center decoder is gated by IR 2XX to convert IR3-5 into eight command levels, each representing eight instructions or two instructions with four modes. The 0 and 1 outputs each represent two of the move instructions and are ORed to generate the command level for the group. At the left, IR SH is further decoded into the six shift instructions plus two unused codes (the two types of arithmetic shift are ORed in B4 for use by the overflow logic); at the upper right IR 25X is decoded into seven miscellaneous instructions plus one unused code. Just below, a gate generates the signal IR 254-7 to inhibit AC fetch and storage, which is not required for these codes.

Although the decoder gated by IR 2XX generates two outputs for fixed-point multiply and divide, the condition representing both of these outputs also generates a combined command level IR MD (Figure 5-8, center). Further decoding produces the appropriate fetch and store gates. All modes but the immediate (01 in bits 7 and 8) fetch C(E). Only fractional division uses a double-length dividend and fetches AC2. The two modes with a 1 in bit 7 store the result in memory; whereas only the memory mode inhibits storage in AC. For all modes that do store in AC, a second AC storage is required for the remainder in division and for the low-order half of the double-length product in fractional multiplication.

Note that three of the decoder outputs at the top of the figure have a suffix "A:" these are IR UUO A and IR IOT A in the upper left, IR JRST A in the upper right. These decoder outputs do not drive the command lines for the corresponding instructions. Instead, they are applied to the executive logic (5.5) to determine in the latter two cases whether the instruction is allowed or must be executed as a UUO. The command level for UUO is generated by the executive logic; the command lines for the other two instructions are driven by the gates in C4, each of which is enabled by the appropriate decoder output when the instruction is not being executed as a UUO. The gate just below ORs the 1 states of IR9 and IR10 for use by the executive logic in testing a JRST.

ACCP V MEMAC, ACBM (Figure 5-9)

For the ACBM group (upper right), bits 3 and 4 are decoded for the action on the masked bits: do nothing clear, complement, set. IR5(1) selects C(E) for the mask direct, as against immediate which uses E. A 1 in bit 8 swaps the AC halves before masking. The net in the upper left decodes bits 3 and 4 for instructions in the class that includes both ACCP and MEMAC: one fourth of the instructions do an arithmetic comparison of AC against either E or C(E); the other three types compare C(E) or AC against 0 and either merely test, or add or subtract 1 before testing. Any instruction in the latter three types generates the main control level MEMAC; the state of bit 5 determines whether the test word shall come from memory or an accumulator. In ACCP, a 1 in bit 5 specifies a comparison with memory (direct); otherwise the test is made against E (immediate).

The logic nets at top center in the figure test the skip or jump condition for all three groups, ACCP, MEMAC, and ACBM. The condition is determined by the pair of gates at the left: bit 7 selects the condition that AR is 0; bit 8 that the test word is less than the standard, be it 0, C(E), or E. The upper gate functions only for ACCP and MEMAC: the logical comparison employs an AND function and the only test is whether or not the masked bits are all zeros. In an arithmetic comparison, the function AR=0 represents equality of the test word and the standard. The function representing the inequality is the exclusive OR of the AR sign bit and the overflow condition. This function is true when the subtrahend is greater than the minuend; but since the standard is subtracted from the test word, it is true when the test word is smaller. In MEMAC, the overflow condition is automatically false (for a complete description, see the flag logic, 6.2e) so the entire function is true when AR0 is 1, i.e., when the test word is less than 0.

FWT (Figure 5-9)

For a full-word transfer, the four standard modes are decoded from bits 7 and 8; a left-right swap is made for MOVS (bits 5, 6 = 01); and the word is negated either when moving the negative or when moving the magnitude while the word is negative, both situations being represented by the condition IR6(0) V AR0(1) in MOVN or MOVM.

HWT (Figure 5-9)

For a half-word transfer, the standard modes are decoded from bits 7 and 8; a 0 or 1 in bit 3 specifies whether the half word shall be transferred into the left or right half of the destination; and a 1 in bit 6 specifies that the source word shall be swapped before the transfer is made so that the left half of the source is transferred into the right half of the destination or the right into left. The other levels control the operation on the other half of the destination. If bits 4 and 5 are both 0, there is no action. If either bit is 1, AR is cleared; the word is constructed by loading a half word into one half and complementing the other half if it should be all ones. If bit 4 is 0, the other half is left clear; however, on IR4(1) the other half is set to all ones (i.e., complemented) if the instruction requires that it be set or that its bits be made equal to the sign of the transferred half and that half is negative. The gates to the left generate the appropriate functions. For a transfer to the right, the upper net generates HWT LT SET if instruction bit 5 is 0 or the sign bit MB18 of the right half word is 1. The lower gate performs an equivalent function for the transfer left.

BOOLE, AS (Figure 5-10)

Since BOOLE and AS have the same modes, the OR function of the two instruction levels gates a decoder for bits 7 and 8. For further decoding of AS, the command level is merely gated by the states of bit 6 to determine whether addition or subtraction is required. To decode for the 16 Boolean operations, bits 4, 5, and 6 are applied to binary-to-octal decoders, one gated by the 0 state of bit 3, the other by the 1 state. The table at the left lists the 16 operations by name, function, and number, shows the result for each of the four possible pairs of operand bits, and lists the basic functions which, taken together, produce the required result for a specific instruction. Each operation is performed by three of the execute time pulses and may require from one to three of the basic functions.

JP (Figure 5-10)

Bits 6-8 of the jump and pushdown group are decoded into eight individual instructions. For controlling transfers between MB and AR at the end of the execute cycle, there are two subsidiary levels—one that represents all JP instructions except JSP, and another that represents the pushdown and pullout instructions. For PC control, JP JMP represents the six instructions that jump, i.e., all except PUSH and POP. Three of the instructions that save PC (PUSHJ, JSR, JSP) also generate JP FLAG STOR to save the miscellaneous bits (JSA also saves PC but it stores E instead of the miscellaneous bits).

XCT, UUO (Figure 5-10)

IR XCT causes ET3 to trigger XCT T0, which returns the processor to the instruction cycle at a point beyond the memory subroutine so that the processor then executes the operand in MB as an instruction, just as though it had been retrieved by the instruction cycle. For all codes in the UUO class, the execute clears MA and MBLT, then transfers address 40 into MA and the instruction code into MBLT. The last event is equivalent to UUO T0, which triggers a memory write to deposit the trapped instruction, with its address portion replaced by the calculated effective address in location 40. The memory return triggers UUO T1 to index MA, and UUO T2 makes a read request and sets IF1A in the instruction cycle logic. The memory return automatically triggers the remainder of the instruction cycle so the processor performs the instruction in location 41.

5.4 PROGRAM CONTROL

Each instruction in the program is retrieved from the memory location addressed by the 18-bit program counter (Figure 5-11). At the beginning of every execute cycle, the counter is stepped one position so that instructions are taken from consecutive memory locations. The program controls its own sequence by means of skip and jump instructions. Skip instructions cause the processor to skip one instruction in the normal sequence if a specified condition is satisfied; the skip is implemented by advancing PC one extra position at the end of an execute cycle. Jump instructions transfer program control to any chosen location, sometimes upon satisfaction of a condition, by loading a new address into PC. An address can be transferred in only from MA so any input from the console ADDRESS switches or a jump address from MB must be made via MA and the transfer must be preceded by a clear. The flip-flops are connected in a carry configuration so a pulse at the PC+1 input to PC35 adds 1 to the contents of the counter.

Figure 5-12 shows the control logic for the counter: three control pulses are produced in the upper half; the lower half shows the generation of the gates that control counting, skipping, and jumping. Every program begins with the console operation start or read in, in which KT1 clears PC and KT3 loads the ADDRESS switches into it. For normal counting, PC is incremented at the beginning of every execute cycle. The circumstances which inhibit program counting at ET0 are those in which another execute cycle will occur before the next instruction in normal sequence is to be performed. Character operations generally require two main parts, the second beginning with the address cycle, so PC is not incremented during operations on the pointer except in the single instruction that affects only the pointer and has no second part (C3). XCT, UUO, and a block IOT BLT all involve execution of instruction pairs and counting occurs in the second execute cycle. Since a PI cycle interrupts the normal sequence between instructions, the count must be inhibited in it because the processor has not yet executed the currently addressed instruction. Counting is inhibited throughout a block transfer because the BLT subroutine returns to the fetch cycle to process each word. When the block is complete (MQ0 = 0), PC is counted directly from the subroutine at BLT T5A. The inhibit also applies to an instruction executed from the console.

For changes in the program location out of normal sequence, PC SET causes ET7 to clear and ET8 to load, whereas PC+1 (ET9) causes ET9 to count. The net for PC SET has as input all unconditional jump instructions and an enable level which is asserted for any conditional jump when the condition is satisfied. Conditions include JFCL when the addressed flag is 1, the add-1 jumps when AR has the appropriate sign, and those arithmeitc compare instructions that use an accumulator when the test is satisfied. An extra count occurs at ET9 on two unconditional subroutine jumps so that the subroutine begins one place beyond the storage location of PC or AC. Any other extra count is for skipping and is represented by PC+1 ENABLE. The skip conditions include the two IOT status test instructions when the appropriate condition appears in AR, and the satisfaction of the test condition for any logical compare instruction or those arithmetic instructions that compare memory against 0 or an accumulator against either E or C(E). There is another conditional skip for a block IOT that does not use the PI system: if the indexing of the pointer did not produce a carry into AR0, IOT T0A counts PC (the computer performs the next instruction in normal sequence only if the block is complete). If any jump or skip occurs (other than an IOT block skip), the OR gate at the right generates the level PC SET $\vee$ PC+1 for setting the PC change flag (6.2e).

5.5 EXECUTIVE LOGIC

The executive logic allows the executive routine to control the sharing of processor and memory by a number of programs. The executive routine selects a user program and the area in core assigned to it by loading the protection and relocation registers, placing the processor in the user mode, and jumping to a location appropriate to the selected program. The user program may be interrupted temporarily by a block IOT in a PI cycle (which is under control of the executive routine and is hence unrestricted); but if a JSR is performed in a PI cycle (such as following the completion of a block IOT or for servicing some other type of interrupt), the processor leaves the user mode with control returning to the executive routine. Other than for priority interruptions, the user program has control until it attempts to use a protected area of memory or to perform an illegal instruction. The former action causes the processor to go the end of the current main sequence and triggers a priority interrupt on the processor channel; the latter causes the processor to perform the illegal instruction as a UUO (all UUOs are unrestricted but automatically return control to the executive routine).

Figure 5-13 shows the executive logic except for the memory protection and relocation registers which are discussed with the memory address logic (7.1c). The pulse amplifiers at top center provide the clear and set pulses for PR and RLR when the DATAO clear and set pulses are gated by CPA, i.e., when a DATAO for the processor appears in the program. The only unprogrammed clear for these registers is the master start because their states must remain until deliberately changed by the program. Similarly, the user flag and the illegal operation flip-flop (A3, B5) are also cleared only by the master start because they must maintain control functions from one main sequence to another. The remaining three flip-flops in Figure 5-13 provide synchronization and are cleared by the master clear. However, since the set function for EX PI SYNC is a level output from the PI cycle flip-flop, it remains set as long as PI CYC is 1 even though there may be a master clear between a pair of PI instructions. After PI CYC is cleared, the sync remains set until the beginning of the next main sequence. EX UUO SYNC is set at the beginning of every address cycle and then cleared at the end of the main sequence. The purpose of this flip-flop is merely to prevent the generation of the UUO command level during the time that IR is clear in the instruction cycle.

To set up a specific user program, the executive routine loads PR and RLR with a processor DATAO. It then jumps to the location for the program with a JRST that also sets the executive mode sync flip-flop (B3). The setting may be done either by programming a 1 in bit 12 of the JRST, or by restoring the flags (a 1 in bit 11) provided that a user program was running at the time the flags were stored. Instructions that store the miscellaneous bits with PC store EX USER in bit 5, but the restoring JRST does not act on the user flip-flop directly. Instead, it sets the executive mode sync if MB5 is 1. The processor does not leave the executive mode until the end of the main sequence in which the sync is set: at this time, the transition of the sync back to 0 sets EX USER.

The addresses in the user program are checked against PR to determine whether they are legal and relocated to the area assigned to the program. The net in the lower right generates a level that inhibits both relocation and protection; the inhibit is always asserted if the user flag is 0. The net also inhibits the relocation of fast memory addresses so that these locations are available to all programs (the protection inhibit is really unnecessary here because these addresses are ipso facto less than the minimum block size). There is also no protection or relocation of addresses that occur in a PI cycle because the instructions executed are under control of the executive routine and must be unrestricted even if a user program is running. Again only the relocation inhibit is necessary for the PI channel addresses, but both inhibits are necessary for the addresses given by the instructions. If the interrupt should require a jump to the executive routine, the JSR that calls the routine also stores and clears the user flag (B2).

The nets in the upper right monitor user instructions by receiving the IR decoder outputs for JRST, IOT, and UUO to generate the UUO command level. The JRST or IOT decoder output drives the corresponding command line only if the instruction is not executed as a UUO. When the system is in user mode, EX IR UUO replaces a JRST that attempts to dismiss a priority interrupt or halt the processor and replaces user IOT (a block IOT in a PI cycle is not part of the user program). Any UUO regardless of mode enables EX IR UUO (after the UUO sync is set); this thus provides a means by which a user program can communicate with and return control to the executive routine. EX IR UUO inhibits relocation since it must use addresses 40 and 41, and at ET1 it sets EX ILL OP. The 1 state of this flip-flop then continues the inhibit into the JSR in location 41. At ET7 the JSR clears the user flag and at ET8 the illegal operation flip-flop. Just in case a block IOT that does not overflow should interrupt the JSR, EX ILL OP is cleared by all block IOTs.

The illegal operation flip-flop is also used to inhibit relocation during an examine or deposit that is inserted between two main sequences while the processor is running. For this purpose EX ILL OP is set by ST7 at the same time that the key cycle is triggered and cleared by the read/write return.

CHAPTER 6

ARITHMETIC LOGIC

The first half of this chapter describes the registers used for arithmetic calculations. Three full-word registers, AR, MQ, and MB, are used for computations on full-word and half-word fixed-point numbers and the fractional parts of floating-point numbers. All data transfers and logical operations on computer words are also performed in these registers. Besides the full-word registers, there are two 9-bit registers SC and FE that are used for floating-exponent calculations and for various subsidiary computations such as calculating the size and position portions of the pointer in character operations and counting the number of steps required for fixed- and floating-arithmetic operations. Included with each register is a discussion of its input gating, its control logic, and any other hardware associated specifically with the register (e.g., the flag logic and AR subroutines with AR, the SC subroutines with SC).

The second half of this chapter describes the time chains that control the execution of data and arithmetic instructions outside of the execute cycle. The test discusses the generating conditions for the pulses in a given chain and also describes the events that occur at each step. In many instances, a number of pulses from the different time chains all must trigger the same operation in an arithmetic register. Many subroutine time pulses are therefore connected to the register control logic through OR gates in a subroutine interface (6.5). Lines from this interface are labeled by the functions they perform and all have the prefix CFAC (computer floating-arithmetic connection). When only one or two subroutine time pulses trigger a given event, they are supplied directly to the register logic from the subroutines. In the following discussion of the arithmetic registers, the significance of the generating conditions for an individual register operation is given only for the execution of those instructions performed in the execute cycle and for other events in the main sequence. To determine the significance of any event triggered by a subroutine time pulse, either directly or through the subroutine interface, the reader should refer to the discussion of the appropriate subroutine.

The three full-word arithmetic registers are contained in the same set of double-height modules, that is, MB_i, AR_i, and MQ_i are all on the same module (it also contains the ith bit of the memory indicator register MI, which is described in the memory logic). Each 36-bit register is controlled as a pair of half registers and is shown in two logic drawings, the first for the left half in mounting panels D and E of bay 2, the second for the right half in panels H and J of the same bay (e.g., see MB, Figures 6-1 and 6-2). The flip-flops of all half registers are in panel locations 5 to 12 and 16 to 24. All control pulses are supplied to them by a pair of pulse amplifiers in locations 14 and 15. On the register drawings, these are shown merely as blocks with the pin connections labeled; the actual pulse amplifiers with input gates are shown in the drawings of the control logic associated with the registers. Each PA output drives 18 input gates and is connected to 18 register modules for ordinary 0 or 1 transfers, but to only 9 for jam transfers. All inputs to every register bit are labeled by signal name; but since all the registers are contained in the same set of modules, connections between corresponding bits are made internally and pin connections are listed only for external signals.

6.1 MEMORY BUFFER

Figures 6-1 and 6-2 show the left and right halves of the full-word memory buffer. All transfers to and from memory are made via MB, butthe register is discussed here because it holds one of the operands in most arithmetic and logical instructions. Each MB flip-flop has a direct clear input (which receives the register clear pulse) and gatable clear and set inputs. The MB modules include six sets of internal gates, but the gatable inputs are also available at the connector so that transfers can be made with single-bit pulses from external gates. Transfers of either zeros or ones may be made into MB from either AR or MQ. The two halves of MB may also be swapped; i.e., the left may be jam transferred into the right and the right into the left. For various executive instructions, the program counter may be stored in the right half of MB; the instruction register in MB LT. Transfers of information into MB from the memory bus (7.2) are made via single-bit pulse set inputs shown at the bottom of the figures. Clear inputs are also available, but these are used only for bits 1-8.

The external gates for the single-bit inputs other than those from the memory bus are shown at the top in Figure 6-4. Besides the clear gates for bits 1-8, there are also set gates for bits 0-8 that parallel the memory bus inputs. The gates are controlled by three transfer pulses, two of which set or clear bits 1-8; whereas the third stores the miscellaneous bits in MB0-5.

The generating logic for these transfer pulses is shown in the upper right of Figure 6-3. The set and clear functions occur in floating-point operations to nullify the exponent part of the register and are conditioned by the MB sign bit (refer to the appropriate subroutine). Storage of the miscellaneous bits occurs at ET6 in several of the subroutine-calling jumps. These bits include the four AR flags which can be used by a subroutine but return to their original states following the interruption. The miscellaneous bits also include CHF7 and EX USER. The latter is saved so that the executive routine can return control to a user program by restoring the flags. Saving CHF7, which is set at the end of the first part in a character operation, allows the computer to return properly to the second part if there is a priority interrupt between the two.

The remainder of Figure 6-3 shows the logic that governs all of the regular MB transfers. The top section shows the pulse amplifiers that drive the register gates; these PAs are triggered through OR nets whose inputs come either from the subroutine interface (6.5) or from the transfer gates in the center section of the figure. Timing inputs to these gates are supplied by the various main sequence and subroutine chains; the gating levels for the execute time pulses are supplied by the OR nets shown in the bottom section.

The clear function (upper left) is not required for transfers to MB from AR or MQ since these are always jam transfers. For transfers in from memory, a signal from memory control clears MB before the single-bit pulse inputs arrive via the bus. For a UUO, ET1 clears only the left half of MB leaving the effective address in the right half, and the ET3 loads IR into MBLT (D2, B7). The transfer pulse also triggers the UUO subroutine (5.3). MB must also be cleared before the transfers from PC that occur in various jumps. These jump instructions generate the level MB PC STO (C3) which clears MB at ET5 (B1) and transfers PC into it at ET6 (A6, B6). All but the restoring jump JRST also save the miscellaneous bits along with PC (A8). There is one jump, JSA, that transfers PC at ET6 but is not included in MB PC STO; this instruction gates ET6 via the diode net in D6. No prior clear is required because the left and right halves of MB are swapped at ET0 (D7) placing E in the left half and leaving the right half clear.

The two diode nets in D7 provide six conditions for the left-right swap of MB. Three of these conditions are standard instruction modes and a fourth, JSA, is described above. For the other two, CONO makes the swap so that E is available over both halves of the I/O bus, and BLT does it to restore the address pair to its original configuration in order to store the data word in the location specified by the right half. These two levels gate the appropriate execute time pulses at B2 to trigger the transfer PAs in A2, 3. The swap is also triggered at FT1A (B2) whenever the processor must fetch a word addressed by ACLT (5.2c). The fetch cycle swap includes BLT, which then requires the second swap to restore the original address pair at ET1.

The logic that controls transfers from AR to MB occupies the entire center portion of the drawing from top to bottom. Almost all the transfers are of zeros and ones simultaneously, but there are several cases in which zeros are transferred alone (B3). In the ACBM group, a 0 transfer is always made at ET1 (the group command level is inverted through the diode net in D5) to AND the data word with the mask; then at ET6 zeros are transferred again if the masked bits are to be cleared. The 0 transfer is also used in a character deposit to produce an actual transfer: the character portion of MB contains all ones and all bits outside of the character in AR are also ones. The other three control pulses trigger the transfer of zeros and ones together: they jam transfer AR to MB, but two of the signals (one from the subroutine interface) are also applied to AR control (6.2c) to trigger the transfer from MB to AR at the same time. The double transfer always occurs in the fetch cycle to move AC to AR and E to MB; FT1A does it (D3) if there is no fetch of an additional word addressed by either half of AC, otherwise FT3 makes the transfer (B3). The MB-AR switch is required at three points in the block transfer subroutine (6.6a); the remaining transfers are at execute times gated by levels generated in Figure 6-3 (ET0, C2; ET4, D8; ET9 and ET10, C6). In a few instances, the transfer is actually necessary in only one direction but additional hardware would be required to eliminate the superfluous one. For an explanation of these cases as well as any other events that are not immediately obvious in the flow charts, refer to 4.5 (also see below).

The single transfer from AR to MB (B5) follows index register modification in the address cycle at AT3A to move the calculated address back to MB; ST5 triggers the transfer in order to store the result of an instruction in an accumulator. Any deposit or instruction execution from the console uses the transfer because the contents of the DATA switches can be sent to MB only via AR; AR also goes to MB at the beginning of the multiply and divide subroutines (6.8b, c). In the execute cycle, the transfer may be made at either ET0 or ET10: the former involves only standard instruction situations (D5), but the latter is complex and requires some comment.

In most instructions, the result appears in AR; and if it is to be stored in an accumulator, the transfer from AR to MB occurs at ST5 (the MB-AR switch at ET9 in ACBM is made specifically to move the result from MB to AR because the transfer in the opposite direction occurs automatically for AC storage). However, if the result is to be stored in E, either by requesting a write or restarting a read/write, the result is transferred from AR to MB at ET10 in preparation for the store cycle unless a transfer inhibit is asserted. The net that generates the transfer gating level is in C7; the net for the inhibit is in D4. The conditions generating the inhibit represent situations in which the result is already in MB or is being moved there by a 2-way transfer. In EXCH, the switch is made at ET0 (C2); in a character deposit, the character is inserted into the data word in MB. The transfer is also inhibited for all instructions in the jump and pushdown group: instead these instructions use the 2-way transfers at ET9 and ET10 (C6). At ET9, the switch is made for all instructions except JSR, whose result is already in MB (flowchart, Figure 4-7 left). In JSP, JSA, and JRA, the ET9 transfer results in the appropriate configuration for the store cycle. In the other four instructions, represented by the logic level JP ∧ IR6(0), the result is already correctly placed before ET9, so a second switch of MB and AR is made at ET10. This double switch is made so that the address from AC right is available to MA at ET10 in PUSH and PUSHJ (the transfers are unnecessary in POP and POPJ, but extra hardware would be required to eliminate them).

The remaining transfer is that of both ones and zeros from MQ (Figure 6-3, upper right) which is triggered by two signals from the subroutine interface, one for a single transfer, the other for a switch of MB and MQ. The conditions within MB control that trigger the transfer (B5) include FT4A to return E to MB following a second accumulator fetch operation, and ST6 prior to storing a second accumulator. The transfer also occurs at the beginning of the block transfer subroutine (6.6a). In the execute cycle, it is used at ET0 in three of the JP instructions (C1) to move to MB the word fetched from a location addressed by half of AC.

6.2 ARITHMETIC REGISTER

Figures 6-5 and 6-6 show the left and right halves of the full-word arithmetic register. Each AR flip-flop has a direct clear input (which receives the register clear pulse), gatable clear and set inputs, and two complement inputs, one of which accepts a positive-going pulse, the other a negative-going pulse. The AR modules include ten sets of internal gates, but the gatable 0 and 1 inputs are also available at the connector so that transfers can be made with single-bit pulses from external gates.

The AR outputs are connected within the 6205 modules to the input gating of the memory buffer and are also available through the module connectors for connection to the shift gates in other AR flip-flops, the in-out bus, and various other places in the processor. The outputs of the sign bit AR0 are used throughout the processor for control purposes, such as in the program control test nets and in many of the arithmetic subroutines, so these outputs are buffered by the drivers shown in the lower left of Figure 6-5. The 0 outputs of all AR bits are also available at the connectors through diodes that are joined externally to form large AND gates as shown in the left of Figure 6-10. Assertion of the output from a single AND gate indicates that the corresponding portion of the register contains all zeros (note that bit 9 is not included among the four gates). The first stage decoder outputs are further ANDed in the two nets in C3: the lower net decodes AR for the condition that every bit is 0, the upper net for the condition that bit 9 is 1 and bits 10 to 35 are all zeros. The latter condition, represented by assertion of the signal AR = FP HALF, is necessary for normalizing in floating-point operations. The floating-point fraction -1/2 (AR bits 0 and 9 both 1, and zeros in bits 10 to 35) is considered normalized even though bit 9 is the same as the sign bit.

In addition to the arithmetic register gating and control, this section describes the addition algorithm, the AR subroutines, and the flag logic.

a AR Gating

The external gates for single-bit pulse inputs are shown at the bottom of the two AR drawings, Figures 6-5 and 6-6. The only inputs presently used are for bits 0-8 and the gates for these are shown in the lower part of Figure 6-4. The upper pair of transfer pulses merely clear or set bits 1-8; the lower pair (the two gates at the lower left are triggered by the same transfer pulse as the row above) provide jam transfers from SC1-8 to AR1-8 and from SC3-8 to AR0-5.

The bottom two sets of gates in the register drawings, Figures 6-5 and 6-6, provide 1 transfers into AR from the I/O bus and the console DATA switches. Both of these transfers are preceded by the clear pulse for the register. The next two rows are jam transfer gates for right and left shifting; gating levels for a given bit are the 0 and 1 outputs of the adjacent bits. Since at the register extremities the connections vary depending upon the type of shift, there are special inputs for the left shift gates at AR35 and for both sets at AR0. The generating nets for these special shift inputs are shown in the upper portion of Figure 6-7. At the left are listed the different types of shifts with the time pulses at which they occur and block diagrams showing the shift configurations. These diagrams are also drawn at the appropriate places in the flow charts. Among the level inputs to the shift nets are several composite functions that represent groups of conditions, all of which require the same shift type. The arithmetic shift of AR and MQ combined (B4) is required by the corresponding shift operation, but it is also required in the normalize return and floating-add subroutines. Another composite function is SHC DIV, which is asserted by any type of division provided NRF2 is 0 (D4). The flip-flop condition does not apply to fixed division but is necessary in control over floating division so that the control level cannot affect shifting in the normalize return subroutine, which follows all floating-point arithmetic subroutines. Just above is another composite function (C4), which is asserted during a division or a combined logical shift.

The three shift input nets use these composite functions as well as individual instruction levels to determine the effect of any given shift on the AR extremities. In left shifting, AR0 receives the state of AR1 unless some type of arithmetic shift is being performed, in which case AR0 is unaffected. The center net controls AR0 whenever AR is shifted right. The upper two gates make MQ35 the source of data for AR0 on a combined rotation and AR35 the source on a single rotation. The gates at the lower right of the net disable the 1 input so that AR0 is automatically cleared in a character load or a logical shift; the gate at the lower left disables both inputs so that AR0 is unaffected by any right shift in multiplication or division, or a right arithmetic shift. The net at the right provides shift left input to AR35 from AR0 in a single register rotation, from MQ1 in any double length arithmetic shift, and from MQ0 in a combined logical shift, combinedrotation, or division shift. The gate at the lower right disables the 1 input so that AR35 is automatically cleared in a single left arithmetic or logical shift, or a character deposit.

Above the shift gates are three rows of gates that use the outputs of the corresponding bits in MB. The bottom two rows supply 1 and 0 transfers that provide the OR and AND functions of MB and AR when used separately, but provide a jam transfer when pulsed together. The third set is connected to the AR complement inputs and is conditioned by ones in MB. Pulsing this set of gates produces in AR the exclusive OR of MB and AR. The next row of gates above the MB gates provides a simple complement function, i.e., pulsing these gates complements all AR bits. The top two sets of gates generate carry pulses, both of which are applied to the flip-flop to the left of the module containing the gate that generates the pulse. For example, the gates on the AR8 module use AR8 and MB8 outputs as level inputs, but the carry output complements AR7 (this method of placing the gates saves pin connections because the level inputs are internal to the module). The two sets of gates provide related carry functions, a ripple carry and a carry initiate. The lower set, the ripple carry, is a serial function, i.e., there is no control pulse applied to all gates simultaneously. The chain starts at AR35 with the pulse AR+-1T1, which is applied both the the AR35 complement input and the lower carry gate. This pulse, which occurs only when adding 1 to the contents of the register, complements AR35; and if AR35 is 1, it also triggers the ripple carry to AR34. This second carry in turn complements AR34; and if that bit is a 1, carries on to AR33. The chain continues through the register in this manner except for a break between AR18 and AR17: the carry out of AR18 automatically enters AR17, but the latter may be pulsed independently in order to index two 18-bit words simultaneously.

The upper set of carry gates provides the full-register carry-initiating function for addition. This arithmetic operation is carried out in two stages, first a partial addition, then a carry function. The partial addition is the exclusive OR function of MB and AR ($AR \leftarrow MB(\veebar)$). After the partial sum (the result of the partial addition) has been formed, the full-register carry-initiating pulse triggers the upper set of carry gates to change the exclusive OR into the arithmetic sum. At the end of the operation, the number in AR represents the sum of the contents of MB and the previous contents of AR.

For any given bit, the partial sum (the exclusive OR function) of two numbers is actually the correct sum if there is no carry into that bit. But if there is a carry for that bit, the partial sum is the opposite of the arithmetic sum. For each bit where both summands are 1, the carry initiate directly complements the next more significant bit. However, since the processor cannot sense the prior state of a flip-flop, it instead senses the corresponding configuration of the partial sum. If after partial addition, AR_i is 0 and MB_i is 1, both bits must originally have been 1 and the carry therefore complements AR_{i-1}. Anytime a bit is complemented, a ripple carry is initiated into the next more significant bit, but this carry is inhibited if the bit was complemented from 0 to 1. At each stage, a carry produced through the upper gate goes not only to the complement input of the next more significant bit but also to its lower gate. Thus, a carry initiated by the partial addition of two ones ripples up the register until it terminates when a 0 bit is complemented. That this algorithm does produce the correct sum of two binary numbers is proved below.

b Addition Algorithm

Let A be the original contents of AR, B the contents of MB, PS the partial sum produced in AR by the partial addition, and S the arithmetic sum of A and B. For convenience, let A and B be positive binary fractions whose sum is less than 1, i.e., there is no overflow. A bit of the partial sum PS_i is equal to a bit of the sum S_i if there is no carry into S_i. If there is a carry, PS_i is the complement of S_i. Since there can be no carry into the least significant bit, $PS_{35} = S_{35}$.

To understand the operation of the two carry functions, divide PS into sections from the right so that the first section starts with PS_{35} and ends at the first bit PS_i that satisfies the conditions $PS_i = 0$, $A_i = B_i = 1$. The second section starts with PS_{i-1} and extends to the next bit that satisfies the same conditions as PS_i. Proceed in this way through the entire partial sum. Since there can be no carry input to the least significant bit of the partial sum, it must be correct. If this bit is 1 or if it is 0 resulting from the partial addition of two zeros, there is no carry out and the next bit of the partial sum is also correct. Proceed with each more significant bit of the partial sum until reaching the bit PS_i, which is 0 resulting from the partial addition of two ones. This bit is also correct; therefore, all bits in the first section are correct.

Because the partial sum in PS_i generates a carry, PS_{i-1} is not correct and a 1 from the first section is carried into it by the carry initiate. If PS_{i-1} is 1 (resulting from the partial addition of 0 and 1), there is a ripple carry into PS_{i-2}. The ripple carry propagates up the register until a 0 bit is encountered. If this 0 is the result of a partial addition of two zeros, no further carry is generated; all further bits are correct up to the next 0 that results from the partial addition of two ones, i.e., up to the end of the section. If the 0 that terminates the ripple carry results from the partial addition of two ones, there must be a carry into the next bit. However, the partial addition of two ones is the condition that ends the section and the carry initiate begins a new ripple carry in the next section. Consequently, the carry complements all incorrect bits of the partial sum. At the completion of the carry operation, the result S is the correct sum of A and B.

The preceding example shows that the addition algorithm works for the special case of two positive numbers. Before proving the algorithm for the remaining cases, including negative operands, some further facts should be understood:

The sign bits are included in the partial addition, i.e., the partial sum of two minus signs (ones) is a plus sign (0).

Both carry functions are applied to the sign bit AR0, which is treated as though it were a next more significant bit of AR.

The sign bit conditions both carry functions: there is a carry out of AR0 if two negative numbers are added, or if there is a carry out of AR1 and the sign of the partial sum is minus. Carries into and out of the sign bit (i.e., carries out of AR1 and AR0) are used to detect overflow.

Assume that the binary point is to the left of the most significant bit, i.e., all positive numbers are 35-bit, fixed-point fractions. The computer representation of the positive number x is therefore +.[x] where the brackets enclose the number contained in AR1-35. The sign of this number is represented by the state of AR0. In 2's complement arithmetic, the negative of a number is produced by changing the sign and subtracting the magnitude from 1. The computer representation of the number -x is therefore -.[1 - x]. With this representation, there is no negative 0; the magnitude 1 - 0 overflows, changing the sign back to plus. Furthermore, the largest negative number is -1, represented by the configuration -.[0].

The four cases of addition of two positive 35-bit fractions are:

$$x + y$$

$$(-x) + (-y)$$

$$x + (-y), \quad y \leq x$$

$$x + (-y), \quad y > x$$

Since the 2's complement format allows a representation for -1, either x or y may be 1 in the second case and y may be 1 in the fourth case. In the first case, which is discussed above, the contents of AR after addition represent the number $+.[x + y]$. If $(x + y) \geq 1$, the carry out of AR1 changes the sign. Consequently, if the addition of two positive numbers results in a negative answer, it is apparent that the sum has exceeded the capacity of the register. The processor detects the overflow by checking the sign bit carries: there is a carry from AR1 but none from AR0. The contents of AR then represent the number:

$$-.[x + y - 1]$$

In case two, the addition of two negative numbers, the partial addition and all carries except that into the sign bit would be:

$$\begin{array}{l} -.[1 - x] \\ \underline{-.[1 - y]\qquad} \\ +.[1 + 1 - x - y] \end{array}$$

If $(x + y) \leq 1$, the AR1 carry changes the sign and the complete result is:

$$-.[1 - x - y]$$

which is the computer representation of $-(x + y)$. If $)x + y) > 1$, there is no carry into the sign bit, and its absence in the presence of a carry from the sign bit indicates overflow. AR then contains:

$$+.[1 - (x + y -1)]$$

In case three, the addition of x and -y, where y is less than or equal to x, the partial addition and all carries except that into the sign bit would be:

$$\begin{array}{l} +.[x] \\ \underline{-.[1 - y]\quad} \\ -.[1 + x - y] \end{array}$$

Since $y \leq x$, it follows that $(1 + x - y) \geq 1$. Hence the AR1 carry changes the sign and the complete result is $+.[x -y]$. Since the signs of the operands are different, the magnitude of the result cannot exceed the larger operand and there can be no overflow. Although there is an AR1 carry, the minus sign resulting from the partial addition allows it to ripple through producing an AR0 carry.

In case four, the addition of x and -y with y greater than x, the partial addition and the carry function are:

$$\begin{array}{l} +.[x] \\ \underline{-.[1 - y]} \\ -.[1 + x - y] \end{array}$$

Because y > x, it follows that $(1 + x - y) < 1$. Hence there are no carries from AR1 or AR0 and no overflow. The above result is the 2's complement representation of the number x - y, i.e., -(y - x).

Addition is also used in fixed-point subtraction with the minuend x in AR, the subtrahend y in MB. The subtraction could be performed by taking the 2's complement negative of x, adding -x to y and taking the negative of the result. It is much simpler however to complement the word in AR, then add and again complement the result in AR. The complement of a word, which is produced by exchanging all ones for zeros and zeros for ones, is equivalent to the arithmetic 1's complement in which the sign is changed and the magnitude is subtracted from all ones, i.e., from $1-2^{-35}$. The complement of x is thus:

$$1 - x - 2^{-35};$$

adding y to $\sim x$ yields:

$$1 - x + y - 2^{-35}$$

which equals:

$$1 - (x - y) - 2^{-35}$$

which is the 1's complement of x - y. Overflow is indicated in the same way as in addition; that is, by an AR1 carry without an AR0 carry or vice versa. Overflow is properly indicated even for operations involving -1 because the 1's complement is 2^{-35} more negative than the 2's complement (i.e., when subtracting -1 from 0, the proper carries occur in the addition of -1 to the complement of 0).

c AR Control

Figure 6-8 shows the generating logic for the AR control pulses. The pulse amplifiers that drive the register gates are in the top section; these PAs are triggered through OR nets most of whose inputs come either from the subroutine interface (6.5) or from the transfer gates in the left center section. Timing inputs to these gates are supplied by the various main sequence and subroutine chains, and the gating levels for the execute time pulses are supplied by the OR nets at the lower right. Six of these diode nets are used almost exclusively to gate the AR events necessary for the execution of the Boolean instructions. The flowchart (Figure 4-8) lists at each time the instructions that require each event, and also lists the specific sequence of events necessary for each of the 16 Boolean functions. Because there are no complement gates for MB, all functions that require the AND with the complement of memory use the OR and complement the result; the same is true for the converse. BOOLE 14, which places the complement of memory in an accumulator, clears AR first and then uses the exclusive OR as a 1 transfer. All other functions are quite straightforward. The other diode nets in Figure 6-8 gate the events necessary for the data transmission, IOT, and compare instructions; the functions that are not immediately obvious from the flowcharts are described in 4.5.

The pulse amplifiers that clear AR are shown in the upper left corner of Figure 6-8. To leave the desired address in the right half, only the left half is cleared in the address cycle and the block transfer subroutine. For console operations, both halves are cleared by a signal from the key logic (5.1) and the subsequent transfer from the DATA switches is triggered through the PAs in B7. The other AR clear input is generated by the nets in B3 and C3. Subroutines that require the clear include divide, floating add, and multiply. The clear may also occur at the first two pulses in the execute cycle: it is required at ET0 for four of the Boolean functions (D2), at ET1 by any input IOT (D6) and in any half-word transfer that affects the other half of the destination location. For input instructions, IOT T3 triggers the subsequent transfer into AR from the bus (B8). For an HWT, there is no further action if the instruction is to clear the other half, or the transfer half is positive and the instruction is to extend it; however, to set, or to extend a negative half, the other half is set by complementing through the appropriate gate in B1.

All other complements affect the entire word. One input to the PA gates (A2) comes from the subroutine interface; another is the first pulse in the add-subtract time chain—this pulse occurs only if the subroutine is to subtract (d below). Corresponding time pulses for the index and negate subroutines trigger the complement through the top gate in C2. The gates below that provide the complement at ET0 or ET4 for Boolean functions, and at ET5 and ET7 for the ACBM group in case the masked bits must be cleared at ET6. The bottom gate complements at the end of the carry function in an AR subroutine if the complement control flip-flop has been set.

The four PAs in A3 and A4 are triggered in several combinations to provide 0, 1, and jam transfers from MB into either or both halves of AR. All four must be triggered for a full-word jam transfer. The top two inputs provide for a simultaneous interchange of MB and AR; for any such transfers not made through the subroutine interface, the generating logic is shown with MB control (6.1). Half-word jam transfers include transfer of the address portion of the instruction word into ARRT at the beginning of the address cycle, and transfer of the selected half word in an HWT triggered by ET4 through the gates in B2 with gating levels supplied from the nets in C7. The left transfer also occurs in a CONO so that E is available on both halves of the I/O bus. All other pulses that trigger the PAs in A3 and A4 are for full-word, one-directional transfers. Transfers of zeros alone (B4) are required in both the deposit and load character sequences (6.6<u>b</u>), at ET1 in four of the Boolean functions (D4) and at ET6 for the two IOT status test instructions (D8). The 1 transfer (C5) is used to set the masked bits in an ACBM (D4) as well as to provide a 1 transfer in SETM (BOOLE 3) and an OR function in four of the other Boolean instructions. The gates for the jam transfer are located between B3 and B4 in the figure. This transfer is required in fixed-point multiplication (6.7<u>a</u>), at ET0 in various data transmission, compare and IOT instructions (D5), and in character operations in the first part and in the load sequence of the second part (6.6<u>b</u>).

The PAs for the exclusive OR function of MB and AR (A5) are triggered when ET1 is gated at B4 by the level from D5. This function complements the masked bits in an ACBM, provides a 1 transfer in SETCM (BOOLE 14) and an exclusive OR in XOR and EQV (BOOLE 6 and 11). The exclusive OR pulse amplifiers are also triggered by AR AS T1 in the add-subtract subroutine to provide the partial addition. The next pulse in the chain initiates the carry function through the pulse amplifier in B6. The remaining PAs in the upper right of Figure 6-8 provide for the left and right shifts, which are triggered from the subroutine interface.

The control pulses for the special AR gates that are external to the arithmetic register modules are shown at the right in Figure 6-9. To understand the significance of these transfers, the reader should refer to the description of the subroutines in which they occur. In the exponent calculate and floating add subroutines, AR bits 1-8 are cleared or set according to whether the sign bit is 0 or 1 in order to nullify the exponent part of the register. The jam transfer from SC1-8 into AR occurs in the normalize return and floating-scale subroutines to insert the exponent in the result; a similar transfer of SC3-8 to AR0-5 occurs in the first part of a character operation to insert the new position if the pointer has been incremented.

d AR Subroutines

Figure 6-9 shows the logic governing the negate, index and fixed-point add and subtract subroutines; the flow chart for these is in the upper right of Figure 4-8. In the lower right of Figure 6-9 are the nets that provide the level gates to trigger and control the subroutine time chains. The add gate is asserted only for fixed-point addition, the subtract gate for both fixed-point subtraction and the instructions that arithmetically compare an accumulator against memory. The add 1 and subtract 1 levels are generated by instructions requiring upward or downward indexing of either the entire AR or its two halves independently. AR+1 is asserted in any MEMAC instruction that adds 1 to the test word, in the two add-1-to-both jump instructions, and to index the pointer in any pushdown instruction or block IOT. Similarly, AR-1 is asserted for the MEMAC instructions that decrement the test word and to decrement the pointer in the pullout instructions. Between the add 1 and subtract 1 nets is a net that generates the level AR+-1 LTRT, which causes any entry into the carry chain at AR35 to enter the chain at AR17 as well. The double indexing occurs in the two add-1-to-both jumps and for operations on the pointer in a block IOT or in any pushdown or pullout instruction (these last four are included in the level JPA IR6(0)). The net in the lower right corner generates the level that causes the execute cycle to pause after ET3 for an AR subroutine. AR SBR is generated by any of the four subroutine calling levels described above, and also for negation in a full-word transfer and for a floating subtraction (this last case is for negation of the subtrahend before entering the floating-add subroutine).

The AR subroutines are actually two separate time chains, each with multiple entries, that join for the return to the interrupted sequence. The chain in the upper left handles indexing and negation; downward indexing and negation both require a preliminary pulse to complement AR. Subtracting 1 is done by complementing, adding 1, and complementing again; negation by complementing, then adding 1. All subroutines in the execute cycle are triggered at ET3. AR-1 triggers AR+-1 T0, which complements AR and also sets the complement control flip-flop (C5). The complement pulse for negation may be triggered by a signal from the subroutine interface (6.5) as well as by ET3 in a floating subtraction or the appropriate FWT. Following either T0 pulse, there is a delay to allow the register complement to function before triggering AR+-1 T1 which carries into AR35 adding 1 to the contents of the register. This latter pulse is also triggered directly by time pulses from block transfer, normalize return, and character operations as well as by the input for execute cycle incrementing. For indexing both halves of AR together (this occurs in a block transfer as well as in those instructions that generate AR+-1 LT RT), the same pulse that carries into AR35 is also gated to carry into AR17. Of course, a carry into AR17 occurs automatically whenever there is a carry out of AR18.

Just to the right of the index and negate chain is a separate time chain for addition and subtraction. The first pulse in the chain, which complements AR, is necessary only for subtraction and is triggered not only in the execute cycle, but also from the subroutine interface, and in a block transfer to determine whether the block is complete. AR AS T0 sets the complement control flip-flop, and after a delay sufficient for the complement to function, triggers the partial add pulse AR AS T1. The chain begins at this pulse for any execute cycle or subroutine addition and also when adding an index register in the address cycle. After the partial addition, the chain continues to the next pulse which initiates the full register carry.

There are three pulses that initiate carries in AR. These are AR+-1 T1 in the right half, AR17 CRY IN in the left half and AR AS T2 which triggers carries at any required point throughout the register. All three pulses are applied to the OR gate in B1 cutting off the transistor just to the right by grounding pin 2F5Y. ORed with these carry initiating pulses is a vast OR gate (lower left) that receives the carry outputs from all AR bits. Once any carry function is initiated, all further carries are from the ripple carry, i.e., a carry for any bit can come only from a contiguous bit. Successive carries overlap because of the carry speed and the OR gate is enabled until all carries die out: 2F5Y then goes negative triggering the carry completion pulse. If the complement of AR is not required (i.e., the subroutine is not performing a subtraction) the completion pulse triggers the subroutine return ART3. However if the complement control flip-flop has been set, the carry completion complements AR, and after an appropriate delay triggers the return. ART3 clears the complement control flip-flop and triggers the next pulse in the time chain that has called the AR subroutine.

e AR Flags

The jump-addressable flags and associated logic are shown in the right half of Figure 6-10. Although some AR flag control functions also affect the user flag and the flip-flop that differentiates the two parts of a character operation, only the four AR flags can be sensed or cleared directly by the program. The PC change, overflow, carry 0,and carry 1 flags are cleared initially by the master start rather than the master clear because their states must remain from one main sequence to the next. Program control over the flags is exercised by three instructions, JRST with a 1 in IR11, JFCL, and the CONO for the processor. The last instruction, which governs the many flags in the processor I/O interface (8.3), clears the PC change flag if a 1 is programmed in bit 29 or the overflow flag if bit 32 is 1. JFCL selects flags with bits 9-12 of the instruction word and clears the selected flags after sensing them to make the jump. The JRST, which must be addressed indirectly, clears all the flags and restores them according to the first four bits of the word taken from memory as the direct address. The flag clear and set pulses generated here also handle the restoration of CHF7 and EX USER.

The remaining gates include conditions for setting the flags so that the program may determine whether or not certain events have occurred. Any jump or skip sets the PC change flag provided the instruction in process is not a JRST that is restoring the flags. All other gates are for arithmetic conditions indicating overflow. Overflow in most fixed-point cases is determined from the pair of flip-flops in the lower right, which are cleared at the beginning of every execute cycle and then set by carries out of AR0 and AR1. Since overflow is indicated by the presence of one of these carries in conjunction with the absence of the other (as is shown in the discussion of the addition algorithm in b above), the level AR OV SET is asserted when the two flip-flops have oppositestates; i.e., the overflow level is the exclusive OR of the two carry flip-flops. In fixed-point addition or subtraction or in the MEMAC instruction group—the latter condition being relevant only for those MEMAC instructions that index the test word—ET10 transfers the states of the carry flip-flops to the corresponding flags, and if overflow has occurred, sets the overflow flag. The remaining gates in C6 set the overflow flag on an overflow signal from the subroutine interface (6.5); on the attempt by an FWT to form the negative of -1; and on the loss of a significant bit in an arithmetic shift to the left, i.e., if a 1 is lost in a positive number or a 0 in a negative number (the exclusive OR input for this gate is generated in E5).

The top nets at the far right in Figure 6-10 provide control levels for use in various arithmetic subroutines: the exclusive OR of AR0 and MB0, and the complementary function of AR0 and SC0. The remaining nets at the lower right test for the inequality conditions in arithmetic compare instructions (5.3). The lower net ANDs the overflow condition with $\sim$ MEMAC, and this output is exclusive ORed with AR0 in the upper net. In a MEMAC, the output of the lower gate is automatically false and the output of the upper gate is true or false as AR0 is 1 or 0. In this case, since the comparison is with 0, the function specifies whether the test word is negative or positive, i.e., less or not less than 0. In the ACCP group, a test word from AC is compared against a standard, either E or C(E). The comparison is made by subtracting the standard S from the test word T. The four possible configurations of T and S result in the following signs and overflow conditions:

$T \geq 0,\ T < S$: - and $\sim$ OV
$T \geq 0,\ T \geq S$: + if $\sim$ OV, - if OV
$T < 0,\ T < S$: - if $\sim$ OV, + if OV
$T < 0,\ T \geq S$: + and $\sim$ OV

Hence, the function AR0 $\veebar$ AR OV specifies the relation between the test word and the standard: it is true for $T < S$, false for $T \geq S$.

6.3 MULTIPLIER-QUOTIENT REGISTER

Figures 6-11 and 6-12 show the left and right halves of the full-word, multiplier-quotient register. Each MQ flip-flop has a direct clear input, whichreceives the clear pulse for the register, and gatable clear and set inputs. The MQ modules include four sets of internal gates, but the gatable inputs are also available at the connector so that transfers can be made by single-bit pulses from external gates (at present, these are used only for MQ0).

The upper two sets of gates provide transfers of zeros or ones from MB. The lower two rows are jam transfer gates for right and left shifting; gating levels for a given bit are the 0 and 1 outputs of the adjacent bits. Since the connections at the ends of the register vary depending upon the type of shift, there are special inputs for the left shift gates at MQ35, the right shift gates at MQ1 and both sets at MQ0. The generating nets for the special shift inputs are shown in the lower portion of Figure 6-7. At the left are the different types of shift with the time pulses at which they occur and block diagrams showing the shift configurations. These diagrams also appear at the appropriate places in the flow charts. Among the level inputs to the shift nets are two composite functions that represent groups of conditions, all of which require the same shift type. The arithmetic shift of AR and MQ combined (B4) is required by the corresponding shift operation, but it is also required in the normalize return and floating-add subroutines. Another composite function is SHC DIV which is asserted by any type of division provided NRF2 is 0 (D4). The flip-flop condition does not apply to fixed division but is necessary in control over floating division so that the control level cannot affect shifting in the normalize return subroutine, which follows all floating-point arithmetic subroutines.

The four shift input nets use these composite functions as well as individual instruction levels to determine the effect of any given shift on the MQ extremities. In left shifting (C4), MQ0 receives the state of MQ1 unless some type of double-length arithmetic shift is being performed in which case AR0 is the source. The next net controls MQ0 whenever MQ is shifted right. AR0 is shifted into MQ0 by any double-length arithmetic shift or by the final multiplication shift, i.e., when the shift counter contains 777. In all other circumstances, AR35 provides the input. The next gate to the right causes information to be transferred directly from AR35 to MQ1 in any double-length right arithmetic shift (MQ0 is skipped in the shifting), otherwise MQ0 shifts into MQ1. The final net connects AR0 to MQ35 for any left combined rotation but causes MQ35 to receive the complement of AR0 in a division shift. The two gates below provide constant inputs to MQ35. The left gate grounds the 0 input so that ones are always shifted in the first part of character operation; similarly zeros are shifted in a combined logical shift, in any double-length arithmetic shift, and in the character deposit sequence. There is a single additional gate that affects MQ0 when MQ is not itself shifted. In the first floating division shift at DST10A, only AR is shifted but AR35 must enter MQ0. The pair of gates at the left in Figure 6-13 allow entry through the single-bit pulse inputs, leaving the remainder of the register unaffected.

For use in multiplication, the register actually has an additional bit MQ36 (Figure 6-13, right). This bit receives the state of MQ35 on any right shift but is of significance only in multiplication (6.8b). For the multiply subroutine, the net at the right supplies a level indicating when bits 35 and 36 are equal. Another net in C4 supplies the exclusive OR function of MQ35 and MB0 for use by the divide subroutine (6.8c).

At the top of the figure are the pulse amplifiers that control the register. MQ is cleared only at the master clear and the shift PAs are triggered only from the subroutine interface (6.5). The remaining PAs provide transfers of zeros and ones from MB. Transfer of ones occurs in the deposit character sequence (6.6b). The jam transfer occurs on a signal from the subroutine interface that switches MB and MQ, and at the four time pulses listed at the gate in B4. The transfer at FT4 saves E in MQ if there is to be a second AC fetch operation; FT4A transfers this additional word into MQ at the same time that it returns E to MB. The remaining transfers occur in the multiply and divide subroutines (6.8b, c).

6.4 ARITHMETIC SHIFT COUNTING

In addition to the three full-word arithmetic registers, the arithmetic logic also includes two 9-bit registers for use in auxiliary computations and counting steps in arithmetic operations. The shift counter and the floating-exponent register are shown in Figure 6-14. Each module in the figure includes one flip-flop from each register and all associated gating. FE is used only for storage of intermediate results. In floating multiplication and division, the exponent is calculated in SC and stored in FE while SC counts the number of shift steps in the operation on the fractions. FE also provides temporary storage for the position portion of the pointer in a character operation.

Since actual computations are performed in SC, there is considerable gating associated with its flip-flops, including a carry chain; but FE is used only for storage so it includes only a direct clear and two sets of transfer gates connected to the flip-flop collectors. These provide 1 transfers from SC and from MB0-5 to FE3-8.

a SC Gating

Below the SC flip-flops in Figure 6-14 are the gates that implement the transfers, partial addition, and carry logic. In addition to gatable clear, complement, andset inputs, each flip-flop also has a carry input and a carry output. These are connected from one flip-flop to the next so that a pulse at the SC+1 input to SC8 adds 1 to the contents of the counter. The clear input is used only for the register clear. There are three sets of complement gates: the top one provides a simple complement function and the other two provide the partial add and carry functions which are described below. The remaining gates are connected to the 1 inputs.

The top gate provides an ordinary 1 transfer from FE to SC. The middle gate provides a complement transfer from MB18, 28–35 into SC. The last gate is a diode net (not part of the 6203 module) which receives no level inputs, but instead control pulses are applied to individual gates in order to place a specific number in the counter. In multiplication and division, for example, SC receives the complement of the number of steps to be counted; it is then incremented until it contains all ones, terminating the operation.

At the bottom of the figure are two networks, the lower a carry chain, the upper a set of level gates that supply the data inputs to the partial add (exclusive OR) gates at the complement inputs to the SC flip-flops. The partial add gates are used not only for partial addition but also for transfers provided SC is cleared or set first. The source of information for the partial add gates varies depending upon the operation in which SC is being used. The nets that generate the data levels are sets of four AND gates ORed together; each AND gate receives an enabling level that is common to all nets and an input from a single bit of a source register. For example, the top set of AND gates places the complement of MB0–5 into SC3–8, i.e., with SC clear, a 0 in a given MB bit causes the partial add pulse to set the corresponding SC bit by complementing it. For this function, the first three data levels are automatically asserted. For the other three sets of AND gates, the data levels are asserted by ones in the source register: the second set enables input from MB6–11 to SC3–8 with the first three bits negated; the bottom two sets enable input to SC from AR0–8 and MB0–8.

To calculate the exponent in floating-point operations and the pointer in character operations, numbers must be added in SC. The addition is performed with essentially the same algorithm used in the arithmetic register (6.2b). First, the partial add pulse produces in SC the exclusive OR of the contents of SC with the number represented by the data inputs, then the carry pulse adjusts the partial sum to produce the arithmetic sum. The carry function for SC differs from that in AR in that no ripple carry is used. The carry connections from one flip-flop to the next are for indexing and are not associated in any way with the full-register carry function; no carries propagate from one bit to the next when a bit is complemented by the carry pulse. Instead, as soon as the partial sum is formed, a series of level transitions from right to left across the carry chain determines the carries for all bits, and all bits are adjusted simultaneously by the carry pulse. The conditions for a carry are the same as those in AR. There is no carry into the least significant bit (D8) and a carry out occurs only when two ones are added—a condition that is indicated by a 1 in the data and a 0 in the partial sum. For the other bits, there are two carry out conditions, one dependent upon a carry in. For any bit, there is a carry out if both summand bits are ones, or if the partial sum is 1 and there is a carry in. After the level changes have propagated through the chain defining the carries for all bits, the carry pulse complements those bits that receives carries; the SC8 carry supplies the gating level for SC7, and so on through the register. Since there can be no carry into the LSB, the carry gate for SC8 is disabled (B8).

b SC Control

The various functions of SC and FE are triggered directly from the subroutine time chains including the SC subroutines. There are no connections through the subroutine interface and SC functions are triggered by execute time pulses only prior to entry into a subroutine. The logic governing the SC and FE functions is shown in Figure 6-15. The conditions governing FE, which is used only for temporary storage, are quite simple: it is cleared only on the master clear (B6), and may receive an exponent from SC in the exponent calculate subroutine for floating multiplication or division (B7), or may receive the position portion of the pointer in the first part of a character operation (B4). In all cases, the information is subsequently transferred to SC (B2).

The complement transfer of MB18, 28-35 into SC (C8) is made at ET0 in a shift operation or floating scale. The only other function triggered from the execute cycle is the complement pulse, which occurs at ET1 in FSC if AR is positive (B3). All other functions govern SC for shifting or calculating in the data and arithmetic subroutine instructions, and they are triggered by pulses from the special time chains. In most instances, these pulses are ungated although many of those that trigger the complement are gated by sign conditions. Read the appropriate subroutine description (all are included in the final three sections of this chapter) to determine the significance of each SC event in a given subroutine. SC is cleared through the net in the upper left (which includes the master clear), complemented through the nets in the top center, and its partial add function is triggered through the net in the upper right. This last function is used for addition only in the SC add subroutine in which it is followed by the carry (B3); in all other cases, the exclusive OR is used for transfers. The source of information for the data levels to the partial add gates is determined by the enabling levels in the upper right of Figure 6-16. A flip-flop in the logic for the appropriate subroutine enables the required input for each transfer or partial addition. There are no flip-flops associated with the SC add subroutine; instead the enable level is derived from a flip-flop in the main subroutine that calls for the SC addition. SC may also be incremented by 1 through the nets shown below the enable levels.

In the lower left of Figure 6-15 are several control signals derived from the SC outputs. The function in D6 indicates that the first three bits are all ones and hence SC contains a 1's complement negative less than or equal to 63 in magnitude. The net in D1 decodes SC0-7 for all ones and its output is ANDed with SC8(1) for a signal indicating that a shift-count has been complemented, i.e., SC has counted to - 0 which is all ones. The termination of the count may also be indicated by a pulse through the pulse amplifier in C4. When SC0-7 are all ones,the next count pulse in theshift-count, multiply or divide subroutine triggers the PA to produce a leading edge at output SC8B. The PA output is ORed with the - 0 configuration, so SC8B remains asserted even after the PA output disappears.

c SC Subroutines

There are two subroutines associated with the shift counter, an add subroutine for use in calculations on exponents and pointers, and a shift and count subroutine that counts the number of steps required in an operation and shifts the intermediate result at each step. The logic for these subroutines is shown at the left in Figure 6-16 and the flow charts are in the lower right of Figure 4-8. Listed with the shift-count subroutine are the entry conditions, the control levels governing the type of shift, and the pulses to which the subroutine returns in the interrupted sequences. Similarly, the flow chart for add lists the entry, the source enabled for the partial addition, and the return for each call.

The time chain for add is in the upper left in Figure 6-16. The first pulse always clears the character control flip-flop CHF1, although this is of relevance only in character operations. SAT1 then triggers the partial add, and after a delay sufficient for all level transitions through the carry chain, SAT2.1 triggers the carry. SAT3 then returns to the interrupted sequence.

The shift-count subroutine is used only in character operations, shift operations, and the floating-add subroutine. All other shifting counted by SC is produced directly by pulses from the arithmetic subroutines. Entry into the shift-count sequence is at SCT0, which performs no operation but provides a delay before the first shift. If SC does not contain - 0 (indicated by the condition that either SC8 is not 1 or SC0-7 does not contain all ones), SCT1 increments SC and triggers the appropriate shift (6.5). If the counting is still incomplete, the output of the delay triggered by SCT1 again triggers SCT1 for a new shift and count. When the count is complete, the delay output returns to the interrupted sequence via SCT2.

6.5 SUBROUTINE INTERFACE

Because the same event is often required at many different times in the various subroutines, the processor includes a subroutine interface that collects signals from the subroutines to reduce the number of signals applied to the control logic for the arithmetic registers. For example, all shifts of AR or MQ are triggered through the nets at the left in Figure 6-17. The lower set of nets, which includes level gates, is for shifting on SAT1 in the shift-count subroutine. For a shift operation, AR and MQ are shifted left if bit 18 of the instruction word is 0, right if bit 18 is 1. Note that even though the program may request the shift of a single accumulator, the logic shifts both AR and MQ, the latter being empty. The level gates for other SC-controlled shifts are supplied by subroutine control flip-flops. The first part of a character operation requires an MQ left shift; this is followed in the second part by a shift left of both in the deposit sequence or an AR shift right in the load sequence. Floating addition requires a right shift of both registers. The upper set of gates allows pulses from the subroutine time chains to trigger shifts. The regular division shift is both registers left at DST14A, but two other divide pulses produce right shifts of AR alone and one left shifts MQ alone. Similarly the multiplication process shifts both to the right at MST2, but a final shift at the end of the subroutine moves MQ alone to the right. The normalize return begins by shifting both registers right in case there has been overflow in calculations with the fractions, and then the regular normalizing process shifts both to the left. In floating division, a 2-bit overflow is possible; so following the divide subroutine, the final pulse in the floating-divide instruction sequence (FDT1) shifts both registers right to supplement the single shift that begins the normalize return. The effects at the register extremities for all of these shifts are controlled by the special shift inputs shown in Figure 6-7 and described with the gating for the registers (6.2a, 6.3). The exact configurations for all shifts are shown in block diagram form in Figure 6-7 and each shift is shown at the appropriate place in the flow charts.

The remaining nets in Figure 6-17 are mostly for gates that collect subroutine time pulses for transmission to the register gating. In some cases, the pulse inputs are gated by levels, particularly in the upper right nets that detect overflow. All connections from subroutine time chains, other than for functions listed at the top of the figure, are made directly to the control logic for the registers. All subroutines pulses that trigger the AR negate or add subroutine, the switch of MB and MQ, or the AR complement are routed through the subroutine interface. For the other functions, which include entry to the AR subtract subroutine, the transfer of MQ to MB, the switch of MB and AR, and overflow, most pulses are routed through the interface but some are connected directly to the register gating. For example, the net in the upper right of Figure 6-17 handles overflow for multiplication, division, and all floating-point operations, but overflow in an arithmetic shift operation is handled by a net included with the flag logic (6.2e).

Except for one special case, all SBR flip-flops and control flip-flops in the subroutine logic are cleared at the beginning of every main sequence. For this purpose, the master clear triggers several pulse amplifiers to drive additional clear lines. Two of these with prefix MP are in the lower right of Figure 6-21, a third with prefix DS is in the upper right of Figure 6-26.

6.6 DATA SUBROUTINE INSTRUCTIONS

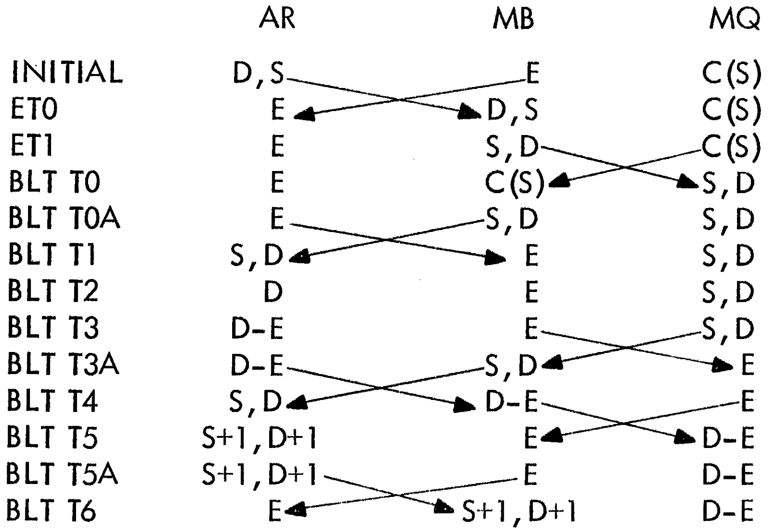
Three types of data transmission instructions switch to subroutines for their execution. Flow charts for all three types are in Figure 4-9. The block transfer moves an entire block of words from one area in memory to another. The character operations handle single characters smaller than a word and can insert a character into a word in memory or retrieve a character from a word without affecting the rest of it. Shift operations move the bits of a word or pair of words to the left or right. There are several shift configurations differentiated mainly by the effects of the shift on the register extremities. The last group may be viewed as logical operations rather than data transmission; the arithmetic shift is equivalent to multiplying the word by a power of 2.

a Block Transfer

A flow chart of the block transfer instruction is at the right in Figure 4-9 and the time chain is in Figure 6-18. The left and right halves, respectively, of the accumulator addressed by the instruction provide source and destination addresses S and D. The first fetch cycle retrieves a word from location S. The subroutine then stores the word in location D, increments both S and D, and returns to the fetch cycle to retrieve and store a second word according to the incremented source and destination addresses. The entire sequence is iterated until D equals the effective address E.

The first fetch cycle retrieves AC, swaps its halves so that S is available to MA, and fetches C(S). At the beginning of the execute cycle, AR contains (D,S), MB contains (0,E), and MQ contains C(S). The first execute pulse switches MB and AR to save E and bring (D,S) to MB. ET1 then clears MA and swaps the MB halves so ET3 can transfer D to MA. ET3 also triggers the first subroutine pulse BLT T0 (Figure 6-18, upper left) which switches MB and MQ to save (S,D) in MQ and make C(S) available to memory from MB. It also requests a memory write to store C(S) in D.

Upon receipt of the memory return, the BLT time chain transfers MQ to MB so that the addresses are now in both registers. The next pulse then switches MB and AR so that E is now in MB and (S,D) in AR. BLT T2 clears ARLT and the next pulse calls the subroutine to subtract E from D. Following the AR subroutine, BLT T3A places E in MQ and returns (S,D) to MB. BLT T4 then moves D-E to MB and the two addresses to AR. The next pulse moves the subtraction result to MQ, bringing E to MB, and triggers the subroutine that adds 1 to both halves of AR, incrementing both addresses. Upon the return BLT T6 saves E in AR, moves the new addresses to MB and reenters the fetch cycle at FT1A. This is the point just following the retrieval of an accumulator, so the processor repeats the entire procedure using the incremented addresses in MB as though they had just been retrieved from AC. For convenience, the following table shows the contents of AR, MB, and MQ following each pulse in the sequence (or following a subroutine called by the pulse).



	AR	MB	MQ
INITIAL	D,S	E	C(S)
ET0	E	D,S	C(S)
ET1	E	S,D	C(S)
BLT T0	E	C(S)	S,D
BLT T0A	E	S,D	S,D
BLT T1	S,D	E	S,D
BLT T2	D	E	S,D
BLT T3	D-E	E	S,D
BLT T3A	D-E	S,D	E
BLT T4	S,D	D-E	E
BLT T5	S+1,D+1	E	D-E
BLT T5A	S+1,D+1	E	D-E
BLT T6	E	S+1,D+1	D-E

Since E is initially greater than D, the result of the address subtraction is negative until the cycle following that in which the indexing of D makes it equal to E; then the result is 0 so the sign is positive. The transfers at BLT T4 and BLT T5 move the result to MQ, and BLT T5A tests MQ0 (D4, B8) to determine whether the block is complete. If MQ0 is 0 at this time, the program counter is incremented (the normal program counting at ET1 is inhibited throughout the block transfer) and the subroutine returns to ET10 instead of going on to BLT T6. There are no operations in the store cycle and the processor goes on to the next instruction.

Since a block transfer may use many main sequences, the subroutine includes provision for strobing the priority interrupt system at every BLT T4. If a PI request is generated, the level BLT DONE is asserted even though MQ0 may not be 0 (if it is, the subroutine terminates in the usual manner). This prevents the final MB-AR switch at BLT T6, so the incremented addresses are still in AR and BLT T5A goes directly to ET10. Since BLT LAST is negated (D6), there is no store-AC inhibit and the current addresses are stored in the accumulator in place of the original ones. Following the store cycle, the processor returns to the instruction cycle ostensibly to repeat the same instruction but it is interrupted by the PI request. After all requests have been serviced, the program returns to the interrupted block transfer, fetches the new addresses from AC, and begins where it had previously left off.

b Character Operations

There are five instructions in the character operation group, four of which require two main sequences for execution: the first part fetches and if necessary increments the pointer, the second handles the character designated by the incremented pointer. The flow chart for all character operations occupies the left half of Figure 4-9 and the logic for the two parts is shown in Figures 6-19 and 6-20. The top of the flow chart lists the different instructions, the main control levels governing their execution, and the configuration of the pointer. The first fetch cycle fetches the pointer according to the effective address calculated from the instruction; the address cycle in the second part calculates the effective address of the operand from the I, X, and Y portions of the pointer. Within the operand, the character is defined by the P and S portions of the pointer: S specifies its size; P specifies its position as the number of bits remaining to the right of the character in the word. Two of the instructions merely fetch the pointer in the first part and then one enters the load sequence in the second part, the other the deposit sequence. The load sequence retrieves a character of size S from position P in the word in location E and loads it right justified into AC. The deposit sequence fetches a character of S bits from the right end of AC and inserts it at position P in C(E). Two other instructions increment the pointer in the first part and use this new pointer in the load or deposit sequence that follows. The last instruction merely fetches and increments the pointer and then returns to the instruction cycle to continue the program, skipping the second part.

The major control levels for the first part are derived from the instruction command levels and two control flip-flops CHF5 and CHF7, both of which are set by the final pulse in the first part. The first flip-flop distinguishes the two parts, the second compensates for the fact that a PI request can interrupt the program between the two parts. The level CH INC is asserted during the first part of any instruction that increments the pointer (Figure 6-19A2). This level is ANDed with CHF7(0) to generate CH INC OP, whose assertion indicates that the first part must actually increment the pointer. If the pointer is incremented and a priority interrupt occurs, the program must not reincrement upon repeating the first part after returning to the interrupted sequence (the first part is repeated only to fetch the pointer that was lost). Unlike most control flip-flops, which are cleared by the master clear, the state of CHF7 (D3) must remain from the first to the second main sequence and is cleared by the master start via the flag clear (6.2_e_). When it is set by CHT9 at the end of the first part, the sequence returns to the address cycle at which point a priority interrupt may occur. If an interrupt merely executes a block IOT, CHF7 remains set for the return. However if there is a jump to a subroutine, the JSR saves CHF7 with the flags and clears it so that it may be used by the break routine. When the routine is complete, the restoring JRST again clears CHF7 via the flag clear and restores its original state from MB4 at the same time that it restores the flags. Then when the program repeats the first part, CHF7(1) inhibits CH INC OP and instead causes CH INC to assert CH ∿ INC OP (C5). This level is also asserted in the first part of the two instructions that do not increment the pointer. Finally, CHF7 is cleared by either sequence in the second part.

The first part fetches the pointer from location E: CH ∿ INC OP allows memory to rewrite the pointer, but CHINC OP requests a fetch and pause so that the incremented pointer may subsequently be deposited. Since there must be only one program count for each instruction, PC+1 is inhibited by all but the IBP (CAO), which uses only the first part. If there is no pointer incrementing, ET0 triggers the subroutine time chain at CHT6 (Figure 6-19B4); otherwise it starts at CHT1 (upper left). The first pulse transfers the pointer to AR and sets CHF1, enabling the zeros of MB0-5 as data inputs to SC so that the partial add at the next pulse loads -P-1 (i.e. ∿P) into SC. CHT3 then sets CHF2 and calls SC add. CHF1 is not cleared until the first pulse in the SC add chain to give the CHT2 partial add a little more time. CHF2(1) then enables the ones from MB6-11 for the partial add in the SC subroutine so that upon the return SC contains -P+S-1. If there are not enough bits left for another character; i.e., if S ≥ P + 1, the result in SC has a positive sign and the instruction must go on to the next location for the character. Thus if SC0 is 0, the chain continues to CHT4, which clears SC and calls an AR subroutine to index the address portion Y of the pointer to the next location. Upon completion, CHT4A loads - 37 (i.e., -P-1 for a character of 0 size) into SC and returns to CHT3 to call SC add again; this time the addition of S generates the complement of the position portion of the pointer for the new location. If SC0 is 1 at the junction following CHT3A, the chain skips to CHT5 (B3), which complements SC so that it now contains P-S, the position of the next character in the same location or the first character in the next location. Actually CHT4A loads - 229 into SC so that an S larger than 36 cannot put the processor into a loop. The result is then interpreted mod 64 so the correct position results. Thus if the pointer must go to the next location and S is larger than 36, the new P is 100 -S rather than 36 -S.

The chain then continues to CHT6, which is the starting point for any nonincrementing instruction. CHT6 again sets CHF2 but there is no subroutine call: CHF2(1) merely enables the MB6-11(1) data inputs to SC so that the partial addition at CHT7 transfers S into it. For

CH ∿ INC OP, the chain then skips to CHT8B. CH INC OP uses the CHT6 and CHT7 operations already mentioned because S is still available from the old pointer in MB; additional events specifically for CH INC OP are that CHT6 inserts the new P into the pointer by transferring it from SC3-8 into AR0-5 (dropping SC0-2 means SC mod 64 is transferred) and clears SC. CHT7 then moves the new pointer to MB after which CHT8 restarts the read/write memory cycle to deposit it. The return triggers CHT8B which clears both CHF6 and CHF2, transfers the new P from MB0-5 to FE and complements S in SC. If incrementing the pointer is the only operation required by the instruction, the subroutine terminates here and the sequence returns to ST7 for a new instruction cycle. For any character operation other than an IBP, CHT8B calls the shift-count (C2) which shifts MQ left S places loading ones in at MQ35. Upon the return, CHT8A clears SC and IR13-17, the latter in preparation for receiving the I and X portions of the pointer in the address cycle following the first part. Then CHT9 transfers P from FE to SC, sets CHF5 so that the next execute cycle will select the second part, and sets CHF7 in case there is a PI request at the beginning of the address cycle to which the sequence then returns.

The two chains for the second part are shown at the top in Figure 6-20. Both sequences start with MB containing the word retrieved according to the effective address of the pointer, and MQ containing a word made up of ones in the last S places at the right and zeros elsewhere. For the two load instructions, AR contains E. The initial gates include FC(E) as there will be no subsequent storage in E, and ET0 triggers the first pulse in the chain (A2). LCT0 moves the data word to AR and the mask to MB, complements P in SC, and calls the shift-count subroutine. The character is then right justified by right shifting AR P places. Following this, LCT0A transfers zeros from the mask into AR thus clearing all of AR except that part containing the desired character. The pulse also clears CHF7 and returns to ET10.

The two deposit instructions request a fetch and pause, and ET0 enters the deposit sequence (A4). DCT0 complements P in SC and calls a shift-count that moves AR and MQ left P places loading zeros in at the right in both registers. Upon subroutine completion, AR contains the character in the appropriate position and the ones in the mask are in the same position. The pair of pulses triggered by the return transfers the mask to MB, the ones from MB to MQ, and complements AR. Thus MB contains the mask, whereas MQ contains the data word other than in the character position which contains all ones. DCT1 then transfers zeros from the mask to AR clearing it other than in tne character position which contains the complement of the character. At the same time, the data word with ones in the character position is moved back to MB. The next pulse again complements AR so that it contains ones outside of the character, and DCT3 then inserts the character into the appropriate position in the data word by transferring zeros from the character (all other bits are ones) into the all ones portion of MB. This pulse also clears CHF7 and returns to ET10. The subsequent store cycle restarts the waiting memory cycle to deposit the data word in E.

If the program specifies a size greater than 36, the character is at most the entire word. For $P \geq 36$, no character is processed. If both P and S are less than 36 but $P + S > 36$, a character of size 36 - P is loaded from position P or the right 36 - P bits of the character are deposited in position P.

c Shift Operations

The lower part of Figure 6-20 shows the logic governing the shift operations and the flowchart for them occupies the right portion of Figure 4-9. The three combined instructions generate the level SHAC2 (B6) which causes the main sequence to fetch and store a second accumulator. The direction of the shift is specified by bit 18 of the instruction word (0 left, 1 right), and the number of places to be shifted is specified by bits 28 to 35.

At the beginning of the execute cycle, AR contains AC; and for a combined shift, MQ contains a second AC. ET0 transfers the complement of MB18, 28-35 into SC. Since left shifting is considered to be positive and right shifting negative, it is assumed that if bit 18 is 1, bits 28 to 35 contain the 2's complement of the number of shifts desired. Thus if MB18 is 0, SC is already correct and contains the complement of a positive number; however if MB18 is 1, SC contains a positive number one less than the number of shifts. Thus MB18(1) gates ET1 to trigger SHT0 (B5), which adds one to SC. Then ET3 starts the subroutine chain for all instructions and the first pulse SHT1 complements SC on the condition MB18(1) so that SC now contains the correct complement. SHT1 also calls the shift-count, which counts SC up to all ones and at each count shifts AR and MQ left or right according to the state of MB18. The shift connections to the registers are made through the subroutine interface (6.5), and the special shift inputs that control the shift actions at the register extremities are shown in Figure 6-7 and described with the AR and MQ gating (6.2a, 6.3). Block diagrams below the flowchart show the configurations for all twelve types of shift.

Since an arithmetic shift multiplies fixed-point numbers by powers of 2, an overflow condition is included in case significant bits are lost in a positive shift. In a single- or double-left arithmetic shift, the overflow flag is set if a 1 is shifted out of AR1 in a positive number, or a 0 in a negative number (6.2e). The return from the shift-count triggers SHT1A which retruns to the main sequence at ET10.

6.7 ARITHMETIC INSTRUCTIONS

This section describes fixed multiply and the floating-point instructions. Each of these instructions goes from the execute cycle to a special sequence which may or may not call an arithmetic subroutine. Fixed add and subtract are both performed within the execute cycle and are described with the arithmetic register (6.2). Fixed divide enters directly into the divide subroutine from ET0 and is described with that subroutine in 6.8c.

a Fixed Multiply

A flow chart of the two fixed-multiply instructions is in the left part of Figure 4-8; Figure 6-21 shows the special time chain for them. Both integral and fractional multiplication use AC and either C(E) or E itself as operands, but the product in the latter case is a double-length fraction, whereas in the former it is assumed that the desired integer is in the low-order half of the double-length product. Both instructions enter the special sequence at ET0 by triggering MPT0 (Figure 6-21, upper left). This pulse sets the appropriate bits in SC to count 35 steps and sets MPF2 if both operands are negative. It also calls the multiply subroutine (6.8b) and waits until MST6 returns to MPT0A. If both operands are negative and the result is also negative, MPT0A sets the overflow flag. This can occur only if -1 is multiplied by -1, whose answer +1 overflows generating the representation for -1. If integral multiplication is being performed (IR6(0)) and the result is negative, MPT0A complements the high-order half in AR. At this point, the fractional process is complete and the sequence returns to ET10 via NRT6, the final pulse in the normalize return subroutine. In the store cycle, the low-order half is stored in a second accumulator for all but the memory mode wherein no accumulator is stored. For integral multiplication, MPT0A continues the chain to MPT1, which transfers the resulting integer in the low-order half from MQ to MB and sets the overflow flag if the high-order half is not clear. MPT2 then transfers the result to AR so that the sequence can make use of the standard transfer and store functions following the return to ET10 via NRT6.

b Floating Scale

This instruction allows the program to change the exponent of a floating-point number without affecting the fractional part. The number in AC is multiplied by 2^y where y is the number contained in bits 28 to 35 of the effective address. This number is interpreted as positive or negative in 2's complement notation as the sign, bit 18, is 0 or 1. The flowchart for the instruction occupies the left quarter of Figure 4-10, and the time chain is shown in the lower right of Figure 6-19.

The first pulse in the execute cycle transfers the complement of MB18, 28-35 to SC. The next two pulses then adjust SC according to the sign of the number in AR: if positive, ET1 complements SC; if negative, ET3 adds one to SC by triggering FST1. In either case, ET3 triggers FST0, which calls the SC add subroutine. During the subroutine, the 1 state of FSF1 enables the AR0-8(1) data inputs to SC. The return pulse FST0A transfers the new exponent from SC1-8 to the exponent part of AR; and if the signs of AR and SC are different, it sets the overflow flag.

To see that the above sequence of events produces the correct exponent and properly detects overflow or underflow, consider the various cases keeping in mind that the floating-point exponents from -128 to +127 are represented by the numbers 0 to 255 and that the scale factor in E is in 2's complement notation. Let x and y be the absolute values, respectively, of the exponent part of AR and the scale factor in MB. Thus if AR is positive, AR1-8 contains x; otherwise $\sim x$, i.e., 255 - x. On the other hand if MB is positive, MB28-35 contains y; otherwise the 2's complement, i.e., 256 - y. At each step, C(SC) is a function of the signs of AR and MB as follows:

	AR+, MB+	AR-, MB+	AR+, MB-	AR-, MB-
ET0	-[255 - y]	-[255 - y]	+[y - 1]	+[y - 1]
ET1	+[y]	-[255 - y]	-[256 - y]	+[y - 1]
ET3	+[y]	-[256 - y]	-[256 - y]	+[y]
SC ADD	+[x + y]	-[255 - (x + y)]	+[x - y]	-[255 - (x - y)]

Hence with no overflow or underflow, SC and AR have the same sign and SC contains the proper representation of the new exponent. However, if in the first two columns $x + y > 255$ (overflow)or in the last two $x - y < 0$ (underflow), AR and SC have opposite signs.

c Floating Add-Subtract

Both of these instructions use the floating add time chain shown in the upper half of Figure 6-22; the flow chart occupies the center portion of Figure 4-10. For floating add, ET0 triggers the special sequence (Figure 6-22A1); but for floating subtract, ET0 switches MB and AR so that the subtrahend is then in AR, and ET3 calls the negate subroutine. Since the number to be subtracted has now been replaced by its negative, the operands may instead be added and ET4 triggers the floating add sequence.

If the signs of the operands are the same, FAT0 complements SC to all ones. It also sets FAF1 enabling the MB0-8(1) data inputs to SC so that the partial addition at FAT1 produces a 1 transfer if the signs are different, but transfers the complement of the MB sign and exponent if the signs are the same. In either case, the sign of SC is opposite that of AR. FAT1 also triggers SC add, and shortly after, FAT1B clears FAF1 and sets FAF2 (A3) to enable the AR0-8(1) data inputs to SC. Since the signs of AR and SC are different, at least one exponent is represented by its complement, and hence the result of the addition in SC is a 1's complement negative of the difference between the exponents unless there is overflow. Let x and y be the absolute values of the exponents of the numbers in AR and MB. The possible signs, exponent representations, and results are as follows:

AR, MB signs	AR0-8	SC Before	SC After	Result
AR+, MB+	$+[x]$	$-[255 - y]$	$-[255 - y + x]$	$y \geq x$: signs $\neq$
AR+, MB-	$+[x]$	$-[255 - y]$	$-[255 - y + x]$	$y < x$: signs $=$
AR-, MB-	$-[255 - x]$	$+[y]$	$-[255 + y - x]$	$y > x$: signs $\neq$
AR-, MB+	$-[255 - x]$	$+[y]$	$-[255 + y - x]$	$y \leq x$: signs $=$

Hence, if the MB exponent is greater than the AR exponent, the AR and SC signs differ after the addition. If the AR exponent is greater, the signs are the same and FAT1A switches MB and AR because the number with the smaller exponent is the one that is shifted. If the exponents are equal, the signs may or may not be the same but it matters not whether the transfer takes place. Since the result in SC is a 1's complement, no further action on it is necessary if it is negative, and in this case FAT1A jumps directly to FAT4 (A6). However if overflow has produced a positive result, the number in SC is one less than the difference between the exponents (since a 2's complement addition was performed on numbers in 1's complement notation); in this case FAT1A triggers FAT2 which adds one to SC, and FAT3 complements it in preparation for the shift-count.

If the number with the smaller exponent must be shifted more than 63 places (a condition represented by at least one 0 in SC0-2), the addition can affect neither the fraction nor the low-order part so FAT6 clears AR (A8). However if fewer than 64 shifts are required, FAT5 ensures that all bits to the left of the fraction MSB are of no significance by loading the sign into AR1-8, sets FAF3(B4) to generate the correct arithmetic shift inputs to the registers extremities (6.2a, 6.3), and calls the shift-count subroutine. After the number has been right shifted in AR and MQ so that its bits correctly match the MB bits in order of magnitude, the return triggers FAT5A which follows directly from FAT6 if there is no shifting. FAT5A clears SC and again sets FAF1 to enable the MB0-8(1) data inputs. Then FAT7 changes SC to all ones if MB is negative, and the partial add at FAT8 loads SC with the MB sign and exponent or their complements depending upon the state of SC. Thus SC always receives a positive sign and the absolute value of the exponent. The next pulse in the chain then nullifies the exponent portion of MB, and FAT9 calls AR add. If AR was cleared at FAT6, the addition merely transfers MB to it. The return triggers FAT10 which clears those control flip-flops that are still set and enters the normalize return subroutine. The return via NRT6 is directly to ET10 for storage of the result.

d Floating Multiply and Divide

These two simple sequences do little more than call the three subroutines necessary for the execution of floating-point multiplication and division. The flow charts are at the right in Figure 4-10 and the logic is shown in the lower half of Figure 6-22. Both chains are triggered at ET0 and the first pulse in each calls the exponent calculate subroutine. The return from FPT4 places in SC the complement of the number of steps required (27 for multiply, 30 for divide) and enters the appropriate subroutine. The return from multiply at MST7 triggers

FMT0B, which transfers the calculated exponent from FE to SC, sets NRF2 to set up the normalize shift gates, and enters normalize return. NRT6 returns directly to ET10 for storage of the result.

The divide subroutine first tests that division can be performed. If the divisor is less than or equal to half the dividend, the sequence sets the overflow flag and jumps directly from DST13 to ST7—the only normalized number that fails to satisfy this condition is a zero divisor. If the division is executed, DST21A returns to FDT0B, which transfers the calculated exponent from FE to SC and sets NRF2 to set up the normalize shift gates but does not enter the normalize return subroutine. This subroutine can compensate for only one bit overflow whereas floating divide, by doing 30 steps, deliberately generates two extra quotient bits, one for possible overflow, the other for rounding. The divide instruction sequence includes an extra time pulse, FDT1, which shifts AR and MQ right to compensate for the rounding bit. It then enters normalize return and NRT6 returns directly to ET10 for storage of the result.

6.8 ARITHMETIC SUBROUTINES

Besides the simple AR and SC subroutines, there are four arithmetic subroutines that are called by more than one arithmetic operation. Floating multiplication and floating division both begin with a subroutine that calculates the exponent. The multiply subroutine is used by both fixed and floating multiplication, the divide subroutine by both fixed and floating division. All floating-point instructions except floating scale call the normalize return subroutine to normalize the result of their arithmetic computations.

a Exponent Calculate

Floating multiply and divide call a subroutine to calculate the exponent before beginning operations on the fractions. The flow chart for exponent calculate is at the left in Figure 4-11 and the logic for the time chain is shown in Figure 6-23. The nets at the right in the logic drawing generate several levels necessary for execution of the subroutine. These are exclusive OR and equivalence functions of the operand signs and FMF1, the last term providing a distinction between multiplication and division. In multiplication the exponents are added, whereas in division the exponent of the divisor is subtracted from that of the dividend. Since

FMF1 is the SBR for the return to multiplication from the exponent calculation, it is 1 in multiplication but 0 in division; therefore, the sign functions for multiplication are exactly the opposite of those for division.

The first time pulse in either multiply or divide triggers the floating-point time chain (Figure 6-23, upper left). The first floating-point pulse sets SC1 thus loading 128 into SC. The next pulse complements SC if the sign of AR is different from the state of FMF1, i.e., the complement occurs if AR is positive in multiplication or negative in division. FPT1 also enables the AR0-8(1) data inputs to SC and calls the SC add subroutine. At the return FPT1A complements SC if the exclusive OR of AR0, MB0, and FMF1 is false; otherwise it adds one to SC. The latter action is triggered by FPT2, which is generated only if the appropriate condition holds (upper right); the gating for the complement is included in the complement net for SC. The next pulse then calls another SC addition, this time using MB0-8(1) as the SC data inputs enabled by FPF2(1). After this addition, the result is correct for two of the four cases; for the other two, MB0 and FMF1 the same, FPT1B complements SC. Then FPT3 transfers the calculated exponent to FE, clears SC, and nullifies the exponent portions of both MB and AR by loading the appropriate sign into bits 1 to 8. FPT4 returns to the interrupted special sequence.

First, consider multiplication. FMF1 is set, thus at FPT1 the complement occurs if AR0 is 0; then at FPT1A if the signs of AR and MB are not equal, SC is again complemented, otherwise it is indexed. Finally at FPT1B, SC is complemented if MB is negative. On the other hand, FMF1 is 0 for division, so the complement at FPT1 occurs if AR is negative; at FPT1A complementing occurs if AR0 and MB0 are the same, and indexing otherwise. Finally, FPT1B complements SC if MB is positive.

To see that these operations give the correct result, let x and y be the true exponents of AR and MB. Since excess-128 code is used, the exponent portion of AR is x + 128 if the number is positive; but if negative, it is 255 - (x + 128) = 127 - x. Since the result must also be in excess-128 code, the sum of the exponents must be reduced by 128 for multiplication; whereas, in division the difference must be increased by the same factor. For multiplication, the above sequence of events operates in SC as follows (the table items show the sign of SC and the contents of SC1-8 as a positive number):

	AR+, MB+	AR+, MB-	AR-, MB+	AR-, MB-
FPT1	-[127]	-[127]	+[128]	+[128]
SC ADD	+[x - 1]	+[x - 1]	-[255 - x]	-[255 - x]
FPT1A	+[x]	-[256 - x]	+[x]	-[256 - x]
SC ADD	+[128 + x + y]	-[127 - x - y]	+[128 + x + y]	-[127 - x - y]
FPT1B		+[128 + x + y]		+[128 + x + y]

For every case, the result in SC is the correct exponent for the product (but expressed in positive form), unless there is overflow, as indicated by a negative result (SC0 = 1). The exponent remains in positive form during the multiply subroutine, and the subsequent normalize return subroutine checks for overflow and puts the exponent into correct form. For division, the sequence of events is:

	AR+, MB+	AR+, MB-	AR-, MB+	AR-, MB-
FPT1	+[128]	+[128]	-[127]	-[127]
SC ADD	-[x]	-[x]	+[254 - x]	+[254 - x]
FPT1A	+[255 - x]	-[1 + x]	+[255 - x]	-[1 + x]
SC ADD	-[127 - x + y]	+[128 + x - y]	-[127 - x + y]	+[128 + x - y]
FPT1B	+[128 + x - y]		+[128 + x - y]	

which again gives the correct result in positive form.

b Multiply

A single subroutine handles multiplication for both fixed point and floating point; the two types differ only in the number of steps—35 for fixed point, 27 for floating point. The subroutine flow chart occupies the center portion of Figure 4-11; Figure 6-24 shows the time chain. A pulse in the fixed- or floating-multiply sequence loads the complement of the number of steps into SC, sets an SBR and calls the subroutine. To control the AR and MQ extremities in multiply shifting, the two SBR flip-flops are ORed by the net at the right in Figure 6-24.

For fixed point, the subroutine time chain is triggered by MPT0, which is equivalent to ET0; for floating point, the entering pulse is FMT0A, which follows the exponent calculation. Either pulse triggers MST1 (upper left), which moves the multiplier from MB to MQ and the multiplicand from AR to MB. AR is cleared shortly thereafter. Form MST1, the sequence may continue to either MST2, MST3, or MST4 depending upon the relationship between bits 35 and 36 of MQ, but every step includes the incrementing of SC and the shifting of AR and MQ right one place. Initially, MQ35 contains the LSB of the multiplier and MQ36 is 0. On each successive step, MQ35 contains the next more significant bit of the multiplier and MQ36 contains the bit MQ35 held on the previous step. Arithmetic operations are performed whenever there is a transition from 1 to 0 or vice versa in a pair of multiplier bits, rather than on the basis of a particular bit being 1 or 0. At the 3-way decision following MST1, there is no arithmetic action if the two bits are equal and in this case the chain goes to MST2. If there is a transition from 1 to 0 in the bit pair, the chain goes to MST3, which calls the AR add subroutine. For a transition in the opposite direction, MST4 calls AR subtract. Either return triggers MST3A, which goes to MST2. Thus MST2 appears in every loop whether any AR subroutine is called or not. This pulse increments SC and shifts AR, MQ right one place. It then returns the sequence to the 3-way decision to check the next bit pair (which has just been shifted into MQ35, 36).

At the 3-way decision, the equality branch has the additional condition that the step count is not complete, i.e., SC does not contain 777. In the final shift, MQ36 receives the multiplier MSB and MQ35, the sign. If these two bits are equal, MST2 leaves the loop, jumping to MST5 (lower left). If they are not equal, MST2 returns to the 3-way decision, only two of whose branches are now open—those which necessarily call AR subroutines. Since the step count is complete, the return jumps directly from MST3A to MST5. This pulse clears SC, shifts the low-order half of the product in MQ to the right, and makes its sign equal to that of the high-order half in AR. MST6 then provides the return to the appropriate instruction sequence.

The multiplication sequence is as follows: Initially, MB contains the multiplicand, MQ the multiplier, and AR is clear. AR and MQ are connected so that the low-order bits of the product are shifted into MQ as the multiplier is shifted out. At each step, the current bit of the multiplier is available at MQ35, and the effect of the multiplicand in MB on the partial sum in AR is one binary order of magnitude greater than in the preceding step because the partial sum was right shifted. Thus MB can be combined directly with AR. Since MQ36 is initially 0, the sequence shifts without calling an AR subroutine until a 1 is shifted into MB35.

At this transition, the sequence jumps to MST4 to subtract the multiplicand from AR, which is clear. The shifting then continues until the next transition (which will be from 1 to 0), at which time the sequence goes to MST3 to add the multiplicand to the previous partial sum. The process continues in this way, subtracting the multiplicand at every transition from 0 to 1, adding at every transition from 1 to 0. At the end, AR contains the correct sign and the high-order half of the double-length product. The low-order half is in MQ0-34 with the sign of the multiplier in MQ35. Thus, there is an additional right shift of MQ to move the correct sign from AR0 to MQ0 and to place the low-order fraction in the MQ magnitude positions.

To see that this procedure results in a correct product, consider the positive binary integer:

$$\begin{array}{ccccccccc} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{array}$$

(The decimal digits below the binary digits are the powers of 2 corresponding to the bit positions.) This number is obviously equal to:

$$\begin{array}{lr} & 100000000 \\ + & 111000 \\ + & 11 \end{array}$$

Now an n-bit string of ones whose rightmost bit corresponds to 2^k is equal to $2^{k+n} - 2^k$, or equivalently $2^k(2^n - 2^0)$; i.e., $2^n - 2^0$ is a string of n ones and the 2^k shifts the string left k places. Thus:

$$\begin{array}{rcccc} 100000000 & = & 2^{8+1} - 2^8 & = & 2^9 - 2^8 \\ 111000 & = & 2^{3+3} - 2^3 & = & 2^6 - 2^3 \\ 11 & = & 2^{0+2} - 2^0 & = & 2^2 - 2^0 \\ \hline 100111011 & & = & & 2^9 - 2^8 + 2^6 - 2^3 + 2^2 - 2^0 \end{array}$$

In this last representation, each power of 2 that is subtracted corresponds to a transition from 0 to 1 (in the direction of increasing significance), whereas each that is added corresponds to the opposite transition. The largest term corresponds to the transition to the sign bit, which is 0 for a positive number. The multiplication algorithm in PDP-6 interprets the multiplier in this manner, alternately subtracting the multiplicand from the partial sum and adding it to the partial sum in the order-of-magnitude positions corresponding to the transitions. If a multiplier of the same magnitude were negative, it would have the form:

1 0 1 1 0 0 0 1 0 1
- 8 7 6 5 4 3 2 1 0

in which the extra bit at the left represents the sign. The number is now equivalent to:

$$-2^9+2^8-2^6+2^3-2^2+2^1-2^0$$

wherein opposite signs correspond to opposite transitions. The algorithm may thus use exactly the same sequence for a negative multiplier: this time the subtraction of greatest magnitude is detected by the transition to the sign bit, which is now 1.

c Divide

The time chain for the divide subroutine is on two logic drawings, Figures 6-25 and 6-26; Figure 4-12 is the flow chart. Entry is at ET0 in fixed point, at FDT0A in floating-point. Either pulse loads SC with the complement of the number of steps required—30 for floating-point, 36 for fixed-point—through pulse gates at individual bits. For this purpose, ET0 gated by IR DIV generates DS DIV T0 (Figure 6-25A8). Although the subroutine is triggered only by ET0 or FDT0A, there are nevertheless six entries, depending upon the sign of the dividend in AR and upon whether the fixed division is integral or fractional. To make the latter distinction, the fixed-point command level is further decoded into a pair of levels according to the state of IR6 (B1).

For a negative dividend, entry is at DST0 (upper left) for integral or floating-point operations. The first subroutine pulse sets DSF7 (D6) to remember that the dividend was originally negative, and calls the AR negate subroutine. Following negation, the sequence jumps to DST10 for floating-point but continues to DST1 for division of integers. This is also the entry for integral division with a positive dividend. Integral division uses a double-length dividend just as does fixed-point fractional division, but only one accumulator is fetched and it is used for the low-order half. Since the fetch cycle brings AC to AR, DST1 moves the divisor from MB to MQ and the significant half of the dividend from AR to MB. DST2 then switches MB and MQ, placing the dividend in low-order position, and clears the high-order half in AR. The sequence then jumps to DST10 for computations.

The only other entry earlier than DST10 is for a negative fixed-point fraction, wherein ET0 enters at DST3 (A6). For all other entries, the dividend is either one word in length or is already positive. The sequence starting with DST3 is necessary to change a double-length negative dividend into positive form. DST3 sets DSF7 because the dividend is negative and interchanges MB and MQ. DST4 then switches MB and AR so the low-order half is now in AR, and DST5 calls AR negate. After the return (B3), the sequence branches depending on whether AR is clear. The 2's complement of a double-length number is formed by complementing the entire double word and adding one into the LSB. A carry into the high-order half is necessary if the 2's complement of the low-order half is null (the complement was all ones before the LSB addition). Thus if AR is clear, the sequence goes on to DST8, 9: this reverses the preceding interchanges, returning all words to their original positions, and DST9 calls AR negate to form the 2's complement of the high-order half (i.e., to complement it and then add one as required). If AR is not clear, the sequence goes instead to DST6, 7 which also returns all words to their original positions but merely complements AR.

The computational part of the subroutine begins at DST10, which is triggered through the net at the right in Figure 6-25. This pulse begins the subroutine for a positive dividend in floating point or fractional fixed point. Otherwise, it follows from one of the preliminary sequences discussed above. Entry is direct from DST2 or DST7; from DST0A, it is conditioned by ∿ DSDIVI, which in this case represents floating point; and if DST9 calls AR negate, entry is made at the subroutine return. For a floating division, DST10 triggers DST10A (lower left) which right shifts AR, moving the LSB of the high-order half into MQ0, so the double-length dividend now has the sign bit in both AR0 and AR1 and the 70-bit magnitude in AR2-35 and MQ0-35. This action closes up the hole between the two halves of the dividend but it also divides the dividend by 2. For fixed division the sequence goes to DST10B, which left shifts MQ, closing the hole but leaving the magnitude the same. The shift brings the complement of AR0 into MQ35 because of the division shift connections, but this is of no significance.

The next step in the process is to compare the divisor with the dividend to determine whether the division can be performed. Thus, the next pulse is either DST11 or DST12 depending upon whether the divisor is negative or positive. The former pulse calls AR add; the latter, AR subtract. Since the dividend is positive, the result of the computation is also positive if the magnitude of the divisor is less than or equal to that of the number in AR. There are actually three different conditions being tested here for the three types of division. For a fixed integer, AR is clear and the result is positive only for a 0 divisor; if the divisor has anything of significance in it, the quotient cannot possibly be greater than $2^{35}-1$ which does not overflow. For a fixed fraction, the divisor is actually subtracted from the dividend and no overflow is allowed. For a floating fraction, the divisor is subtracted from half the dividend (the preceding right shift having divided it by 2), so there may be one bit overflow but the normalize return automatically compensates for this. Since all floating fractions are normalized, the only case that fails the test is that of a 0 divisor (a dividend of -1 could also fail but this number cannot occur as the result of any floating-point operation). After the return from the AR subroutine, DST11A tests the result. If it is positive, the sequence continues to DST13, which sets the overflow flag and jumps directly to the end of the store cycle. If AR0 is 1, the overflow cannot be greater than allowed in the particular type of division in progress, and DST11A continues the sequence to DST14A (Figure 6-26B7).

The next few pulses comprise the loop that performs the actual division. In division on paper, one subtracts out the divisor the number of times it goes into the dividend, then shifts the dividend one place to the left (or the divisor to the right) and again subtracts out. In binary computations, the divisor goes into the dividend either once or not at all at each step. The loop thus subtracts the divisor to generate a single bit of the quotient. If the subtraction does not overflow, i.e., if the dividend is larger than the divisor, the sign of the result is positive and a 1 is entered into the quotient. If there is overflow, a 0 is entered. To compensate for the overflow, one could add the divisor back into the dividend before going to the next subtraction step. However the PDP-6 algorithm instead shifts first and then adds the divisor back on in the new position. It then continues to shift and add putting zeros into the quotient until the result again becomes positive. This procedure generates the same quotient without ever going back a step.

The following processor operations correspond to the procedure outlined above. If the initial test subtraction produces a negative result, the sequence enters the loop at DST14A, which increments SC and triggers the division shift of AR, MQ to the left. This shift is equivalent to a combined rotation except that MQ35 receives the complement of AR0. The complement of the sign of the result is the next bit of the quotient: if the divisor does not go into the dividend, the result produces a 0 quotient bit. Each loop loads one bit of the quotient into

MQ35, and the low-order half of the dividend is shifted out from MQ as the quotient is shifted in. Following DST14A, the loop checks the previously generated quotient bit in MQ35 to determine what action to take next. If the quotient has received a 1, the divisor is subtracted; for a 0, it is added. The gate is the exclusive OR of MQ35 and MB0 (Figure 6-26, upper left) because the divisor may have either sign. If MQ35 is 1, DST14 subtracts the divisor if positive, or DST15 adds it if negative. A 0 quotient bit reverses the functions. Either DST14 or DST15 sets DSF5 (lower left) to gate the AR subroutine return which goes back to the beginning of the loop (A7). The loop iterates 36 times for fixed division, 30 times for floating. The first bit loaded into MQ35 is not actually part of the quotient but rather the sign bit to the left of the binary point. It must be 0 because the sequence enters the loop only if the divisor is larger than the dividend. The 36 fixed-point iterations generate the sign and 35 magnitude bits of the fractional or integral quotient. Floating-point requires 30 steps for the 27-bit fraction because there may be one bit overflow and an additional quotient bit is computed to allow running.

The test for termination of the process is made not at the end of the loop but in the middle. Thus each step is not shift and subtract, but rather subtract and shift, the first subtraction occurring before the loop. The test for completion follows DST14A. When this pulse generates the final shift and increments SC to 777, the assertion of the SC completion signal prevents the delayed DST14B from continuing in the loop (upper left) and it instead jumps out to DST16.

The remaining operations in the subroutine generate the correct remainder, adjust the signs of quotient and remainder, and place them in correct position. The final shift in the loop places the last quotient bit in MQ, but it leaves the remainder off one place to the left. Therefore DST16 right shifts AR. Since all operations have been performed on positive operands, the remainder should also be positive. If it is already, the sequence jumps directly to DST17A (C4). A negative remainder indicates that too much has been subtracted from the dividend, so either DST17 or DST18 adds the divisor back in (B4). Again this addition may call either AR subroutine, depending upon the MB sign. DST17A then checks the sign of the original dividend: if DSF7 is 0, the sequence jumps to DST19A (lower left); but if the dividend was originally negative, DST19 (A6) calls the AR negate subroutine. Thus at the end, the remainder has the same sign as the dividend. The setting of the SBR for the return from AR negate (lower right) is delayed slightly to prevent the return from the previous subroutine from getting through both SBR. DST19A then moves the quotient from MQ to MB, and the divisor in the opposite direction. Next DST20 (A6) clears SC and switches the remainder presently in AR with the quotient in MB. The sequence then determines the proper sign for the quotient by checking the exclusive OR of DSF7 and MQ0, which is generated by the net in the lower left. If the operand signs are the same (A5), the result is already in correct form and the sequence jumps to DST21A. If the signs are different, the result should be negative and DST21 calls AR negate. DST21A then moves the remainder to MQ where it is available for storage in a second accumulator. The subroutine returns to ET9 for fixed division, FDTOB for floating division. The reader should note at this point that the remainder is the correct one for a fixed-point instruction; but in the floating case, the remainder in MQ9-35 is correct for a 29-bit quotient which cannot be stored in its present form. For further particulars, refer to the normalize return subroutine (d,p. 6-46).

As an example of the way this algorithm operates, consider a division of 3-bit fixed-point fractions with a dividend of +.100100 and a divisor of +.101. By paper computation, we obtain the quotient this way:

```
          .111
  101 ) 100.100
         10 1
         10 00
          1 01
            110
            101
              1
```

Assuming the computer registers to be four bits in length, AR contains 0.100, MQ has 0.100, and MB has 0.101. Before starting the division, MQ must shift left to close the hole, giving MQ = 1.001. The sequence has four steps as follows:

```
         0.100 | 1.001
        -0.101
         1.111 | 1.001
    1 ←  1.111 | 0.010
        +0.101
         0.100 | 0.010
    2 ←  1.000 | 0.101
        -0.101
```

```
          0.011 | 0.101
   3 ←    0.110 | 1.001
         -0.101
         ------
          0.001 | 1.011
   4 ←    0.011 | 0.111
     →    0.001
```

The quotient is in MQ at the right, the remainder in AR at the left.

d Normalize Return

All floating-point instructions that operate on the fractional parts of the operands end by calling a subroutine to normalize the result. A floating-point number is considered normalized if the MSB of the fraction (bit 9) is opposite in state to the sign bit or if the magnitude of the fraction is 1/2 (the fraction -1/2 has the same magnitude representation as +1/2). The normalize return flow chart occupies the right half of Figure 4-11, and the time chain is shown in Figure 6-27.

Entry into the normalize return chain is immediate from FAT10, the last pulse in the floating add sequence (upper left). The first pulse in the chain, NRT0.5, sets NRF2 (C5), which enables the shift inputs at the AR and MQ extremities for a double-length arithmetic shift (6.2e, 6.3). NRT0 then adds one to the exponent in SC (which is in positive form, see 6.7c, 6.8a) and right shifts AR, MQ to accommodate any overflow that may occurred in the fractional computations. Since a right shift is equivalent to dividing by 2 and indexing the exponent is equivalent to multiplying by 2, the result is unchanged. The next pulse tests for a 0 result. Since the answer has already been shifted right, the net in the center of the figure generates the gating level as the AND function of AR = 0 and MQ1(0). The test can include all of AR because any operation on the fractions that produces a 0 result clears the exponent part of the register. If there is a 0 result, NRT.1 jumps directly to NRT6, the final pulse in the subroutine (B4). For a nonzero result, the chain goes to NRT1, which complements SC (so the exponent is now in negative form) and enters the normalizing loop (A5).

Multiply skips the right shift because it cannot overflow. The last pulse in the multiply sequence, FMT0B, sets NRF2 and after a delay that allows the exponent in SC to settle (C3) enters the chain at the 0 test. MQ1 is unnecessary for the 0 test in this case, but its inclusion merely allows attainment of a significant result with certain unnormalized operands for which the result would otherwise vanish. For divide, there must be two right shifts, one to compensate for a possible overflow bit, the other because of the extra quotient bit generated to allow rounding (c above). FDT0B, the pulse in the divide sequence generated by the return from the divide subroutine, sets NRF2 and transfers the calculated exponent from FE back to SC. The final pulse in the sequence, FDT1, then triggers the required extra right shift of AR,MQ; the regular right shift occurs in normalize return. Divide enters at the beginning of the chain but is delayed slightly so that the total delay between FDT1 and NRT0 is ample to allow the preliminary shift-count to settle down. Even though there are two shifts, only MQ1 need be included in the zero test because MQ2 contains an extra quotient bit generated only for rounding.

In the normalizing loop, if AR9 and AR0 are the same and the fraction is not of magnitude 1/2 (the gating levels are generated respectively by the net in C2 and the decoding nets for AR, 6.2), NRT1 triggers NRT2. This pulse adds one to the negative exponent in SC thus decreasing its magnitude, and triggers an arithmetic left shift of AR,MQ. NRT2 also enters a delay whose output retriggers NRT2 if the number is not yet normalized. The loop continues until either AR9 and AR0 differ or the fraction has magnitude 1/2, at which time NRT2 goes on to NRT3. If SC is now positive, NRT3 sets the overflow flag. This does not necessarily mean that the normalizing loop counted from negative into positive exponents—the overflow or underflow may have occurred when the exponent was calculated before multiplication or division. Since the loop uses the negative representation of the exponent, it is already in proper form for insertion into the result if the result is also negative. However, if AR0 is 0, NRT3 complements SC.

At this point, the subroutine must determine whether any rounding action is required. The net that generates the round gate is in C3 and C4. The program specifies rounding by a 1 in IR6, but the gate is asserted only if a rounding action is necessary, i.e., if the MSB of the low-order part of the fraction is 1. The rounding sequence requires the repetition of the entire subroutine, so the assertion of NR ROUND first causes NRT3 to complement SC, making the exponent positive again for the initial right shift. Then NRT3.1 continues the chain to NRT5, which calls the subroutine that adds one to AR. The completion triggers NRT5A (lower right) which returns to the beginning of the subroutine (C2). Thus the entire routine is repeated in case the rounding has overflowed (which can occur only if the fraction was all ones). NRT5A also sets NRF3 (D3) disabling the round net. Then when the chain again reaches NRT3.1, it automatically continues to NRT4, to which it goes on the first pass if no rounding is required. NRT4 transfers the exponent from SC to AR so the result is now complete and in proper floating-point form. This pulse also triggers NRT6 which returns to ET10 to store the result.

The rounding used here is in magnitude, i.e., away from 0—if the bottom part is $<1/2$LSB of the top part then it is ignored, but if the bottom part is $\geq 1/2$LSB then the magnitude of the top part is increased by 1LSB. The reason for conditioning the round on a 1 in MQ1 is that the action both rounds and places the result in correct 2's complement form. The answer, if negative, is a 1's complement unless the low-order part is null and a 1's complement is one greater in magnitude than a 2's complement. Adding one because of a 1 in MQ1 increases the magnitude of a positive result when the lower order MSB is significant, but decreases the magnitude of a negative result when the low-order MSB is not significant. Of course, a 0 can be null in a negative number, but only if the entire low-order part is null, in which case the high-order part is already a 2's complement. The program should always round unless the low-order half is actually going to be used.

A further caution is necessary concerning division. The divide subroutine computes 29 quotient bits and leaves the remainder in MQ9-35. Following the two right shifts, MQ0 contains the true sign of the remainder, and MQ11-35 contain a truncated remainder—the two least significant remainder bits are lost in the shifts. If there was overflow in the division after normalization, MQ still contains two quotient bits, otherwise only one. In order to use the low-order part of the result, the program must either reconstruct the true remainder or save the extra quotient bits and append them at the left end of a lower order quotient calculated from the given remainder.

CHAPTER 7

MEMORY LOGIC

In a PDP-6 system, the core memories and fast memories are separate units connected to the arithmetic processor by a memory bus. A core memory may contain an 8K or 16K bank, and a fast flip-flop memory may replace the bottom 16 locations in any core bank. The internal operation of these memories, their control functions, their timing, and the way they respond to processor requests are described in a separate manual. This chapter describes only the hardware at the processor end of the memory bus: the logic elements that request access to memory, provide the necessary addresses, and control the transmission and receipt of data.

A time pulse in a main cycle or a subroutine may request access to memory by triggering appropriate operations in the memory control section of the processor. Memory control places a request signal on the bus and the processor must wait for a response from the memory addressed by the high-order address bits. Once the memory is free and available to the processor, the time required by the processor to transmit or receive data depends upon the type of memory. For a core memory, this time is usually much shorter than the memory cycle. The fast memory contains no buffer, so a read requires only slightly less time than a write. For reading from a core memory, the processor must wait until the data is available and the memory rewrites the word automatically; for writing the processor need wait only until the memory acknowledges the request, at which time the memory stores the data in its own buffer and continues with the clear and write cycle.

In addition to the standard logic for controlling access to memory, this chapter also describes the user mode registers. These govern the protection and relocation of areas in core and are included in the description of the memory address logic.

7.1 MEMORY ADDRESS LOGIC

The memory system appears to a processor as one homogeneous unit: the processor may address any one of 262,144 locations merely by providing an 18-bit address from the memory address register. The actual address put on the bus is the sum of C(MA) and C(RLR). The bus control portion of each memory decodes four or five address bits (depending upon whether a given memory contains a 16K or 8K bank), and a given memory responds only to the address wired in. If the bottom of a core bank is replaced by a fast memory, an additional selection signal is required; this signal is generated by the processor.

a MA Register

Figure 7-1 shows the 18-bit memory address register. Transfers of addresses into MA, which must always be preceded by a clear, may be made from the memory buffer, the program counter, or the console address switches. For the transfer of short addresses of accumulators, index registers, and PI channels, individual pulses may directly set bit 30 and bits 32-35. The transfer gates that produce the pulses are included in the MA control circuits (b below). The MA flip-flops are connected in a carry configuration so that a pulse at the MA+1 input to MA35 adds 1 to the contents of the register. The carry chain is broken, however, between the fourth and fifth bits from the right. A carry out of MA32 cannot go into MA31 if the processor is fetching or storing a second accumulator; such actions occur for double-length shift operations and certain arithmetic instructions that use a double-length operand or produce a double-length result. When such an instruction addresses location 17 as an accumulator, the second AC is in location 0. In all other situations that increment MA, the second address is in the normal order.

To address a location within a single memory bank, address bits 21-35 are supplied over the memory bus with ones asserted negative. Bits 26-35 are supplied directly by MA, bits 21-25 from the sum with the relocation register RLR. Bus drivers for the direct bits are shown above MA; those for the relocated bits, RLA, are with RLR in Figure 7-5. The high-order relocated bits, which select a memory, must be supplied in both states at ground assertion. Figure 7-2 shows the required bus drivers and all control connections to the memory bus. Bits 21 and 35 are supplied over the bus in both forms; that is, they are supplied both as bits to select a memory bank and to select a location within a bank. A switch at each 16K memory allows the operator to select between the bits for the two uses. If bit 35 replaces bit 21, the memory locations are interleaved; i.e. , all odd addresses are in one bank, all even in another. With an 8K bank, addresses must be interleaved because five bits are required to select the bank.

At the upper left in Figure 7-2 are the bus connections for the memory control section of the processor. At the right is a net that decodes MA bits 18-31 for all zeros, a condition which means that the address in MA is 17 or less. The decoder output generates the fast memory selection signals for the bus since a fast memory replaces the bottom 16 locations in core (fast memory addresses are not relocated). Additional fast memories require extra decoding. The signal that selects the fast memory is also conditioned by the 0 state of the readin mode flip-flop. This mode allows the operator to start a program in the area of core that is replaced by the fast memory and is ordinarily inaccessible to the program. The ungated decoder output is used by readin mode control to determine when the program leaves the readin area.

In addition to a flip-flop and three transfer gates, each MA module contains a logic net that compares the MA bit with the corresponding console address switch (these are shown below the input gates in Figure 7-1). The outputs of these nets are ANDed (Figure 7-2, left) to assert a level when the address in MA is identical to the address in the switches. This signal is used by the memory indicator logic (7.2).

b MA Control

Figure 7-3 shows the circuits that control the transfer of addresses into MA. Every transfer requires a pair of pulses: the first triggers the clear PA through the net in the upper left; the second triggers the transfer, usually through one of the PAs at the top of the figure. Two of the transfer pulses, those for MBRT and PC are applied to the gates shown with the register. Figure 7-3 shows no pulse for the transfer from the address switches because that signal is generated by the key logic and applied directly to the register gates. The other transfer PAs in Figure 7-3 set individual MA bits through the gates in the lower right. A UUO loads address 40 into MA by setting MA 30. The transfer of a PI channel address sets MA30 and loads the channel number into MA32-34, producing the address 40 + 2n. The other sets of gates load accumulator and index register addresses into MA32-35 from the appropriate IR bits.

The address transfers are as follows: A signal from the key logic clears MA prior to any transfer in from the address switches. At the beginning of every instruction cycle, IT0 clears MA and IT1, loads PC into it to retrieve the instruction, unless the processor is in a PI cycle, in which case IT1 loads the PI channel address. AT0 clears MA in preparation for the address cycle. If the instruction specifies an index register, AT2 transfers the address in from IR14-17 and AT3 clears it again. If the address is indirect, AT5 transfers in MBRT and the cycle returns to AT0 after the new address is retrieved. When the address calculation is complete, the processor continues to the fetch cycle, in which FT1 transfers IR9-12 into MA to fetch an accumulator and FT1A subsequently clears if addressed by either half of the accumulator; FT3 transfers in MBRT and the clear follows at FT4A. Finally, FT5 again transfers MBRT to fetch C(E) and the effective address is left in MA as the cycle ends.

MA transfers in the execute cycle are required only for particular instructions. UUO, POPJ, and BLT all clear MA at ET1. For UUO, ET3 sets MA30 so that the UUO subroutine will subsequently deposit the instruction code in location 40. For the other two instructions, MBRT goes to MA at ET3. This special transfer is required for POPJ because this instruction jumps to the location specified by the contents of the top location in the pushdown list (transfers to PC must be made via MA). BLT loads a new address into MA in preparation for the subroutine which subsequently moves C(E) to a new location. Three instructions, PUSH, PUSHJ, and JRST, clear MA and load it late in the execute cycle (ET9, ET10). The first two make the transfer so that the following store cycle will deposit C(E) or PC in the top location in the pushdown list; the third uses it to display the current instruction location in case the JRST is a halt.

At the beginning of the store cycle, the appropriate address for the deposit of C(E) is already in MA. If storage of an accumulator is also required, ST3 clears MA and ST5 loads it from IR9-12.

In the upper right of Figure 7-3 is a net that increments the address contained in MA. The PA is triggered by a signal from the key logic for the operations examine next and deposit next, by UUO T1 to switch from location 40 to 41, and by ST6 which is generated by the store cycle only for the deposit of MQ in a second accumulator. The two gated pulse inputs allow counting by FT1A to fetch a second AC, and by IT1 if the indexing of the pointer has overflowed in a block IOT that is using the PI system. Note that this latter event occurs only when performing the second instruction in the pair for a PI channel and that the channel address is transferred into MA at the same time. Since the channel address always ends in 0, MA+1 merely sets MA35.

c User Mode Registers

Figures 7-4 and 7-5 show the two registers that provide protection and relocation for the user mode. The bits in both registers are numbered 18-25 to correspond to the most significant eight bits in a memory address. Information is transferred into the registers by a DATAO for the processor, but the clear and set inputs have the prefix EX because the gating by the CPA selection level is included with the executive mode logic. The DATAO loads the protection register PR from bits 0 to 7 of the I/O bus, the relocation register RLR from IOB18-25.

The executive routine determines the size and location of the block assigned to a given user program by loading PR and RLR. Since both registers correspond to the most significant eight address bits, the numbers they represent are actually multiples of octal 2000 (all numbers in this discussion are octal), but identical contents do not represent the same multiple; the two registers have entirely different functions. An address in MA is compared with C(PR) and it is legal only if MA is less than or equal to PR. Since the remainder of MA may have anything from all zeros to all ones, if PR is clear, MA may contain any address from 0 to 1777, i.e., any address in which MA18-25 is clear. A clear PR represents a block of 2000 words; setting PR25 represents a block of 4000 words. The number in RLR, however, defines the first address in the assigned block as a multiple of 2000 since RLR is added to MA18-25. An address in a user program is legal if $C(MA) \leq C(PR) \times 2000 + 1777$, and each address must be relocated to the actual address to which memory access is made is $C(MA) + C(RLR) \times 2000$.

Every time a user program reloads MA, bits 18-25 of the address are compared with PR by the chain of majority gates at the top of Figure 7-4. Each gate receives a 1 signal from an MA bit, a 0 from the corresponding PR bit and a carry in from the previous gate in the chain. The carry output of the gate is asserted whenever at least two of the inputs are asserted (a block diagram of the gate is in the lower left). The carry mentioned here is really a borrow—the comparison is a subtraction of MA from PR and the outputs are labeled to indicate that a comparison is ∿ OK when the input conditions are fulfilled. At the beginning of the chain (upper right), the borrow in is disabled and there is a borrow out only if the MA bit is 1 and the PR bit is 0. At all other stages, the output is asserted whenever MA is 1 and PR is 0 or if there is a borrow from the previous stage and either MA is 1 or PR is 0. No actual difference is produced and the time required for the comparison is the time taken for level transitions across the chain each time MA changes state. A borrow out at the left end indicates that MA is larger than PR; and the two outputs of the last gate, when asserted at the same polarity, indicate whether or not MA contains a legal address. The nets in the lower right AND these two levels with ∿EX INH REL so that the signals sent to memory control (7.3) can be asserted only when the memory is being protected.

At the same time that MA is compared against PR, the contents of RLR are added to MA to relocate the address to the assigned block. Every block starts at a multiple of 2000 because the 8-bit RLR is added to MA18-25. The addition is carried out by a chain of dc adders shown above RLR in Figure 7-5. The circuit, shown in block diagram at the lower left, has three inputs: a bit of MA, the corresponding bit of RLR, and the carry from the next less significant adder. It has two sets of outputs, one a bit of the sum, the other a carry to the next more significant stage. The sum circuit uses exclusive OR nets to generate a 1 output whenever an odd number of the inputs are ones; the carry circuit uses a voltage division network to generate a 1 output whenever two or more of the inputs are ones. The RLR input is ORed with an enable level (the negation of the inhibit from executive mode control) so that if no relocation is desired, the sum output is equivalent to MA and there are no carries. This enable level is applied to pin H of all adders. The carry inputs to the first adder in the chain are disabled (upper right). Each time MA changes state, the sum of MA and RLR appears at the RLA outputs after a settling interval required for level transitions across the chain.

7.2 MEMORY DATA LOGIC

Transmission and receipt of data at the processor end of the memory bus are controlled by 36 pulse amplifiers, each with AND gates at both input and output (Figure 7-6). For a write cycle, the input AND gates are enabled by the 1 states of MB bits. To transmit data pulses over the bus, memory control applies a transfer pulse to all input gates and triggers those PAs that correspond to ones in MB. To receive information from a read, memory control generates a level that enables all output AND gates; pulses that arrive over the bus are then gated through to set the appropriate MB bits (6.1).

Figure 7-7 shows 36 MI flip-flops that drive a set of console indicators to display the contents of a memory location. The flip-flops are included on the arithmetic register modules, and the MB connections to the input gates are internal to the modules. Whenever the address for a memory access is the same as that in the console address switches, the read restart or write restart from the memory subroutine clears MI and loads MB into it. The write restart includes the condition of a write request for a nonexistent memory so that MI displays the word that would have been written. The indicators also display the contents of any location that the operator examines or deposits information into. The equal address condition suffices for examine and deposit, but an additional gate is necessary for examine next and deposit next, which use MC RST1 to trigger the transfer.

7.3 MEMORY CONTROL

Figures 7-8 and 7-9 show the logic that governs requests for memory access by the processor (flow chart, Figure 4-3). The processor may request three types of memory action: read, write, and read/write—the last being available only for operations in normal mode, which we shall consider first.

Pulses from the main cycles and a few subroutines trigger the memory subroutine for the appropriate type of memory access through the gates, at the top in Figure 7-8. The reason for the access is written along with the triggering pulse. Each gate generates an output that triggers the specific type of access required; however, any specific request also triggers a general request pulse (A5), and either type that requires a read clears MB (B5). For any instruction that requires fetch and pause, FT7 triggers the read/write request, but there are three inputs to the net that generate the restart for the subsequent store (B6). Indexing of a pointer in character operations or a block IOT is performed by a subroutine which skips the store cycle and returns directly to an earlier point in the main sequence. In these special cases, the restart is supplied by the subroutine; whereas, it is supplied by ST2 for all fetch-and-pause instructions that include the store cycle.

The request pulses supply the request levels to the bus by setting up the flip-flops in the upper left of Figure 7-9. A read request sets MC RD and clears MC WR, a write request sets MC WR and clears MC RD, and the read/write request sets both. A request for any type of cycle clears MC STOP and sets MC RQ, although the general request pulse is not applied directly to the latter flip-flop; instead it enters the two delay lines in the lower right of the figure. If the system is in executive mode, the output of the right delay triggers the PA in C7 to set MC RQ; however, if the system is in user mode, a longer delay is allowed to carry out the necessary protection and relocation operations (7.1c). Then if the user program has supplied a valid address, the request flip-flop is set; but if the address is greater than the PR maximum address, the illegal address pulse (C6) causes the processor to skip the remainder of the current main sequence by jumping directly to ST7 (5.2e) and triggers a priority interrupt on the channel assigned to the processor (8.3).

Tne bus levels that request a cycle and specify read or write are derived through bus drivers from the outputs of the three request flip-flops. (Note that the cycle request is generated from MC RQ only if a read or write is also specified. This is done to prevent a possible malfunction in processor memory control from generating a memory request which would not result in a response from memory.) MC RD also supplies the bus enable level that gates data pulses from the bus into MB (7.2). If the operator has pressed the MEMORY STOP key or if the ADDRESS STOP switch is on and MA now contalns the selected address, the main request pulse (delayed) sets the stop flip-flop (D4).

Completion of the memory subroutine is controlled by the circuits in the lower half of Figure 7-8. When the addressed memory becomes available to the processor, it returns a pulse (lower left) that acknowledges receipt of an address. This pulse triggers MC ADDR ACK, which clears MC RQ. For write access, the acknowledging pulse also sends a write restart back to memory, transfers data from MB onto the bus, and triggers the restart pulse MC RST1 from the memory subroutine to the waiting sequence (C6). For read or the read portion of read/write, the processor must wait for the read restart from memory to trigger the MC restart. In this case, the return from memory triggers a preliminary pulse so that the subroutine return is delayed. MC RS T1 clears MC RD and supplies the subroutine return. If MC STOP has been set during the subroutine, neither the read nor the write restart from memory can trigger the MC restart, and the processor stops even though RUN is not clear. The operator may then trigger the return to the waiting sequence by pressing the MEMORY CONTINUE key, which triggers MC RS T0 at KT1.

For a read/write cycle, the response by the memory in the first part is the same as for a read cycle: MC ADDR ACK clears MC RQ and the read restart generates the completion pulses, of which MC RS T1 clears MC RD. For the second part, the write request signal alone remains on the memory bus. The read/write restart pulse supplied by the processor duplicates the action of the address acknowledgement in a write cycle by triggering the write restart (Figure 7-8, lower left). Thus the processor, which already has access to the memory, both restarts the memory and provides its own subroutine completion pulses at the same time that it transfers data over the bus from MB.

The other circuits shown in the two memory control drawings handle special situations. In the lower left of Figure 7-9 is an integrating delay Type 4303 which is placed in the 1 state by every request pulse. The delay remains continuously in the 1 state as long as triggering pulses keep arriving at intervals shorter than the delay period. But should there be no request for 100 μsec, the delay returns to the 0 state, and if MC RQ is still set at this time indicating that the last request for access to memory has not been granted, the 4303 state change triggers the nonexistent-memory pulse (the pulse is inhibited if there is a memory stop). This pulse sets an error flip-flop in the processor I/O interface to trigger a priority interrupt on the assumption that if the addressed memory does not respond within 100 μsec, there is no memory with that address connected to the bus. If the console MEMORY DISABLE switch is off, the error pulse generates a restart that clears MC RQ by simulating the address acknowledgement from memory. For write, the simulated MC ADDR ACK also triggers the subroutine completion pulse in the normal manner; for any other cycle (MC RD(1)), an additional pulse is generated to supply the MC restart.

The remaining logic is associated with a pair of synchronizing flip-flops (Figure 7-9, right) that compensate for memory stops between the two parts of a read/write cycle. If the operator has pressed the MEMORY STOP key or the ADDRESS STOP switch is on (which would allow a stop on the equal address condition), AT4 sets the split/cycle synchronizing flip-flop. There is no need to set it earlier in the main sequence because all instruction and address memory requests are for read only: a fetch-and-pause request can be made only in the fetch cycle. The 1 state of the sync flip-flop prevents FT7 from triggering the read/write request pulse and prevents the read/write restart pulse from triggering the write restart for memory. Instead, separate read and write requests are triggered by FT7 and the read/write restart (Figure 7-8, left center) to prevent the processor from memory in the event it stops. There is an extra switch input to the sync flip-flop set gate that allows the operator to override the split cycle for maintenance purposes. In case the processor might stop while hanging onto memory, the read/write request pulse sets another flip-flop, the stop sync. If the operator should restart the processor with some operation other than memory continue, KT1 triggers the write restart to memory so that it may finish its cycle, writing the same word back into the addressed location.

CHAPTER 8

INPUT/OUTPUT

The PDP-6 input-output system includes the peripheral equipment and three sections of the arithmetic processor. The processor elements are in-out transfer control including the I/O bus, the priority interrupt system, and an I/O interface for the processor that allows IOT instructions to control the processor itself as a device. In addition to the above processor equipment, this chapter describes the control units for four of the more common in-out devices: photoelectric paper tape reader, paper tape punch, Teletype keyboard-printer, and card reader. Other equipment may be added to the system merely by connecting the associated control units to the I/O bus.

8.1 IN-OUT TRANSFER CONTROL

Figure 8-1 shows the logic that decodes the IOT instructions and times the transfer of information into and out of the computer via the I/O bus. The 36 cable drivers for the bus data lines are shown in Figure 8-2. The I/O device selection lines come from bits 3-9 of the instruction register and the bus drivers for them are shown on the IR drawing, Figure 5-7. Every device control unit contains a diode net which receives input from 7 of the 14 IOS lines, one from each bit in IR3-9. The configuration of these input connections determines the selection code for a particular device, which responds to IOT commands only when the appropriate number appears in the device code portion of an instruction word.

The cable drivers for the I/O bus are transceivers, i.e., they handle signal transmission both into and out of the computer. IOT control governs output from AR onto the bus by supplying transfer and reset levels to the drivers; any signal placed on the external bus by an input device is automatically made available through the drivers to the AR input gating, with no transfer signal required other than command levels sent to the device, followed by the driver reset. As an example, consider the driver for bit 0 shown in the upper left of Figure 8-2. A 1 in AR0 is represented by a negative level at input E (the input resistors are mounted on the AR flip-flop modules). The transfer and reset levels from IOT control are connected, respectively, to L and M. In an output sequence, L goes negative for 2.5 μsec, transferring the input at E to outputs J, F, and H: the assertion level of the first and third is negative; the second is ground. J merely drives an indicator on the console in-out panel. F drives the external bus and is connected to the capacitor-diode inputs of the control registers and output buffers in the various devices. F and H together provide bus signals at both assertion polarities for use in I/O operations within the computer—in the priority interrupt system and in the I/O control of the processor. At the same time that the transfer level returns to ground, the reset signal brings input M to ground, placing a hard negative on the bus (F) for 2 μsec. This action discharges the capacitor-diode gates connected to the bus to ensure that there will be no conflict with a subsequent I/O operation. For input, since the outputs of the device control/status registers and input buffers are connected (through diode gates with no dc load) to the same bus lines, an addressed device merely places ones at ground assertion on the bus. With the control signals at both L and M off, a ground at F produces a negative level at H, which is connected to the AR input gating. The gate for incoming status or data is applied to the device control for 2.5 μsec and is followed by the 2-μsec reset to the bus drivers.

The levels and pulses that control the bus and devices are generated by the logic shown in Figure 8-1 (flow chart, Figure 4-13). When IR0-3 contains the code for the IOT class, the net shown at the lower left decodes IR10-12 to determine the specific instruction. The four OR gates at the right develop levels that control functions common to more than one instruction. The IOT time chain always begins at ET4 (upper left). For a block instruction, the chain starts at IOT T0, which restarts the second part of a read/write cycle to write the indexed pointer word back into memory (7.3). The return from the memory subroutine generates IOT T0A which clears MA, sets IR12 to convert the block instruction into its associated data instruction, and triggers the fetch cycle for the data. It also carries out the required skip and overflow operations depending upon whether the block transfer is using the PI system and whether the block is complete.

In any nonblock IOT, including a data instruction within a block, ET4 sets IOT GO. The 0-to-1 transition of this flip-flop triggers the initial setup delay provided the system is not still within the reset period following a previous IOT. If two IOTs occur too close together, the second waits until the fall in the first reset level itself triggers the delay. The terminating pulse from the initial setup delay, IOT T2, triggers a pair of delays: a restart delay whose terminating pulse is IOT T3, and a final setup delay whose terminating pulse is IOT T3A.

The level outputs of the initial and final setup delays are ORed to produce a 2.5-μsec gate that places information on the bus (the pulse IOT T2 keeps the gate well grounded during the initial to final setup transition). During this period, an output instruction connects AR to the bus through the I/O cable drivers; an input instruction connects the data buffer or status register to the bus in the device control. For the two output instructions, IOT pulses T2 and T3 trigger the appropriate clear and set pulses for the device. The initial setup delay allows the device selection from IR to enable the capacitor-diode gates in the device enough ahead of the clear; the restart delay allows enough time between clear and set. For any input instruction AR is cleared prior to the IOT chain by ET0; the transfer in from the bus is accomplished by IOT T3. This pulse also causes the computer to reenter the execute cycle at ET5. When all transfers are complete, IOT T3A triggers the reset delay whose level output prevents the initiation of IOT operations for 2 μsec and resets the bus.

In addition to the timing logic, Figure 8-1 also shows a pair of general control signals (upper right). Whenever computer power is turned on, -15 volts is applied to pin C of I/O cable 3, turning on all peripheral equipment. The reset signal at pin B clears the device control units on the power clear and when the operator triggers the reset from the console (both conditions are included in MR START, see 5.1). The program can also generate this reset by means of a 1 in bit 19 of the same CONO that controls the flags in the processor I/O interface.

8.2 PRIORITY INTERRUPT

The priority interrupt system allows the program to be interrupted by a signal on any one of seven channels arranged in a priority chain. The priorities assigned to the I/O devices and to the processor are completely under program control. Moreover, the program may turn any channel on or off and may request a break on any channel. The processor checks for requests at the beginning of every instruction and address cycle, and honors immediately the highest priority request that has been made. Any requests made after the processor has completed the effective address calculation must wait until the end of the current instruction except in a block transfer. The processor honors a request by entering a PI cycle and executing the instruction in location 40 + 2n where n is the channel number. No further interruptions can occur while the computer is in a PI cycle. If the break instruction is a block IOT and the block is not complete, the processor leaves the PI cycle and dismisses the break automatically.

If the break instruction is not a block IOT or the processor goes on to the second instruction in the break pair, the PI cycle terminates at the end of the non-IOT instruction, which must be a JSR to a break routine. This routine may then be interrupted at any time by a higher priority request, and the program must dismiss the channel with a JRST at the end of the routine.

The processor I/O interface and all device control units each contain a 3-bit PI assignment register: the program assigns a priority to a device by loading a channel number into its assignment register (0 means no selection). The outputs of the register are applied to a gated binary-to-octal decoder whose outputs 1 to 7 are connected to the seven PI request lines. When a device requires service, it requests a break by gating on the decoder, grounding the line corresponding to the number in the register.

a Priority Chain

The 21 flip-flops in the priority chain are shown in Figure 8-3. Each channel is controlled by a column of three flip-flops, PIOi, PIRi, and PIHi, where i is the channel number. The setting of each flip-flop represents a stage in the interruption process for a given channel in the sequence: channel on, request made, break held. Between the rows of PIO and PIR flip-flops are the logic gates through which breaks are requested by a device or the program; between the PIR and PIH rows is the priority-determining chain of level gates which is activated by the input PI ACTIVE at the left. All other signals at the left control state changes in the flip-flops by rows (for the generation of these pulses, see b below). For program control over the system, the individual channels 1 to 7 are selected by ones in bits 29-35 (IOB29-35) of CONO.

Since the control sequence for all channels is identical, let us consider channel 4. The program selects this channel by a 1 in CONO bit 32 and turns the channel on or off by setting or clearing PIO4. To request a break on channel 4, a device must ground the PI4 line, pin V of IOC3. At specific times during its operations, the processor strobes the request lines by generating PIR STB. This signal sets PIR4 if the channel is on and a device has requested a break on it. The program, however, may bypass the PIO flip-flops by selecting a channel and setting the associated PIR flip-flop directly (PIR ← IOB (1)).

Several PIR flip-flops may be set simultaneously, but the PI system honors only the highest priority request (i.e., for the lowest numbered PIR flip-flop that is set), by asserting one of the PI REQ outputs shown at the top of the figure. The selection of a request output is made by the chain of level gates between the PIR and PIH flip-flops. The 1 state of PI ACTIVE enables the highest priority stage of the chain at the left. If a break is neither requested nor held on channel 1, the first stage enables the second. Similarly, if stage 2 is enabled and an interrupt is neither requested nor held on channel 2, stage 3 is enabled, and so on to each successive stage. For example, a request is honored on channel 4 in the following manner: If the request and hold flip-flops for the first three channels are all 0, stage 4 is enabled at 2N11R. If there is currently no break held on channel 4, PIH4 is 0, satisfying AND gate 2N11R,S and producing a ground at 2N12S and 2N13R. The 1 state of PIR4 satisfies the upper AND gate asserting the request output PI REQ4, but negates input S of the lower AND gate disabling the rest of the chain to the right. After the break has been started, PI control generates PIH ←PI CH RQ, which sets PIH4. The 1 state of this flip-flop negates input 2N11S, disabling both PI REQ4 and the remainder of the chain. Furthermore, it also holds PIR4 in the 0 state, preventing any further request on the same channel while the break is in progress. These actions, however, have no effect to the left, so a request on a higher priority channel can interrupt the current break.

To dismiss a break when returning to the interrupted program, PI control generates PIH(0) ← PI OK(1), which clears the PIH flip-flop for the channel on which a break is currently being held. It does this by clearing all PIH flip-flops from the left, up to and including the first one that is 1. The enabling levels for the clear gates in the stages to the left require that both the PIH and PIR flip-flops be 0, but this condition is bound to be satisfied because setting any higher priority PIR flip-flop would have caused an interruption of the break in progress. Thus, the system always dismisses the current break which is of highest priority and returns to a lower priority break, or if there is none, to the main program.

b PI Control

Figure 8-4 shows the circuits through which the program controls the priority interrupt system, and which governs the sequence of operations in the priority chain when a device requests a break. When an IOT with device code 004 appears in the program, the AND gate in B6 enables the selection level for the PI system. In CONI, the status level from IOT control gates PI ACTIVE and the PIO flip-flops onto the I/O bus (the bus gates are shown in Figure 8-5, upper right). CONO bit 23 gates the clear pulse to clear the entire system by generating PI RESET (B4). This pulse clears PI ACTIVE, clears the PIO flip-flops directly, and generates other pulses which clear the PIR and PIH flip-flops (A1, 3). The following set pulse in CONO triggers PI CONO SET, which performs the various operations specified by bits 24-28. Bit 28 turns on the PI system by setting PI ACTIVE (C7); bit 27 turns it off. The other three control bits perform operations on channels selected by ones in bits 29-35: bits 25 and 26 turn the selected channels on and off (A8); bit 24 requests a break on the selected channels (A3).

The remaining logic in Figure 8-4 controls operations associated with the requesting, execution, and dismissal of a break (most events are listed in the flow charts for the instruction and execute cycles, Figures 4-4 and 4-5). When any device grounds a request line, the inverted level from the PIR input gating lights an indicator located at 2L20 (right of the figure). The initial time pulses in both the instruction and address cycles check for an interrupt request by generating PI SYNC (B3), which triggers PIR STB (B5), provided the processor is not in a PI cycle. The PIR strobe sets the PIR flip-flop in any on channel over which a device has requested a break. As soon as one or more PIR flip-flops are set, the priority chain asserts one and only one PI REQ level. The nets in the lower left of Figure 8-4 generate a level PI RQ if any request is honored, and encode the channel number into binary. PI RQ allows a delayed PI SYNC (5.2a) to generate IATO (A5) which places the processor in a PI cycle by setting PI CYC (C6). The processor then returns to the instruction cycle, and using the binary-encoded channel number, retrieves from memory the instruction contained in location 40 + 2n (see 7.1b).

Since the most common interrupt for an I/O device is one that merely executes a single block IOT that takes the place of a whole subroutine, let us consider this type first. If the indexing of the pointer word overflows, IOT T0A sets PI OV when the processor switches from the block to the data instruction. If the block is not complete, both a hold and a restore level are generated (upper left). In the execute cycle, ET0 holds the break by setting PIH, only to have ET1 immediately dismiss it; then ET10 ends the PI cycle by clearing PI CYC. If the block is complete (PI OV(1)), the break is not held, and after completing the data instruction, the processor performs the instruction in location 41 + 2n (which must be a JSR to the break routine). The sequence of events is now the same as it would have been had the first instruction not been an IOT. The break is held at ET0; ET10 dismisses the PI cycle and clears PI OV. The routine must then terminate with the appropriate restore instruction to dismiss the break (B2).

When the program performs a block transfer, PIR STB is generated at BLT T4 during the processing of every word in the block. If a request is discovered, there is no IATO; the processor instead terminates the incomplete block as though it were finished (6.6a), and then returns to the instruction cycle to finish the remainder as a new BLT with initial addresses one greater than those last used. However, since a request is waiting, the instruction cycle is interrupted and the processor does not restart the block until all requests have been honored.

8.3 PROCESSOR I/O CONTROL

The interface between the arithmetic processor and its own IOT control section is primarily a system of flags and enable flip-flops that allows the processor to check its own status and permits processor actions to request sequence breaks through the priority interrupt system. Certain data transfers may also be triggered through the interface. An IOT instruction selects the processor in the same manner as any other device: when the device code 000 appears in IR3-9, the diode net at the left in Figure 8-5 generates the level CPA which gates the various command signals from IOT control. For DATAI, CPA gates the contents of the console data switches into AR (this transfer does not use the I/O bus and the gate is shown with the key logic, Figure 5-1); a processor DATAO loads the memory protection and relocation registers from the bus (7.1c). The condition commands, gated by CPA, control and sense the flip-flops shown in Figure 8-5 plus the user mode flag (5.5) and the PC change and overflow flags in the AR logic (6.2e). CPA CONO SET regulates the flip-flops according to information on the bus; CPA STATUS gates the flip-flops onto the bus. Extra bus lines are used for conditions out so that the set pulse may set or clear individual bits; some flip-flops are set only by external conditions and CPA CONO SET can only clear them. A complete list of the bus lines and the CONO actions and status bits associated with them is included in the IOT flowchart (Figure 4-13). In addition to the enable flip-flops and flags, CPA includes a PI assignment register PIA which is loaded from IOB33-35. Various flag conditions gate the PIA decoder to request an interrupt on the PI line addressed by the register.

The first two flip-flops at the left indicate errors: the left one is set when a user program attempts to use a protected area of memory; the right one, whenever the processor requests access to memory but specifies an address which fails to elicit a response within 100 μsec (7.3). Either error flip-flop triggers an interrupt when set. The next pair of flip-flops provide a means of synchronizing computer operations to real time. A filament transformer in the power control drives a pulse generator that sets the clock flag 60 times per second. If the enable flip-flop is on, the flag causes a break every time it is set. The other two enable flip-flops allow the PC change or overflow flag in AR control to request an interrupt. There is also an overflow flag for the pushdown list: if there is a carry out of AR0 in a pushdown or pullout instruction, ET10 sets PDL OV causing a break. Both types of instructions are represented by the level that gates a switch of MB and AR at ET10. AR CRY0(1) represents overflow for pushdown, underflow for pullout.

8.4 I/O INTERFACE LOGIC

Every peripheral device attached to PDP-6 must have a control unit that acts as an interface between the device and the IOT control section of the arithmetic processor. A control unit is connected to the processor via the in-out bus, but an automatic control unit for a high-speed device may also be connected to the memory bus for direct transfers of data to and from memory. The basic control unit includes a data buffer and a control section. The data buffer contains a number of bits appropriate to the device (up to a maximum of 36) and is connected to both the device and the I/O bus data lines. The control section handles receipt and transmission of control signals for both the processor and the device.

The data and control connections from a control unit to its device depend upon the organization of the device and are described with the appropriate equipment. All control units are connected to the processor via the in-out bus, whose configuration is shown in Figure 7-10, and which includes the following lines:

1 Power On Line - This line is at -15 volts whenever computer power is on. It is connected to the remote terminal of the local/remote switch in the power control for each device and normally turns on all peripheral equipment when the computer is turned on.

1 Reset Line - This line supplies negative pulses to clear the control registers and data buffers of all equipment attached to the bus. The line is pulsed when computer power goes on or when the operator presses the I/O reset key on the processor console; the program may also generate a reset.

36 Data Lines, IOB0-35 - These lines transfer all data and control information, with ones asserted at ground. For output, the lines connect AR to the device control registers and data buffers through capacitor-diode gates; for input, the status registers and data buffers are connected to AR through Type 4657 Diode-Gate Bus Drivers (connection must be made by a 4657 or equivalent circuit that places no dc load on the bus).

7 PI Request Lines, PI1-7 - A device requests an interrupt on one of seven priority channels by placing a ground on the appropriate line.

14 In-Out Selection Lines, IOS3-9 - These seven pairs of lines are derived from the outputs of IR3-9, which contains the device code in an IOT instruction. Assertion is at ground for both ones and zeros. A device control is connected to only seven of the lines, one from each pair. The configuration of these connections determines the selection code for the device, which then responds to only those IOT commands that are accompanied by the appropriate code.

6 Command Lines - These lines provide two pairs of negative pulses for output, two negative levels for input. Both pulse pairs include a clear and a set: one pair loads conditions from the bus into the control register, the other loads the data buffer. The two input levels gate status and data onto the bus.

Every control unit includes control and status registers, although these registers usually overlap and may even be identical. That is, all control flip-flops can usually be sensed by the program as status, and all status flip-flops within the control unit can usually be governed by CONO. Some devices have additional status signals which can be sensed by the status checking instructions but cannot, in general, be affected by the program directly.

The control/status registers for all devices include the same basic elements. As an example, consider the control/status register for the paper tape reader, Figure 8-6. The basic elements are a 3-bit priority interrupt assignment register PIA and two control flip-flops, BUSY and FLAG. These elements have the same names in most control units but are distinguished by an appropriate equipment prefix. The tape reader has an additional control flip-flop that governs the data mode, i.e., whether data is to be read from the tape in binary or alphanumeric. The punch register contains the same bits; whereas the Teletype has two busy flip-flops and two flags because it is actually two devices—one input, the other output. Even in large scale in-out systems such as magnetic tape, the control connections to the processor are the same as in the tape reader, and the control and status registers differ only in complexity, not in kind. Every device has a PIA register, but it may have a command register associated with the busy flip-flop, several flags, and a number of status bits for error and other conditions. CONO can provide a maximum of 18 control bits; control information on a larger scale such as full addresses and word counts must be supplied by a DATAO. A maximum of 36 bits can be checked by a single status instruction; if more are necessary, additional device codes must be used.

Selection of a device and the timing of all transfers of data, initial conditions, and status between the processor and the peripheral control units are described in detail in 8.1 (which the reader is strongly advised to study before investigating any of the peripheral equipment). At the left in Figure 8-6 are the connections to the processor for PI requests, IOT commands, and device selection. Every device is connected to the six command lines that govern the transfer of conditions out, data in and out, and status in. The pulses provide output clear and transfer functions while IOT control gates AR onto the bus; the levels gate input onto the bus while IOT control pulses the information into AR at the processor. All devices require the condition and status commands; however, only a bidirectional device uses both types of data commands.

Although all devices are connected to the six command lines, to have any effect on a control unit, the commands must be gated in by a selection level that is generated by the net in the lower left of the figure. Every control contains such a diode net, wired to the seven IOS lines that define the device selection code. The control unit then responds to IOT commands only when the appropriate code appears in the instruction. Note that the reset bypasses the selection gate in triggering IC CLR and clears the control registers for all devices.

In the normal sequence of operation for any device, the program first provides the initial conditions by means of a CONO, whose pair of command pulses generates the IC pulses within the control unit. The first pulse usually clears all the control and status flip-flops; the second pulse loads the assigned channel number into PIA and sets up the other control bits as necessary. The control information is supplied over the in-out bus by the effective address calculated for the CONO: bits 33-35 always provide the priority assignment; the configuration for other bits depends upon the device and is listed in the flow chart of the input-output operations (Figure 4-14).

The 1 states of the status bits are separately ANDed with the status level, which is generated by any of the three conditions-in instructions. The AND gate outputs are applied, in general, to the same bus lines over which the equivalent initial conditions are sent to the control register. The outputs of individual bits in the right of the figure provide control levels for the various operations within the control unit. The outputs of the three PIA bits, however, are applied to a gated binary-to-octal decoder. Outputs 1 to 7 of this decoder are connected to the request lines of the priority interrupt system. A group level at pin P enables the decoder, placing a ground on the output corresponding to the number contained in PIA, i.e., requesting a break on the channel assigned to the device. The 0 decoder output is not connected, so it corresponds to no PI assignment.

In addition to governing information transfers, the data commands also perform certain control functions so that an entire block of data can be processed after giving the initial conditions only once. CONO must assign a PI number and provide the necessary data mode information, but it need not necessarily set BUSY. A common procedure for output is that each DATAO not only loads the buffer but also sets BUSY causing the device to operate. When a control unit has completed the transfer of data from buffer to device, its completion signal clears BUSY and sets FLAG. The latter action enables the PIA decoder requesting an interrupt. The processor then responds with another DATAO which provides new data, clears FLAG, and again sets BUSY. Following the last word in the block, the program must provide an extra CONO to clear FLAG. For input, the initial conditions must include setting BUSY to make the control unit retrieve information from the device; the completion of the transfer from device to buffer then clears BUSY and sets FLAG for an interrupt. Then the DATAI that gates the buffer onto the bus also clears FLAG and sets BUSY causing the control unit to retrieve more data. The turnoff of the DATAI level clears the buffer in preparation for the next word. To end a block, the program must follow the last DATAI with an immediate CONO to clear BUSY, preventing the retrieval of an extra word.

8.5 STANDARD IN-OUT EQUIPMENT

Small scale in-out devices most commonly used with PDP-6 are paper tape reader and punch, Teletype keyboard-printer, and card reader. This section describes the control units for these four devices; control units for other peripheral equipment are described in separate publications. The remaining Chapter 8 logic drawings are in four groups of two or three each for the standard devices. In each group, the first drawing shows the control/status register and control connections to the processor via the I/O bus as described in 8.4. The other drawing or drawings in each group show the control connections to the device, and the data buffer with its connections to both the device and the I/O bus.

a Paper Tape Reader

In addition to PIA, BUSY, and FLAG, the control register for the tape reader (Figure 8-6) contains a data mode flip-flop B. There is also an extra status bit which indicates that the reader motor is on. Since the reader is an input device, for initial conditions CONO must provide a PI assignment, clear FLAG, set BUSY to cause the device to retrieve the first data from tape, and set up B according to the desired mode. Alphanumeric mode (B=0) reads a single 8-bit character from tape and makes it available over bits 28-35 of the bus with hole 1 in IOB35. Binary mode (B=1) reads holes 1-6 of only those characters in which hole 8 is punched and assembles six such characters into a 36-bit word. The first character encountered is made available over bits 0-5 of the bus with hole 1 on IOB5.

Figures 8-7 and 8-8 show the 36-bit buffer, which is made up of six Type 4221 6-Bit Shift Registers. A single shift register module contains one buffer bit from 0 to 5 and every sixth bit after it; output from a single hole position is always read into a single module. For example, hole 1 is associated with that part of the buffer containing bits 5, 11, 17, 23, 29, and 35 (upper right, Figure 8-7). The strobe generated by the feed hole reads hole 1 into bit 35 (the presence of a hole is indicated by a negative level), hole 2 into bit 34, etc. In alphanumeric mode, B(0) causes the strobe to load holes 8 and 7 directly into bits 28 and 29 at the same time that the other six holes are loaded into the least significant bits of the 4221 Registers. Thus in reading a single 8-hole character, the 4221 Registers are not used as shift registers at all. But in binary mode, the first strobe loads the six data holes into the least significant bits of the register, and each subsequent strobe shifts the previously read character one place to the left in parallel at the same time that it reads in a new one. The characters are counted in an extra 4221 Shift Register in the upper left of Figure 8-7. Every strobe sets the right bit SR6 and also shifts left those ones that have already been strobed in. The sixth strobe sets SR36 indicating that the entire 36-bit word has been assembled.

Retrieval of information from tape is controlled by the logic shown in the lower left of Figure 8-8. When the operator turns on the reader from the console, the level MOTOR ON generates a pulse that clears BUSY to ensure that the reader does not go into operation inadvertently. This start clear pulse also sets FLAG to cause an interrupt when the motor goes on, and the same action is also produced through a pulse generator in Figure 8-6D6 when the motor goes off. When a CONO for the reader appears in the program, the first command pulse clears the control register. The second command pulse must then set BUSY, whose 0-to-1 transition generates PTR CLR (Figure 8-8C4) to clear the buffer and the character counter SR. BUSY(1) also engages the clutch and releases the brake (lower left), placing the tape in motion. (The operator may duplicate this action by means of the console tape feed switch but no reading occurs.) The feed hole signal is connected to two pulse generators—one through an inverter—so that a leading edge pulse is generated on the transition-to-ground when the feed hole is encountered and the trailing edge generates a second pulse on the negative return. If BUSY is 1, the leading pulse triggers a 400-μsec delay whose termination generates the strobe that loads the buffer. This is timed to occur when the holes are centered with respect to the photocells. In alphanumeric mode, every feed hole strobes all eight data holes into buffer bits 28-35. The trailing edge pulse then sets FLAG and clears BUSY: the former enables the PIA decoder, the latter releases the clutch and engages the brake, stopping the tape. (The trailing edge pulse is gated by BUSY(1) to prevent the tape feed from setting FLAG.) When the processor responds, DATAI gates the buffer onto the bus and clears FLAG. The termination of DATAI sets BUSY to restart the reader and the flip-flop transition again clears the buffer and SR by generating PTR CLR.

In binary mode, every feed hole triggers the delay but the terminating pulse generates a strobe only if hole 8 is punched. The strobe loads holes 1-6 into the least significant bits of the shift registers and sets SR6. Six strobes fill the buffer and shift a 1 into SR36, which allows the trailing edge pulse to set FLAG and clear BUSY. Retrieval of the word by the processor and restart of the reader are the same as for alphanumeric mode.

b Paper Tape Punch

The punch has the same control bits as the reader: a 3-bit PIA register, BUSY, FLAG, and a data mode flip-flop B (Figure 8-9). B determines whether punching is in alphanumeric or binary mode; unlike the reader, however, either mode punches just one 8-bit character per DATAO and the only difference between modes is in the character format. Alphanumeric mode (B=0) punches the character supplied by bits 28-35 of the bus, with IOB35 controlling hole 1. Binary mode (B=1) always punches hole 8, skips hole 7, and punches holes 6-1 according to the characters supplied by IOB30-35. Also unlike the reader, the program rather than the operator turns on the punch motor and there is a wait of 1 second after turnon to allow the motor to reach speed. To reduce wear, the motor goes off whenever the program does not call the punch for 5 seconds. Because the program must turn on the motor, the initial conditions may vary. However, in normal circumstances CONO supplies a PI assignment, sets up B according to the desired mode, and may set FLAG if the programmer wishes to handle an entire block including the first character through the PI system. Then the clear for every DATAO clears FLAG and sets BUSY to trigger the punch cycle for the character supplied by the instruction; when the cycle is complete, a done pulse clears BUSY and sets FLAG to request a priority interrupt by gating on the PIA decoder.

Figure 8-10 shows the data buffer with associated gates, punch solenoid drivers, and timing circuits that govern the motor and punching cycle. Whenever the program sets BUSY or the operator feeds tape from the console (either of which generates GO), the 4303 Integrating Delay in D6 is set and remains set as long as it is triggered at intervals shorter than the delay period. The 1 state of this delay turns on the punch motor by enabling the Type 823 SCR Driver (A1) and holds the motor on for 5 seconds after the final GO. The gate to the driver is inhibited, however, by the power clear enable from the key logic so that the punch cannot go on when computer power is turned on. The delay 1 state is also ANDed with the off level from a second delay to generate SPEED, which is in turn ANDed with BUSY(1) to generate READY. The second delay is triggered by the turnon of the first, so the assertion of the two signals by GO is delayed 1 second if the motor is off. Both signals indicate that the motor is up to speed and punching may proceed, but the busy condition in the ready signal prevents the tape feed from setting the flag and causing an interrupt.

The circuits in the upper left synchronize punching to the period of the motor and drive mechanism. A reluctance pickup provides a sync mark through a pulse generator whose output, gated by READY, triggers a 5-msec delay. The delay level output, ANDed with BUSY(1), gates the data buffer 1 outputs to enable the drivers for the punch solenoids. At the end of the punch interval, the solenoids release and the delay terminating pulse, DONE, clears BUSY and sets FLAG. Note that the input from the tape feed key operates the feed hole solenoid driver directly, provided the motor is up to speed. This allows the operator to generate tape leader: the tape feed key enables the speed level after the appropriate delay but does not generate the ready level, so it punches feed holes without punching the contents of the buffer.

The data mode flip-flop controls the character format through the gates in the lower left. In binary, B(1) provides a hard ground through the 4113 Diode Gates to hold PTP8 set and PTP7 clear regardless of the action of the DATAO clear and set pulses on the remainder of the buffer. In alphanumeric, the 4113 outputs float, and punching of holes 7 and 8 is determined solely by the contents of the corresponding buffer bits.

c Keyboard-Printer

The Teletype is actually two separate and distinct devices with a common PI channel and device code. Signal names in the logic drawings use three prefixes: TTY for elements common to both devices, TTI and TTO for signals unique to the keyboard and printer, respectively. In addition to PIA, the control register (Figure 8-11) includes two pairs of control flip-flops, TTO BUSY, TTO FLAG, TTI BUSY, and TTI FLAG. In order to allow the program to control the devices separately, the first pulse in the CONO pair clears only PIA; both clearing and setting of the other control bits are handled directly by CONO bits (of course the IOB reset does clear the control bits in addition to PIA). In regular operation, the control bits are handled by the data instructions and signals derived from flag state changes in the transmitter and receiver. For output, the DATAO clear sets TTO BUSY and clears TTO FLAG; the subsequent setting of TTO DONE in the transmitter clears TTO BUSY and sets TTO FLAG. For input, TTI BUSY is set by the 0-to-1 transition of TTI ACTIVE when the operator strikes a key. When the entire character has been received, TTI DONE is set, clearing TTI BUSY and setting TTI FLAG. The response by the program with a DATAI then clears TTI FLAG. Setting either flag gates on the PIA decoder to request an interrupt. During a prolonged idle interval, PIA should be left clear to prevent interrupts due to inadvertent keyboard manipulation. When setting up PIA following an idle period, the CONO should also clear TTI FLAG to prevent readin of a character that may have been typed accidently.

Data transmission between the processor and the Teletype control unit is in 8-bit characters over bus lines 28-35. IOB35 corresponds to the first character bit and the eighth bit (IOB28) is always 1. Between the control unit and the keyboard-printer, data transmission is in the form of 11-unit characters which are presented serially at 110 bits per second, so one complete character requires exactly 100 msec. Character transmission always begins with a start impulse (space), followed by the eight data bits in order—with ones represented by marks—and transmission is terminated by a stop impulse (2 marks). An idle line marks continuously.

At the right in Figure 8-12 are a crystal oscillator and a countdown chain that generate a pair of clocks to regulate the transmission and receipt of data. The higher speed clock for input provides greater time resolution: this allows sampling the keyboard data near the center of each bit to prevent ambiguity during level changes. Almost all of the remaining logic in the figure is included in two modules, a 4706 Teletype Receiver and a 4707 Teletype Transmitter. Each includes an 8-bit data buffer with associated control circuits and flags. Both buffers are shift registers (the output buffer is actually included within a 10-bit shift register).

TTO (Figure 8-12, upper half)

The transmitter module receives no initial reset, so the program or the operator should always send a rubout character after computer power turnon. After every transmission, the data buffer is left clear in readiness for the next character. When a DATAO triggers a transmission cycle by setting BUSY, it also clears DONE and then loads the character into the buffer and sets ENABLE. The stop timer is a 4-counter that holds at 4 until enabled by a 1 count applied to a terminal separate from the clock input. The 4 count ANDed with ENABLE(1) conditions the set gate for ACTIVE so the next clock sets it. ACTIVE(1) enables ÷2, and the transition clears OUT LINE, placing a start impulse (0) on the printer line. Clocks 2 and 3 then set and clear ÷2, whose clear transition generates the first shift pulse. Shift 1 clears ENABLE, shifts the 1 that was in ENABLE to TTO8, and moves the entire character left one unit, placing the first character bit in OUT LINE and out to the printer. The seven succeeding shifts, at clocks 5, 7, ..., 15, 17, move the rest of the character out to the printer one bit at a time; they also fill the register with a 1 followed by zeros immediately after the eighth data bit. When TTO1-8 contain 10 000 000, the AND gate in A6 is satisfied, enabling the ACTIVE clear gate. The shift at clock 19 then clears ACTIVE and shifts left again, clearing the register and setting OUT LINE to feed a stop impulse to the printer. ACTIVE(0) disables ÷2 to inhibit further shifts, enables the stop timer by loading in a 1 count, and sets DONE, which clears BUSY and sets FLAG. It also holds OUT LINE on to keep the line marking.

The next character cycle cannot begin until the ACTIVE set gate is enabled again. Clock 22 asserts the 4 count: if the new DATAO has already set ENABLE or does so before the next clock, ACTIVE is set at clock 23 and printing continues at maximum rate; if not, ACTIVE(1) does not occur until the clock following DATA SET. Thus the stop is held on the line for four clock periods or two character units (minimum) as required by the printer; and for maximum printing rate, DATAO should appear within this interval. The left and right sections of the output timing chart show, respectively, the flip-flop conditions for relatively late and normal DATAO arrival within the maximum rate interval. Levels shown at the left also apply to first-character or slow-speed printing except that the 4 count would be up already.

TTI (Figure 8-12, lower half)

At the beginning of an input cycle, the data buffer contains the previous character; other conditions are as shown at the left end of the input timing chart. The negative start impulse from the keyboard distributor asserts SPACE, which enables the ACTIVE set input through the AND gate in D8. ACTIVE(1), produced by clock 1, enables ÷ 8; the transition clears ÷ 8, clears LAST UNIT, sets BUSY, and generates SET, which sets all flip-flops in the buffer. LAST UNIT(0) enables the shift one-shot, so each 4 count asserted by ÷8 generates a shift pulse. The first shift, at clock 5, clears DONE and loads the start impulse into TTI8 as a 0; successive shifts read in the character bits and move the 0 left. When TTI1 contains the 0, the last shift fills the buffer and sets DONE and LAST UNIT. The DONE transition sets FLAG and clears BUSY; when the program responds, the DATAI gates the buffer onto the bus and clears both FLAG and DONE. LAST UNIT(1) inhibits the one-shot but enables the lower ACTIVE clear gate; no more shifts are generated within the cycle but the next 4 count clears ACTIVE, inhibiting ÷8 and enabling the upper input to the AND gate for the ACTIVE set input. The next character cycle begins when a start impulse satisfies the lower input to this gate.

The AND gate in D7 connected to the upper ACTIVE clear input prevents activation of the unit by momentary noise pulses at the data input. The start impulse must prevail through clock 5; otherwise ∾TTI SPACE satisfies the gate (the buffer was set at clock 1) so the first shift clears ACTIVE to reset the device.

d Card Reader

In addition to PIA and FLAG, the control register for the card reader (Figure 8-13) contains START instead of the usual BUSY, a data mode flip-flop B, an error flip-flop DATA MISSED, and an auxiliary flag, FLAG ALL. All of these but START are available for status checking, and the reader provides four additional status signals. Since START is cleared as soon as the reader starts a card cycle, the busy status is the OR of START(1) and CARD CYCLE, which is derived from the CCL signal from the reader. The negation of this signal, CARD DONE, is also available separately as a status bit (27) as are the other four reader status signals, FEED CHECK, CRL, CREL, and EOF (bits 23-26). CRL indicates that the card reader is not ready for operation, CREL indicates a validity check or read check error in the reader (for a full explanation of reader conditions, refer to card reader operation, 3.2d). The EOF signal indicates that the card hopper is empty and its assertion triggers the pulse generator in B8 to set FLAG. The PG is also triggered whenever a feed check error occurs.

Both modes determined by B assemble standard 36-bit computer words from six 6-bit characters, but the type of information and the number of columns varies. Binary mode (B=1) reads all 12 bits in a single column and assembles three columns into one word. Although 12 bits are taken from the column, it is read as two 6-bit characters. The data lines are designated CRXL where X has the values 1, 2, 4, 8, A, B, and CR1L is the least significant bit. Each column read requires two strobes: the first reads the lower half (holes 4-9) with hole 9 corresponding to CR1L; the second reads the upper half (holes 12, 11, 0, 1, 2, 3) with hole 3 corresponding to CR1L. In alphanumeric (Hollerith) mode, the reader converts the 12 bits in a column to a 6-bit code which is then read by a single strobe. This mode therefore assembles six columns into a single word. Either mode, however, requires six strobes for six characters.

Since the reader is an input device, for initial conditions CONO must provide a PI assignment, set START, and set up B according to the desired mode. Usually the program begins a card in binary because the first column contains information about the card format. Setting START causes the reader to read an entire card even though the program can retrieve no more than six columns at a time. To retrieve a single column at a time, CONO sets FLAG ALL, allowing every column read to set FLAG, which otherwise is set for every six characters. FLAG(1) requests a priority interrupt in the usual manner, and the DATAI response by the program clears FLAG but does not affect START, which is cleared when the card cycle begins. If the program does not respond quickly enough after FLAG is set, the buffer is cleared and DATA MISSED is set, informing the program that there are still six characters in search of a reader. Completion of the entire card sets FLAG even though the buffer may not contain a complete word (a card has 80 columns). The program must provide a new CONO to read the next card.

Figures 8-14 and 8-15 show the 36-bit buffer, which is made up of six Type 4221 6-Bit Shift Registers. Each shift register module contains a buffer bit from 0 to 5 and every sixth bit after it; output from a single data line is always read into a single module. For example, CR1L is associated with that part of the buffer containing bits 5, 11, 17, 23, 29, and 35 (Figure 8-14, upper right). The strobe generated for a character reads CR1L into bit 35 (a 1 is asserted negative), CR2L into bit 34, etc. The first strobe loads the six data bits into the least significant bits of the register and each subsequent strobe shifts the previously read character one place to the left in parallel at the same time that it reads in a new one. The characters are counted in an extra 4221 Shift Register in the upper left of Figure 8-14. Every strobe sets the right bit SR6 and also shifts left those ones that have already been strobed in. The sixth strobe sets SR36 indicating that the entire 36-bit word has been assembled.

Retrieval of information from a card is controlled by the logic shown in the lower left of Figure 8-15. The first CONO pulse clears the entire buffer and character counter (C4) at the same time that it clears the control register. When START is set, SCCL is asserted (D2) causing the reader to process an entire card. When the reader starts the card, CCL asserts CARD CYCLE which clears START but continues to assert the busy status. For alphanumeric mode, CBIL is negated causing the reader to translate each column from the 12-bit Hollerith code to a 6-bit character; if B is 1, however, signals from the hole positions are applied directly to the data lines. The reader indicates that data is ready by sending a column strobe pulse to the control unit, which responds by asserting a level that causes a validity check in alphanumeric mode only. If the column does not contain a valid Hollerith character, the data lines are cleared so all zeros are read; and if the operator has pressed the validity check button on the reader, it stops and must be reset manually. The CSP from the reader also triggers the PA in D1 to generate a strobe that loads the character into the buffer and counts it in SR. The PA output also triggers a delay which asserts CBHL for 20 μsec. This signal has no effect in alphanumeric, but in binary it causes the reader to put the upper half of the column on the data lines in place of the lower half, and the terminating pulse from the delay triggers a second strobe. There are thus six strobes for six characters whether the unit reads six columns or only three. The delay terminating pulse also indicates the presence of a column in the buffer and sets FLAG if FLAG ALL is 1. If full words are being assembled, the fifth strobe sets SR30 which then allows the sixth strobe to set FLAG. When the program responds, DATAI gates the buffer onto the bus, and the turnoff of the gating level triggers the clear to prepare the buffer for the next character. Each pulse from the 20-μsec delay also triggers a 2.2-msec delay. If this terminates while FLAG is still 1, indicating that the data has not been retrieved, DATA GONE clears the buffer and sets DATA MISSED. Finally when the card is finished, CCL from the reader is negated, generating CARD DONE, which sets FLAG and again triggers the DATA GONE delay.

The logic also includes a gate that allows the program to read any column twice. If the program gives a CONO during the card cycle, the set pulse for the initial conditions, which may change the mode, also triggers a short delay whose termination triggers the same PA that is usually triggered by the column strobe. The CONO thus generates a strobe or strobe pair that rereads the column in either the same or the opposite mode. This feature is usually used in conjunction with FLAG ALL to determine what mode to read in a card, based on information contained in the first column. The program usually starts the card in binary to inspect the first column. It may then clear FLAG ALL and reread the first column as part of a full word either in binary or in alphanumeric. The program may also use this feature to reread the third column of a binary card so that the final 78 columns are read in groups of three. If a CONO given during a card cycle sets START, the next card will be read as soon as the present one is finished.

CHAPTER 9

MAINTENANCE

This chapter discusses preventive and corrective maintenance for the arithmetic processor, tape reader, punch, keyboard-printer and card reader. Since neither the processor nor its memories can operate in isolation, the information herein must be used in conjunction with maintenance information in the memory manual. Except for the exhaustive preventive maintenance procedures for the basic in-out equipment, the maintenance is discussed at the system logic level. Circuit troubleshooting and repair are described in PDP-6 Circuits; specific lubrication and adjustment procedures as well as corrective maintenance for the in-out devices are included in the following manufacturer's manuals:

Digitronics Perforated Tape Reader Model 3500

Teletype Bulletin 215B: High Speed Tape Punch Set (BRPE)

Teletype Bulletin 281B: Model 35 Send-Receive Teletypewriter Set (KSR), Vols 1 and 2

Teletype Bulletin 1187D: Parts, Model 35 Send-Receive Teletypewriter Set (KSR)

Burroughs B122 Card Reader Technical Manual (B122.51)

The first section of the chapter discusses operation of the equipment for maintenance purposes and describes those controls and indicators not used in the course of normal system operation. The second section discusses maintenance programs and includes a list of those for the equipment described in this manual, plus the fast and core memories. The third section includes preventive maintenance schedules and a description of marginal check procedures. Finally, corrective maintenance includes troubleshooting procedures and a description of the construction of a diagnostic and exercise program loops. Special tools and test equipment required for the performance of the various procedures are listed below; except for DEC equipment, suggested commercial brands are given for purposes of specification only, and do not constitute exclusive endorsement.

Multimeter	Triplett Model 630-NA; Simpson Model 260
Dual-channel oscilloscope	Tektronix 580 series, preferably with delayed sweep trigger facilities
System module extender	DEC Type 1954
System module puller	DEC Type 1960
Paper tape gauge	DEC Type 18467
1-inch width gauge	DEC Type 0001
Tape reader cleaning kit	Digitronics MS-133
Feeler gauges	Any quality set, 1-25 mils
0-10 pound spring scale	Chatillon 719-10
0-20 pound spring scale	Chatillon 719-20
Tape punch maintenance kit	Teletype Bulletin 1124B lists all special punch tools; order with discretion
Punch lubricants	Teletype KS7470 oil; Mobilgrease #2
KSR-33 lubricants	Teletype KS7471 grease (installations without punch add KS7470 oil)
Card reader lubricants	Burroughs S15821-26 and S64960-2 oils; S15821-32 and -39 greases
Lint-free cloths	Cheese cloth or equivalent
Cotton swabs	Q-tips or equivalent
Cleaning fluids	Dupont Freon TF; denatured alcohol
Test cables and probes	Low-capacity probes for the oscilloscope; alligator clips; etc.
Super Filter Kote (aerosol)	Research Products Corp., Madison, Wisconsin

Also have standard hand tools including Phillips screwdrivers and a complete set of Allen wrenches. The card reader requires a drift punch, a brass drift rod, a plastic-headed hammer, and a pair of 18-inch water-pump pliers. Standard cleansers for the bay exteriors, etc. should also be available. At installations that include magnetic tape equipment, consult the tape transport manual for additional housekeeping necessities.

9.1 OPERATION FOR MAINTENANCE

Chapter 3 discusses normal operation of the processor and basic in-out equipment, and explains the use of the controls and indicators that are regularly available to the operator. There are many additional controls, primarily for maintenance. They include controls for ac line and for marginal checking, and maintenance switches for the logic.

a Power Controls

Most power controls and all power supplies are mounted on the plenum doors at the rear of the bays; all their switches and indicators face outward, directly behind the exterior double doors. The main power control, usually a Type 835, is at the bottom of bay 4 (the left bay of the console viewed from the rear). It has a red light that is on whenever the external ac line is plugged in, a ganged pair of ac circuit breakers, and a LOCAL/OFF/REMOTE toggle switch. Switching to LOCAL turns on power; when the switch is in REMOTE, power is controlled from the console. The total time that power has been on is registered by an elapsed time indicator at the lower left on the in-out panel (Figure 3-3). In some processors the main control is a Type 829 which has an additional switch for a delayed output that is not used. This type of control also supplies a power clear enable directly to the power clear clock; so in a system using the 829 the integrating delay shown in the lower right of Figure 5-1 is not present. Turning on the main power control supplies ac to the fans and the power supplies.

Located in the upper part of the bay 3 plenum door is a secondary power control that supplies ac for the motors in the reader, punch, and Teletype. This control is usually a Type 834, otherwise Type 811; both have controls and indicators identical to those on the 835. When this secondary control is in REMOTE, it goes on whenever the power supplies are turned on by the main control. Also at the top of bay 3 is a panel containing six ac convenience outlets.

As explained in 3.2, power to the reader and Teletype motors is controlled by the operator. The ac line to the punch is controlled by a Type 823 (mounted directly on the punch motor) through which the logic turns the punch on and off. For maintenance purposes the logic may be bypassed by means of an ON/OFF toggle switch located behind the right end of the chad box. The card reader can go on only if the operator has turned on its main power switch located on the left side of the stacker. Then it turns on with system power in the same way as the 834: a −15 vdc turnon signal is supplied through the I/O bus. If system power is off, the reader may be turned on and off independently by the POWER ON and POWER OFF buttons on the reader console.

b Marginal Check Controls

At the top of the bay 4 plenum door is a Type 734 Power Supply whose floating output may be varied from 0 to 20 volts. Its controls are on the front of the console at the bottom of the in-out panel (Figure 3-3). The 3-position switch controls the polarity of the supply output; the large knob controls the output voltage amplitude, and the dc voltmeter indicates the magnitude of the output (a similar meter is mounted directly on the supply).

In every logic mounting panel (except the lower panel of a pair containing double-height modules) pins A to D are connected to the power and ground lines. To allow submodular marginal checking, there are two independent +10 volt power lines, +10A and +10B; one is bussed to all A pins in a panel, the other to all B pins. All D pins are grounded. The C pin on every module receives −15 volts, but all the C pins are bussed together only if the panel contains no pulse amplifiers. The C pins of the pulse amplifiers in a mounting panel are separately bussed to permit independent application of negative marginal check voltage to them (in the processor the pulse amplifiers are module types 1607, 1609, 4606, 6603, 6609); the remaining modules in the panel always receive the fixed −15 volts.

At the left end of each logic panel are three toggle switches. When all three switches are down, all the modules in the panel receive the normal fixed voltages. Turning the polarity switch to +10 MC and pushing up the top or middle toggle switch applies the output of the variable power supply to the +10A or +10B bus respectively on the panel. Turning the polarity switch to −15 MC and pushing up the bottom toggle switch applies the marginal check voltage to the C pins of the pulse amplifiers in the panel. Turning the polarity switch OFF applies normal voltage to all power lines regardless of the setting of the toggle switches. While marginal checking a panel, the toggle switches need not be turned off when switching from one polarity to the other because the polarity switch applies correct fixed voltages to all unselected lines (i.e., those of opposite polarity) even if the individual panel switches are left on. However, this interlock and the center-off polarity switch position are provided only as a convenience and should not be used as a substitute for turning off the toggle switches. At the completion of marginal check procedures the technician should turn off all three marginal check toggle switches on every mounting panel.

Marginal check voltages may also be applied to the photo amplifiers in the tape reader. Two switches, one for the eight code holes, the other for the feed hole, are located below the console shelf in the upper right corner of mounting panel 4K. When both switches are in the +10V position (left), all amplifiers receive the normal fixed +10 volts. Turning the polarity switch to +10 MC and pushing the FEED HOLE switch to the right (also labeled +10 MC) applies the output of the variable supply to the amplifier for the feed hole photodiode output. Similarly the CODE HOLE switch allows application of marginal voltage to the amplifiers for the information holes.

c Maintenance Switches

Mounted on brackets between the mounting panels on the wiring side are five toggle switches that allow the technician to alter the normal operation of the processor as an aid in troubleshooting the logic. All of these switches are off when down. One switch was mentioned in Chapter 3 because it must be used when depositing a readin loader in the bottom of core. The RIM MAINT switch is mounted just above 2M1,2, and it is shown in the lower left of Figure 5-2. Turning the switch on grounds the 0 output of the RIM SBR flip-flop so that it cannot be cleared no matter what location is used for instruction retrieval. The processor thus stays in the readin mode indefinitely, and a memory checkerboard may be run that includes the readin area. Also shown on Figure 5-2 is the REPEAT BYPASS switch which is mounted above 1M22, 23. While this switch is on, KT0A triggers the repeat delay, so the key cycle can be repeated even when the chain does not include KT4.

Two of the switches provide interruptions in the normal flow of events in the arithmetic and shift-count subroutines. Shown in the upper right of Figure 6-9 and mounted above 1H18, 19 is ART3 MAINT, which prevents the carry completion in an AR subroutine from generating the return pulse ART3; the return is instead simulated by KT2. A technician may single step through a shift-count by using SCT MAINT, which is mounted above 2C7,8 and shown at the left in Figure 6-16. With this switch on, neither SCT0 nor SCT1 can trigger the next step in the subroutine; instead each step is triggered by KT2.

The last switch may be used for troubleshooting the memory from the processor. This is SPLIT CYCLE OVERRIDE, shown at the right in Figure 7-9 and mounted above 1N15, 16. Ordinarily if the MEMORY STOP key or the ADDRESS STOP switch is on, AT4 sets the split cycle sync flip-flop. Then if the fetch cycle should generate a fetch-and-pause request, the memory subroutine makes only a read request to memory; so a processor that is undergoing troubleshooting procedures does not hold the memory during a stop. But in troubleshooting the memory it may be desirable to stop it between the read and write parts of a cycle. The override switch accomplishes this objective by disabling the set gate to the flip-flop.

Other than the power and marginal check controls mentioned in a and b above, there are no special maintenance switches for the reader, punch, or Teletype. The card reader, however, has switches inside the hopper and stacker that generate NOT READY when the former is empty or the latter full, and these may be operated manually for maintenance. Under the reader cover is a local/remote toggle switch, LOCAL RUN. Raising this switch (local) causes the reader to feed cards continuously until the hopper is empty, and then stop with FEED CHECK on.

Besides the switches there are many maintenance indicators. This category includes at the top of bay 1 most of the indicators which display the states of all SBRs and many control flip-flops and levels. The PI REQUEST lights on the console indicate those channels on which requests have been synchronized, but the signals arriving at the logic over the I/O bus are displayed by a set of lights mounted in front of module connector 2L20. The top light is for the PI1 line.

d Single Step Operation

By using the STOP and CONTINUE keys for instructions, the operator may single step a program from one instruction to the next. Similarly, for maintenance the technician may single step from one memory call to the next by using the STOP and CONTINUE keys for memory. In either case the STOP key should be latched on, and each restart may be done manually by pressing the CONTINUE key; or the single stepping can be performed automatically by latching on the CONTINUE key, turning on the REPEAT switch, and setting the SPEED controls to the desired time interval. Each time the processor stops following a memory call, the MEM STOP light is on, and at the top of bay 1 a light must be on for the SBR that is awaiting the subroutine return. The meaning of the information displayed by the indicators on the console and the bay indicator panels can be determined by following the flow charts as the processor proceeds. Using REPEAT, the function associated with any initiating key can be repeated; all require that KT4 set the repeat delay, and the required switch settings are discussed at the end of Chapter 5.

By using the maintenance switches discussed in c above, the processor can be operated in even finer steps. Turning on ART3 MAINT inhibits the return pulse from the AR subroutines; so the processor stops every time an addition, subtraction, index, or negation is performed. The immediate results of the subroutine can then be seen by observing the lights for the arithmetic registers on bay 2, and the point at which the stop occurred is indicated by one of the SBR lights on bay 1. To observe the operation of the shift counter and the effect of shift pulses applied to AR and MQ, turn on SCT MAINT. The processor then stops prior to the shift-count subroutine and also following every step in it. After either an ART3 or an SCT stop, the processor should be restarted by pressing MEMORY CONTINUE. The attendant key cycle supplies the necessary pulse for simulating the subroutine pulse, but triggers no other operations because there is no memory stop.

To produce an automatic restart after every stop, turn on REPEAT BYPASS and latch on MEMORY CONTINUE; this causes every KT0A to retrigger the repeat delay, and the level from the key causes the delay termination to retrigger the key cycle. The ART3 and SCT switches can be left on together, so the processor stops after every AR subroutine and every step in a shift-count; if MEMORY STOP is left on too, the processor will also stop after every memory subroutine. MEMORY CONTINUE supplies the restart for all three types of stop.

While the processor is single stepping, the technician should follow the operations in the flow charts and compare the information given by them with the state of the processor as displayed in the lights. The point at which the processor has stopped is indicated by the lights on bay 1. The indicators display a great deal more information about the workings of the processor at stops within an instruction than they can provide when the processor is stopped only at the end of an instruction. For example, the indirect light I is always off at the completion of an instruction, but when single stepping through an address cycle with AR and memory stops, the light may sometimes be on. For particulars always refer to the flow charts.

9.2 MAINTENANCE PROGRAMS

MAINDEC (maintenance DEC) programs permit self-testing of the PDP-6 for check-out, preventive maintenance, or diagnosing equipment malfunctions. Each MAINDEC package consists of program tapes and a reference manual. All manuals have the same format: an abstract; a section containing operator information; another suggesting applications of the programs; and a third describing the programs.

The first section is all that the operator need use when running a MAINDEC. It lists the required tapes, specifies the usage of the console switches, and gives detailed instructions for loading and starting the programs. The final part specifies how errors are indicated to the operator (e.g., programmed halts or typeouts), provides information as to the cause of the error, and tells the operator how to repeat or restart the program. The second section suggests various applications including procedures for using the MAINDEC with marginal checking. It specifies the panels and margin levels that apply to each program. The third section contains detailed program descriptions including octal and symbolic listings; flow charts are included where appropriate. Each description explains what the program is testing and in what manner, and furnishes information to aid in updating or modifying the program.

The following MAINDECs are applicable to the basic hardware (consult the DEC program library for the current list of all MAINDECs for PDP-6).

Test	MAINDEC
Instruction Test	
Part 1	601-1
Part 2	601-2
Part 3	601-3
Part 4	601-4
Part 5	601-5
Micro Checkerboard	602
Memory Address Test	603
Clock Test	604
Memory Speed Test	605
Memory Retention after Power Failure	606
Memory Overlap	607
Power Failure Test	608
Reader Binary Test	610
Reader Alpha Test	611
Punch Test	612
Memory Data Test	613
Teleprinter Test	614
Memory Checkerboard	
Low 64 x 4K	662-1
Hi 4 x 4K	622-2
16 x 16K	622-3
16 x 16K Interleave	622-4
Protect and Relocate Test	623
Card Reader Test	641
Fast Memory Test	662

9.3 PREVENTIVE MAINTENANCE

This section discusses preventive maintenance schedules and use of marginal check during PM procedures, and lists recommended procedures for the arithmetic processor and the basic in-out equipment. Preventive maintenance consists of tasks performed prior to initial operation of the system and periodically during its operating life to ensure that it is in satisfactory operating condition. Faithful performance of these tasks forestalls possible future failure by discovering progressive deterioration and correcting minor damage at an early stage. The tasks consists of mechanical checks, including cleaning and inspection; marginal checks, which aggravate borderline circuit conditions or intermittent failures to make them easy to detect; and checks of specific elements such as power supplies, in-out interface circuits, drivers, and sense elements.

a Schedules and Margins

Preventive maintenance procedures should be performed according to strict schedules. Recommended intervals for PM checks are based on elapsed operating time as well as calendar schedules. For convenience in scheduling normal operations, and to minimize malfunctions, large-scale PM procedures that occur at long intervals should be staggered. Each user should set up a schedule for his entire installation that distributes the total PM task evenly over the longest interval recommended. In every operating shift, a specific period should be assigned to performance of the scheduled PM program for that shift. Without an appropriately designed program of this type the PM function would necessitate system unavailability for two or three days every month.

Marginal checking utilizes the MAINDEC diagnostic programs to test the functional capabilities of the system with module operating voltages biased within specified margins above or below nominal levels. Failures brought about by over- or under-biasing are detected by the program, which provides a printout or other visual indication helpful in locating the source of the malfunction. Marginal components can then be replaced during scheduled preventive maintenance. After such preventive replacement, or when no marginal components exist, operating voltages are then biased beyond the specified margins, and the margins at which circuits fail are recorded in the maintenance log. By plotting bias voltages obtained during each scheduled PM, progressive deterioration is easily observable and expected failure dates are predictable. Accurate information acquired over a long period provides a basis for planning future preventive replacement. These plots are also useful in locating marginal or intermittent components such as deteriorating transistors.

Raising the operating voltage above +10 volts increases the transistor cutoff bias that must be overcome by the previous driving transistor; therefore low-gain transistors fail. Lowering the voltage below +10 volts reduces transistor cutoff bias and noise rejection, and thus provides a test for high-leakage transistors and simulates high temperature conditions (to check for thermal runaway). Raising or lowering the −15 volt supply (this is done only to pulse amplifiers) increases or decreases the output pulse amplitude. Each panel has a separate negative bus to pulse amplifiers; the line to pin C for logic level modules comes directly from the power supply, whereas the line to pulse amplifiers comes through the marginal check switch. The +10 margins are normally +5 and +15 volts; the −15 margins are −7 and −8 volts. The −15 volt supply must never be made more negative than −18 volts or damage to the logic can result.

For every system in the PDP-6 the user should keep preventive maintenance voltage charts on which are plotted the failure margins for each major section of logic while running the appropriate MAINDEC. DEC supplies a standard form for such plots; each page has three pairs of charts for +10A, +10B, and −15 marginal checks. The ordinate is the margin level as read from the meter on the supply; the abscissa is the time (nine spaces are provided below each chart for the date). At each check-out, two points are plotted: the voltage level which is too high for normal operation, and that which is too low. Thus a chart shows the change in time of the voltage region over which a particular section of the logic operates properly. When equipment malfunction is induced by abnormal margin levels (e.g., the region being too narrow), troubleshooting procedures (9.4) should be performed to diagnose the abnormality. At the top of each chart, spaces are provided for entering the page number in the maintenance log where such dysfunction is explained.

b Arithmetic Processor PM

The procedures in this section apply only to the arithmetic processor; procedures for the in-out equipment are given in c below.

Daily Operator Maintenance

1. Run all five parts of the Instruction Test (MAINDEC 601) without margins. Log all error halts, noting the cause if known.

2. Check that all cooling fans of the system are running and that cooling air flows freely through the filters.

3. Replace any noncritical components such as indicators, fuses, etc; note any replacement in the log.

The remaining procedures are to be performed by trained personnel only.

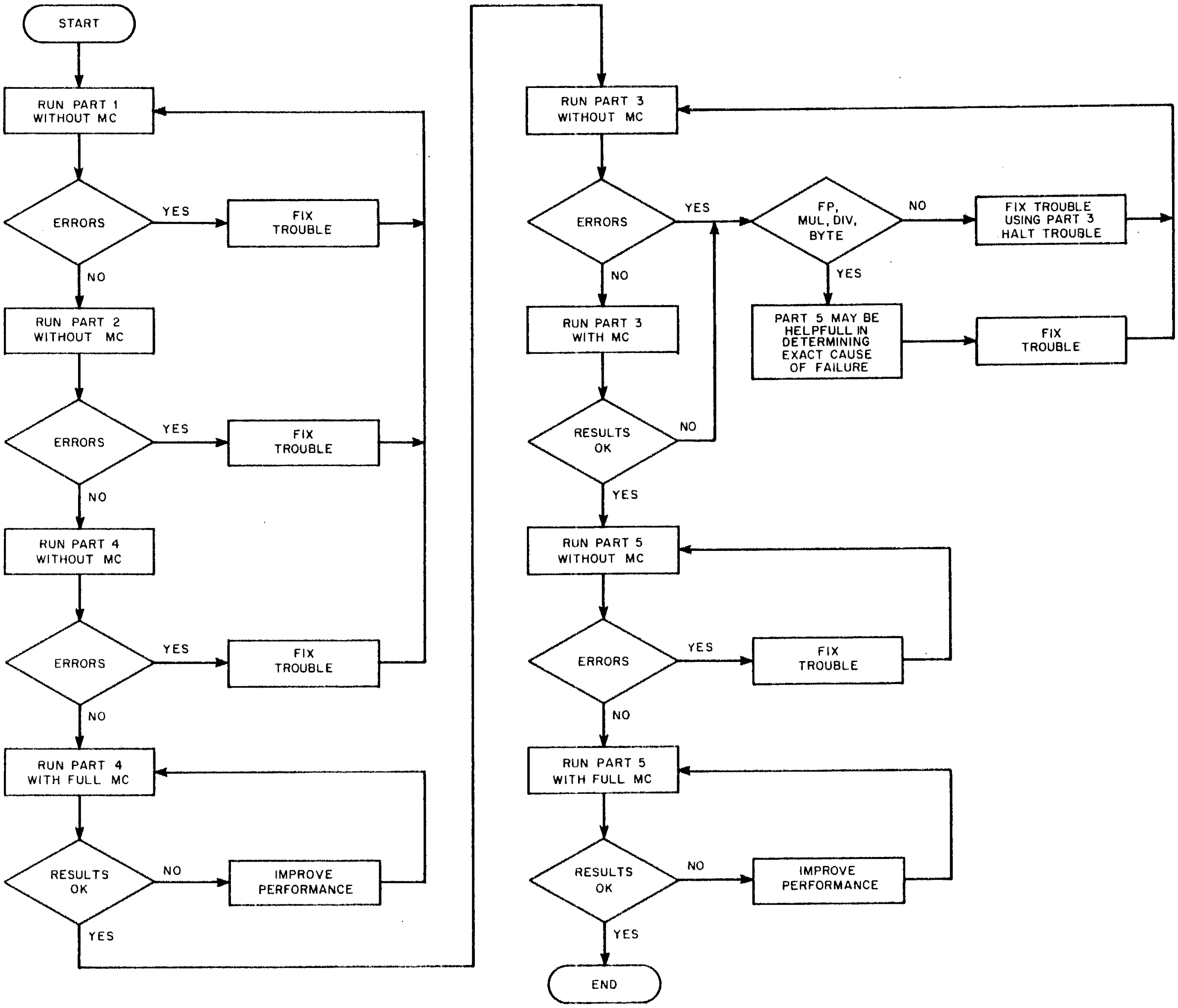


Figure 9-1 Processor Marginal Check Flow

Weekly

1. Check the operator log; note any malfunctions which have occurred in the machine, and take corrective measures if necessary.

2. If trouble noted in the log cannot be reproduced by normal trouble-shooting methods, contact the operator who made the note; using his program, try to reproduce the trouble. If the system can be kept operational by use of moderate margin voltages on certain small sections of logic, do so. Do not fail to make an appropriate entry in the log describing any provisional measures adopted, and notify the scheduling authority that diagnostic down time will be required to rectify the trouble.

Every 1000 Hours

1. Run all five parts of the Instruction Test (MAINDEC 601) with margins, using the flow diagram (Figure 9-1) and plotting the results on the preventive maintenance voltage charts. In case of failure at abnormal margin levels, note all circumstances in the maintenance log and cross reference the log to the chart by page number.

2. Change and clean the air filters at the bottom of every bay in the entire system using the following procedure:

   a. Loosen the two thumb screws holding the fan and filter housing to the floor of the bay.

   b. Remove the housing. Take the filter out of the housing and install a clean one.

   c. Replace the housing containing the clean filter and tighten the two thumb screws.

d. Clean the dirty filter by flushing it thoroughly with hot tap water in a direction opposite to that of air flow. When all dust and lint is removed shake out excess moisture.

e. Stand the filter on one end for 10 to 15 minutes to allow remaining moisture to evaporate. If the flush water is sufficiently hot, the filter should dry completely in about 15 minutes.

f. Spray the filter with aerosol Super Filter Kote or an equivalent product. The spray serves both as a dirt capturing medium and as a detergent which helps wash out trapped dust and lint during the next reverse flushing.

c In-Out Equipment PM

This section includes preventive maintenance procedures for the tape reader, punch, keyboard-printer, card reader, and DEC logic associated with them. Procedures for all four devices are grouped together under the recommended PM intervals.

The paper tape equipment requires accurately punched tape. A DEC standard paper tape gauge is supplied with the system; use it for all paper tape measurements. A paper tape with the correct specifications is shown on the next page.

The most important dimension is the lateral distance from feedhole to inner edge: it must be .392 ± .002 inch. When examining the tape on the gauge with the naked eye, this dimension must be exact—the smallest visible error will exceed the tolerance. The reader has a wide tolerance to variations in longitudinal hole spacing, but mechanical readers in general do not. To ensure that tape punched by the processor will be readable on mechanical readers such as the Teletype Model 35 or the Flexowriter, longitudinal hole spacing must be within ±5 mils accumulated error in 6 inches. The 5 mil tolerance is about one eighth the diameter of the feedhole. The output of every punch in the system should be checked once every day to ensure that the base dimensions are within tolerance. It is much easier to punch the tape accurately than to try to jockey the reader adjustments to read a wide tolerance of tape dimensions.

The punch can be adjusted to produce accurate lateral spacing on any given tape whose width is within 7 mils of the nominal 1 inch. Once it is so adjusted, any tape of 1 ± .007 inches will punch properly. If possible, make adjustments while punching the narrowest tape (.993 inch).

The following procedures are recommended for the in-out equipment.

Daily Operator Maintenance

Tape Reader –

1. Using Digitronics cleaning kit MS-133, clean the read head, tape guides, roller bearings, capstan, and brake.

2. Run the reader tests (MAINDECs 610, 611) without margins. During the tests, check that the tape is laterally positioned to ride along the inner edge of the tape guide; the photoelectric read head is designed to accept tape in this position. Log all error halts, noting the cause if known.

3. Remove the exciter lamp cover and check that the lens is clear and the photodiodes clean; replace the bulb if there is any sign of yellowing.

4. Check that the lamp cover springs are adjusted so as to just touch the top surface of the tape.

Tape Punch –

1. Run the punch test (MAINDEC 612) without margins. Log all error halts, noting the cause if known.

2. Using the DEC standard tape gauge, check that the punch output conforms to the following standards (see Figure 9-2).

   a. Feed hole to inside edge: .392 ± .002 inch.

b. ±5 mils accumulated longitudinal error in 6 inches.

c. Width: 1 ± .007 inch.

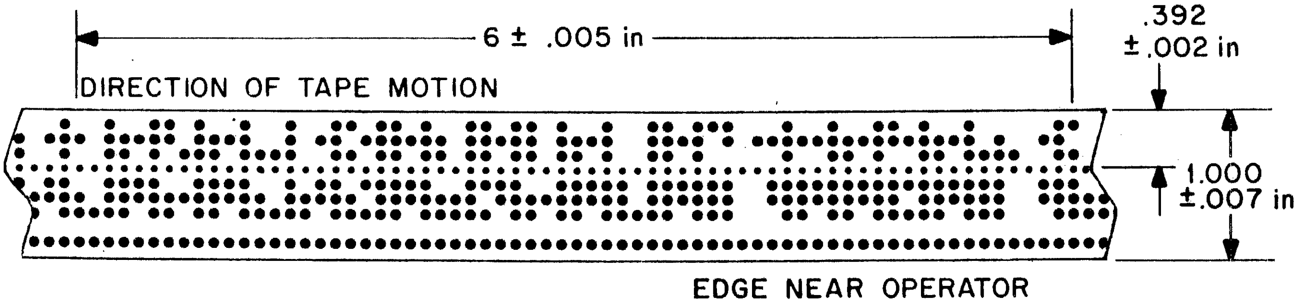


Figure 9-2 Paper Tape Dimensions

3. Check for "fuzzy" holes. This indicates a misadjusted or worn out die block.

4. Empty the chad box (the chad box should also be emptied every time a new box of tape is installed).

5. Clean the punch by blowing dust off the die block. Do not use alcohol or other solvents near the feed pawl or die block since such solvents remove the light lubricating film.

Teleprinter –

1. Inspect and clean the platen and paper guides as necessary (in general, the platen needs cleaning only if typing has run off the page or the printer has been run without paper).

2. Remove lint and other fouling material from the ribbon guides and replace the ribbon if necessary.

3. Run the Teleprinter Test (MAINDEC 614) without margins. Log all error halts, noting the cause if known.

Card Reader –

1. Remove the exciter lamp assembly. Clean the solar cells, the feed head, and the hopper area with a soft bristle brush.

CAUTION

Use only Freon as a cleaning agent for the solar cells. Other cleaning fluids may damage the potting in the assembly.

2. Run the Card Reader Test (MAINDEC 641) without margins. Watch for unusual wear on the test deck. Log all malfunctions with probable cause if known.

Every 160 Hours (Monthly)

Tape Punch –

1. Remove the punch from the console and perform the complete lubrication procedure outlined in 5 of Teletype Bulletin 215B. Be sure that no oil or grease accumulates between the armatures and magnet pole faces or between contact points.

2. Inspect the general condition of all moving parts and tightness of wiring connections. Make sure nuts and screws that lock the adjustments are tight, but do not apply sufficient torque to disturb the adjustment. See that all contact points meet squarely. Rotate the main shaft slowly in the normal direction (clockwise as viewed from the front) and activate all movable elements. Check for freedom of movement.

3. Run the Punch Test (MAINDEC 612) with margins. Plot the failure points on the PM voltage chart; log abnormal failure margins and cross reference the log to the chart by page number.

Every 330 Hours

Tape Reader –

1. Carefully perform the daily PM paying particular attention to the position of the tape as it proceeds through the reader, and making sure that it travels against the inner edge of the tape guides. Use an accurately punched paper tape, i.e., one which is well within the dimensional tolerances in Figure 9-2.

2. Carefully check the alignment of read head photodiodes with the holes in the accurate tape (with lamp cover still removed). Turn on the reader and thread the tape through the guides but not through the clutch or brake. Using a line with all holes punched, center the feedhole directly over the feedhole diode. The diodes for holes 1 and 8 must lie directly under the centers of those holes. If this alignment is incorrect or if skew is apparent, correct it by shimming either the read head or the tape guides.

3. Check the exciter bulb voltage: 8.5 ± .1 volts.

4. Check the pinch roller to capstan gap clearance with clutch de-energized. The proper clearance is 10 to 12 mils; if the clearance is insufficient, dirt may be limiting the full travel of the clutch solenoid.

5. With no tape in the reader, turn it on and lift the READER FEED key to engage the clutch as if reading tape. The correct clutch setting is that which allows the pinch roller just barely to engage the highest point on the rotating capstan, causing a slight jittering sensation in the finger when held against the roller. If the roller does not contact the capstan, or runs continuously with it, adjust the clearance according to Section 4-13 of the Digitronics manual. Severe jitter indicates excessive capstan runout, which should be tested as follows:

a. Turn the reader off, pull it from the console on its extension slides, and remove the drive belt from both capstan and motor pulley.

b. Turn on the reader and lift READER FEED to engage the clutch. Using 1- and 2-mil feeler gauges check the clearance between capstan and pinch roller for various angular positions of the capstan, turning it by hand. If runout is excessive, the capstan must be replaced.

c. Turn the reader off and replace the belt.

6. Using a spring tension scale, check that the force necessary to unseat the activated pinch roller is from 3 to 4 pounds. If adjustment is necessary use the procedure in Section 4-15 step 2 of the Digitronics manual.

7. Check for ridges on the capstan around the area where the edges of the tape ride. Such ridges can cause tape skew; they may be removed by applying a fine file to the capstan while the reader is running. Do not allow rubber dust to collect on the clutch block.

8. Check brake tension using a short piece of paper tape with a loop in one end. Insert the free end of the tape between the brake solenoid and its armature; use a spring scale hooked into the loop to measure the tension required to move the tape with the brake engaged. The force must be from 3-1/2 to 7 pounds.

Teleprinter –

1. Lift the cover and check everywhere for effects of vibration: loose nuts, screws, retaining clips, etc.

2. Check the selector magnet coils for signs of overheating. Make sure there is no lubricant or other fouling material under the armature. If necessary,

insert a piece of bond paper between the pole and armature to soak up any lubricant. Make sure no lint is left.

Card Reader — Sections and figures referenced below are in Burroughs Card Reader Technical Manual B112.51.

1. Check exciter lamp brilliancy according to 3.16; adjust if necessary. The procedure requires a deck of cards punched with a repetitive pattern of all ones in twelve columns followed by all zeros in the next five. This step and step 2 must be performed whenever photo amplifiers are changed in the reader.

2. Using the same test deck, check the leading edge timing of the strobe pulse with respect to the read pulse for all twelve rows (Section 3.17).

3. Check operation of the strobe-8 counter according to Section 3.18 (performance can be checked only by adjusting the counter).

4. Check response of card-detect solar cell CD1 according to Section 3.19 (the procedure requires a deck of cards having a 4 punch in column 80).

5. Check performance of the translator for valid Hollerith codes (Section 2.3), using a deck prepared by punching each valid code in alternate columns of a card.

6. Test performance of the validity check circuits (Section 2.4), using a 50-card deck in which column 40 of each card contains one of the invalid punches (all invalid combinations are listed at the bottom of Table A4-3).

7. Using Burroughs S64960-2 oil, lubricate the feed-slide/gib bearing surfaces (Figure 3.4-1, right), the feed-knife pivot (Figure 3.4-1, C), and the feed-slide fork/actuator-arm pivot post bearing surfaces (Figure 4.5-1, top right).

8. Run the Card Reader Test (MAINDEC 641) with margins. Plot the failure points on the PM voltage chart; log abnormal margins and cross reference the log to the chart by page number.

Every 1000 Hours

Tape Reader –

1. Perform the daily and 330-hour PM schedules. Tape must travel along the inner edge of the tape guides.

2. Check all power supply voltages including the exciter lamp voltage.

3. Using the same length of accurately punched test tape with alternate lines of ones and zeros, run the Reader Tests (MAINDECs 610, 611) with margins applied to the read amplifiers and reader control logic. Run the reader at top speed (400 characters per second). Make no adjustments unless the tape being read has lateral dimensions well within the tolerances in Figure 9-2, and travels against the inner edge of the tape guides.

4. Observe the feedhole signal at pin 18M in the panel containing the reader logic (beneath the processor console) with one vertical input of a dual-channel oscilloscope. Adjust the feedhole output for a width of 1 msec.

5. Apply a +9 volt margin to the code hole amplifiers, and adjust the read amplifier cards while observing both the code hole and feedhole signals with the scope. The trailing edge of the feedhole should lead the trailing edge of the code holes by 300 to 500 μsec; code hole width should be approximately 1.8 msec when it satisfies this trailing edge timing requirement. A read amplifier that exhibits extreme changes in gain with moderate rotation of the adjustment potentiometer should be replaced with a less sensitive card.

6. Using MAINDEC 610, run the test tape at all speeds up to 400 characters per second with code hole margins from 3 to 17 volts and feedhole margins

from 5 to 15 volts. Amplifiers should be touched up as necessary to obtain these margins. Error halts occurring with a splice on the test tape between clutch and brake can usually be disregarded since these are generally caused by poorly made splices. Plot the margin levels on the PM voltage chart; if adjustments are required, log the prior and post adjustment margin levels and cross reference the log to the chart by page number.

Tape Punch –

1. Check the Teletype Maintenance Kit for parts which may require periodic replacement.

2. Run the Punch Test (MAINDEC 612) with margins. Plot the failure points on the PM voltage chart; log abnormal failure margins and cross reference the log to the chart by page number.

Teleprinter –

1. Clean the distributor with Freon and a cotton swab.

2. Perform the complete teleprinter lubrication procedure as prescribed by Teletype Bulletin 281B, Volume 1 (there are four applicable sections; see the Bulletin table of contents).

3. Run the teleprinter test (MAINDEC 614) with margins. Plot the failure points on the PM voltage chart; log abnormal failure margins and cross reference the log to the chart by page number.

Every 2000 Hours

Tape Reader –

1. Check all electrical connections for electrical and mechanical security.

2. Inspect all moving parts for wear.

Card Reader – Sections and figures referenced below are in Burroughs Card Reader Technical Manual B122.51; part numbers specify Burroughs lubricants.

1. Check the following adjustments:

   a. Top and bottom feed roller clearance and feed-roller pivot arm spring tensions (Section 3.1).

   b. Exciter lamp bracket to read-bed plate clearance; light spot registration; side plate to insulation block clearance (Section 3.2).

   c. Top and bottom hopper throat blade clearance (Section 3.3).

   d. Feed-knife to card-column-80 clearance (Section 3.4).

   e. Clutch collar to actuated trip-arm clearance; actuated trip-arm to rear solenoid core clearance; disengaged trip-arm/clutch-collar latching ridge overlap (Section 3.5).

   f. Gap between card deck and front card guide, and slide to left gib clearance (Section 3.6).

   g. Stacker-stop-actuator arm position (Section 3.7).

   h. Reluctance-pickup strobe output pulse duration; pickup tip protrusion and orientation (Section 3.8).

   i. Clutch-reset and feed-test circuit breaker timing (Section 3.9).

2. With S64960-2 oil, lubricate the feed-roll arm pivots (Figures 4.3-1,2), the circuit breaker contact-arm pivots (Figure 4.4-1), the clutch trip-arm pivot (Figure 3.5-1), and the stacker-switch actuator shaft (Figure 3.7-1).

3. Using S15821-39 grease, lubricate the feed roll shaft gears (Figure 4.3-3), the feed cams (Figure 4.5-1), and the circuit breaker cams (Figure 4.4-1).

4. Using S15821-32 grease, repack the clutch housing and spring (Section 4.5-1).

5. Using S15821-26 oil, lubricate the drive motor bearings.

6. Check the following bearings for wear as evidenced by excessive bearing noise under rotation; replace worn bearings using the procedures of Section 4.

   a. Feed-roller shaft bearings

   b. Cam-follower roll bearings

   c. Pulley bearings (at the clutch assembly)

   d. Clutch shaft bearings

   e. Card-feed actuator-arm shaft bearings

9.4 CORRECTIVE MAINTENANCE

PDP-6 is constructed of highly reliable, solid state modules, and the possibility of failure during the first year after installation is exceedingly remote. The maintenance staff should use this period to acquire total familiarity with the system and all its documentation — approaching the system in ignorance guarantees frustration. This section contains no fault classification system with specific remedies; instead it outlines a broad plan of attack for isolating malfunctioning hardware. When diagnosis is complete, proper remedial action is manifest. Diagnosis depends on logical thinking, common sense, secure knowledge of the system, and an organized step-by-step procedure.

Do not attempt to memorize the prints; instead concentrate on associating the drawing number mnemonic with the logic section portrayed. When system logic is well understood, the logic drawings and flow charts are generally sufficient reference documentation during troubleshooting. Logic drawings are cross-referenced to the manual by section number (it appears with the figure number in the lower right) allowing recourse to the text in cases of extreme difficulty. In addition to the DEC prints and manuals, there are manufacturer's manuals for the in-out devices. Familiarity with the contents of these manuals avoids time wasted in futile searches for information.

When confronting a malfunction in the machine, the following plan of attack should be employed:

1. Initial investigation: gather all available information on the problem.

2. Preliminary check: see if the malfunction presents obvious physical symptoms.

3. Console troubleshooting: use pertinent diagnostic programs, maintenance controls, marginal check procedures, etc., to localize the problem within a particular section of the logic.

4. Logic troubleshooting: complete the diagnosis by isolating the malfunction within a particular module, supply, or control.

5. Remedial action.

6. Validation of remedy: ensure that proper system function is restored.

7. Log report: make a complete record of diagnosis, remedy, and validation.

Troubleshooting, which is discussed in a below, includes the first four steps above. The diagnosis is expected to suggest a remedy (step 5), but the remedy may require attendant procedures; such situations are treated in b. The final two steps are discussed in c.

a Troubleshooting

This section details the diagnostic procedure to explain how faults may be isolated to individual modules; troubleshooting within modules is discussed in PDP-6 Circuits.

Initial Investigation

Before commencing troubleshooting procedures, explore every possible source of information. Ascertain all possible information concerning any unusual function of the machine prior to the fault and all symptoms evident when the fault occurred, such as the type of program in progress, conditions reflected by console indicators, etc. Search the maintenance log to determine whether the present fault or a similar one has occurred before or whether any cyclic history of related malfunction exists, to determine how similar past conditions were corrected. Examine the PM voltage charts for any steadily narrowing operating voltage regions under marginal check. If a deteriorating module produces failures seemingly related to the present trouble, diagnosis may be simplified. The more information the technician can gather, the more rapidly he can make a diagnosis. Attempting to troubleshoot the system without first exploring every available source of information usually just wastes time.

Preliminary Check

Troubleshooting begins with a check for physical symptoms of malfunction. If a very large portion of system logic is inoperative, inspect the physical and electrical security of power sources, cables, connectors, etc. Be sure that power supplies provide proper voltages. Check the condition of all filters for free flow of air; a clogged filter may allow the temperature within a bay to rise sufficiently to cause marginal semiconductors to fail. This preliminary check is useful more often for catastrophic malfunctions than for intermittent ones. Except for cable and module connections, most intermittent failures are due to cold-soldered joints or faulty circuit components. Unless the malfunction is isolated with certainty to within two or three modules by the initial investigation, it is poor strategy to start checking arbitrary modules; more sophisticated troubleshooting procedures must be used. Nevertheless, the preliminary check often discloses troubles which would otherwise require considerable diagnostic work. Few things are more annoying than the discovery of a simple power supply failure after complex, time-consuming troubleshooting procedures.

Console Troubleshooting

In many cases the initial investigation discloses an appropriate line of attack, but does not in itself pinpoint the location of a malfunction. Troubleshooting from the console localizes a malfunction within a small section of machine logic, and is usually accompanied by use of the MAINDECs (with or without marginal checking, depending on the nature of the malfunction). Intermittents caused by weakened components can almost always be aggravated and thus transformed to relatively consistent malfunctions by use of marginal checking with an appropriate MAINDEC (Section 9.2), selected on the basis of information derived from the initial investigation. Intermittent connections, however, are substantially more difficult to locate.

If the malfunction does not show up in the first run without margins, perform the operation in which the trouble was initially observed, using the same user program. In-out equipment faults frequently give indications similar to those caused by processor malfunctions. Since the entirely solid state processor is inherently more reliable, check the peripheral equipment first. Faulty ground connections between periphery and processor are common sources of trouble.

If a malfunction is caused by loading conditions unique to a user program, it can generally be detected by a MAINDEC run with margins. Malfunctions caused by differences in loading conditions show up at narrower margins than those listed in the last few plots on the PM voltage charts. If the MAINDEC discovers the error during application of marginal check, do not restart the computer; use the error halt listings to diagnose the trouble.

Console troubleshooting procedures for locating catastrophic malfunctions, or those made consistent through use of margins, should be directed toward discovering a pattern of consistency among the errors. Although some ingenuity may sometimes be necessary to discover it, all malfunctions display consistency in the errors they produce; separate error diagnoses must intersect at a common fault point.

Logic Troubleshooting

Logic troubleshooting is detail work performed on a small section of the logic after a malfunction has been detected but before the source has been identified. The two main parts of troubleshooting are to reproduce the malfunction and then to identify the source. A common technique for reproducing malfunctions is the use of diagnostic loops, though at times it may be necessary to reproduce the exact conditions under which the failure first occurred, e.g., by repeating the program that was running at the time. For an intermittent fault it may be useful to try aggravation techniques while a diagnostic loop is being run.

In general a diagnostic loop is a small string of instructions that causes the computer to perform some operation and checks that it is executed correctly; usually the error is indicated by having the computer halt. If the operation is executed correctly, the program starts over; in this manner the computer runs until a malfunction is encountered. The technician may design his own diagnostic loop and toggle it in from the console, but it is usually easier to modify the maintenance program. MAINDEC 601 consists entirely of small diagnostic routines seldom containing more than ten instructions. Each routine halts the processor on an error by using JRST 4 with the address pointing to the first loaction of the next routine. Any routine may be converted into a diagnostic loop by replacing the instruction following the halt with a jump to the beginning of the routine.

Aggravation is intended to convert intermittent malfunctions to catastrophic ones. The technique involves either marginal check voltages or vibration (weak component detection by marginal checking is part of console troubleshooting described above). As long as reasonable care is used to avoid inflicting permanent damage, maintenance personnel should not hesitate to twist and probe at connections, cables, plugs or modules. All connections in the system are designed for excellent reliability; so plugs, cables, or sockets should be impervious to any reasonable amount of pulling or flexing. Although PDP-6 systems are tested before leaving the factory, nevertheless, a poorly soldered connection occasionally shows up; this type of fault usually appears as an intermittent and is sometimes very difficult to locate. Poorly soldered connections are more likely to appear in mounting panel or cable connections than within modules. Well hidden malfunctions of this type are occasionally located by tapping suspect modules with the plastic handle of a screwdriver or other such harmless implement.

Once the malfunction can be reproduced, the next step is to identify the source and correct it. Three common techniques are substitution, observing panel indicators, and signal tracing. Substitution is simply replacing a suspect module to see whether a malfunction is thereby cured. Whenever possible, swap modules rather than substituting a spare, e.g., swap counter or register bits. If the malfunction moves to the new location, the trouble is in the exchanged module. If the malfunction has not moved, the trouble is likely to be elsewhere, probably in the logic that feeds the module. Regardless of the outcome of the swap the good module should be returned to its original location – it is a desirable maintenance practice to keep each module in a specific place in the system.

Panel indicators may be used to detect malfunctions by following machine state changes. Using REPEAT or REPEAT BYPASS and the SPEED controls, a program loop can be run at slow speed by pausing after each instruction, memory access, AR subroutine, or even every step in a shift-count. A complete discussion of single step operation is presented in 9.1d.

Repetitive memory access can be produced without using any instructions by repeating an examine or deposit operation from the console. A single instruction may be repeated by setting it in the DATA switches and using REPEAT with the EXECUTE key. A single instruction in memory can be repeated by using START or READ IN. Complete information on the repetition of key functions is given in the final paragraph in 5.1.

Signal tracing requires a scope. With the machine running in an exercise loop, synchronize the oscilloscope sweep to one of the main sequence time pulses or to a pulse that should generate the suspect pulse or event. (The flow charts are very helpful for selecting an appropriate pulse. All pulses and levels are available at module output pins; all pin locations are shown on the logic drawings, and every signal name is prefixed by a mnemonic indicating the source logic for it.) In order that events may be viewed on the scope, the exercise loop should not halt the machine when an error occurs (to change a diagnostic loop into an exercise loop, replace the skip with a jump). It is difficult to pinpoint intermittents using signal tracing, so an effort should be made to change intermittent faults into catastrophic ones which can be located far more easily. When constructing an exercise loop keep it simple and use a minimum of instructions. For example, a simple loop can be obtained by starting the processor at a memory location that contains a jump to itself, or by repeating key functions.

Once the malfunction has been isolated and is presumably corrected, be particularly careful during validation to ensure that the module or component replaced was really at fault, and thus that the malfunction was actually cured by the replacement.

b Repair

Once a malfunctioning module has been located, repair should be made immediately by substitution, as intensive system scheduling requirements preclude use of the processor to aid in submodular troubleshooting. It is expected that defective modules will be returned to DEC for repair or replacement; those who wish to undertake module repair may refer to procedures given in PDP-6 Circuits. When replacing a module, make sure the new one is properly prepared to replace the old: in some cases adjustments must be made or internal connections must be made or broken.

Internal Connections

Most inverter, diode, capacitor-diode, and decoder modules have provision for internal jumpering to connect clamped loads at the output inverter collectors. These modules are shipped with all clamped loads connected; to prepare a replacement module for insertion in any given location the unused jumpers must be disconnected (e.g., with wire cutters). Some modules have jumpers for other purposes; for example some flip-flops, decoders, etc., have jumpers to select the connections to the module pins, and some delay lines require selection of the tap to choose the delay duration. Whether jumpers must be cut or soldered in depends upon the type of module. The utilization module lists (UML, see Appendix 1) give the jumper configurations in all locations for modules in which jumpers may be used. The jumper codes as they appear on the UML depend upon the type of module and the way in which the jumpers are used; these codes are explained in the circuit manual.

Delays

Most delay modules contain distributed-constant delay lines that cannot be adjusted. There are, however, three types of adjustable delays, 1304, 4301, and 4303. Whenever one of these modules is replaced, the new module must be adjusted to give the correct time interval. The 1304 and 4301 may be adjusted as follows:

1. Set up an exercise loop that repeatedly triggers the delay.

2. Set up an oscilloscope to observe the duration of the level output at pin J of the module (−3 volts during the delay interval, ground otherwise).

3. Set the scope sweep to the calibrated position, and select a per-centimeter sweep that displays the entire duration of the level output. The output is adjusted to the required duration by means of a screwdriver trimpot accessible through a hole in the rear of the module frame.

The 4303 has flip-flop type outputs. Use the above method but connect the scope at pin W to observe the negative output, or at pin U to observe the ground output. The delays for power clear and punch motor turnoff are not critical, so the replacement may not need adjustment at all; the delay period for detecting a nonexistent memory should not exceed 100 μsec. The 4303 in the repeat logic isadjusted by means of the speed controls: the selector chooses one of five external capacitors or none at all (the smallest is internal); the potentiometer varies the available charge current for the selected capacitor.

Tape Reader

When replacing any reader circuit card having a trimpot adjustment, perform the entire 1000-hour PM checkout (and all checks subordinate to it). This action not only provides for the required card adjustment but also serves as a validation procedure for the repair.

Card Reader

Replacing any photoamplifier package in the card reader requires complete readjustment of the exciter lamp brilliance and of the strobe pulse coincidence, using the procedures of Sections 3.16 and 3.17 of the card reader manual. Replacement of any mechanical part that involves an adjustment requires performance of the adjustment procedure as part of the reassembly; the replacement procedures of Section 4 in the reader manual should be followed by appropriate calibration procedures from Section 3.

c Validation and Log Entry

Following replacement of any electrical component in the system, tests should be performed to ensure correction of the fault condition and to trim up final adjustments or signal levels affected by the replacement. Validation normally requires performance of the PM procedure most applicable to that portion of the system in which the fault occurred. If repair or replacement is made in an area not normally checked during preventive maintenance, the appropriate diagnostic program (MAINDEC) should be run, or an alternate operational test should be devised to ensure proper operation. It is strongly suggested that the entire preventive maintenance task be performed for the malfunctioning system component. This action ensures detection of faults that may have been masked by the just-corrected malfunction and may reveal a failure mechanism, perhaps still extant elsewhere, that is likely to cause recurrence of the fault. If the entire PM procedure can be performed while the equipment is down and available, it need not be scheduled for the whole PM interval.

Corrective maintenance activities are not complete until they are recorded in detail in the maintenance log. Include all data indicating symptoms given by the fault, the method of fault detection, the component at fault, and any comments that might be helpful in future diagnosis. Suggest improvements in PM procedures if indicated. A complete and detailed write-up of diagnostic procedures used to locate a real dog is particularly important. Although some details may seem insignificant and obvious, they should nevertheless be included since examination of the log will be a major part of the initial investigation for a future malfunction. Reduce down time by recording everything.

APPENDIX 1

ENGINEERING DRAWINGS

Reduced copies of engineering drawings are included in Volume 2 of this manual. Chapter 4 explains the drawing numbers and type codes that identify engineering drawings for all components in a PDP-6 system and details the notation and conventions used in the block schematics and flow charts, which are the basis for learning and maintaining the equipment. However, there are many other engineering drawings used primarily for reference in maintenance. For a given system component, every drawing number has the type number of that component following the size letter; drawings that apply to the entire system are identified by "6" as the type number. In the following discussion all drawings mentioned are D size unless otherwise specified; the type of drawing in each case is identified by the code, and individual drawings are specified by the number or mnemonic code that follows the drawing serial number.

The master drawing list (MDL) for the entire system is an A-6 drawing with no individual drawing number. It lists the MDLs for all system components; these are also A size and identified only by the equipment type number and drawing serial number. Each MDL lists all drawings for the system component by title, number, code, revision letter, and number of sheets. The only other drawing for the entire system is a module list ML; in this instance the system module list SML, which lists the module requirements for all system components (this drawing is discussed further in Appendix 2).

The 166 MDL lists first the SD, FDs, and BSs that accompany this manual. Detailed information about the modules that make up the processor is shown in the utilization module lists UML, of which there are ten, numbered 14 to 23. Each UML shows three mounting panels, each divided into 25 sections representing the plug-in locations. Above each location is the type number and jumper code for the module occupying it. Each location is further partitioned according to the individual circuits (flip-flops, pulse amplifiers, inverters) in the module. Circuits are identified by logical function, using the same signal names that appear on the block schematics. For example, a pulse amplifier is labeled by naming the output pulse; an inverter or diode gate, by the output of the net in which it is used.

Of the various cable diagrams, CD, the one that shows the logical signals on the in-out and memory busses is included in the manual (IOMB, Figure 7-10). Another CD that is closely associated with the logic is CONS, which shows the connections to all operating keys and switches on the console. This drawing shows the generation of logic levels from the keys and switches and the connections to the main power control including the 60-cycle signal for the clock flag. Note how the on position of the key-locked console disabling switch holds system power on but disrupts the console power that enables the keys and switches. Note also that the NO and NC labels on switch contacts bear no invariant relationship to the normal position of the switch; these are merely the labels molded onto the switch case for physical identification of connections.

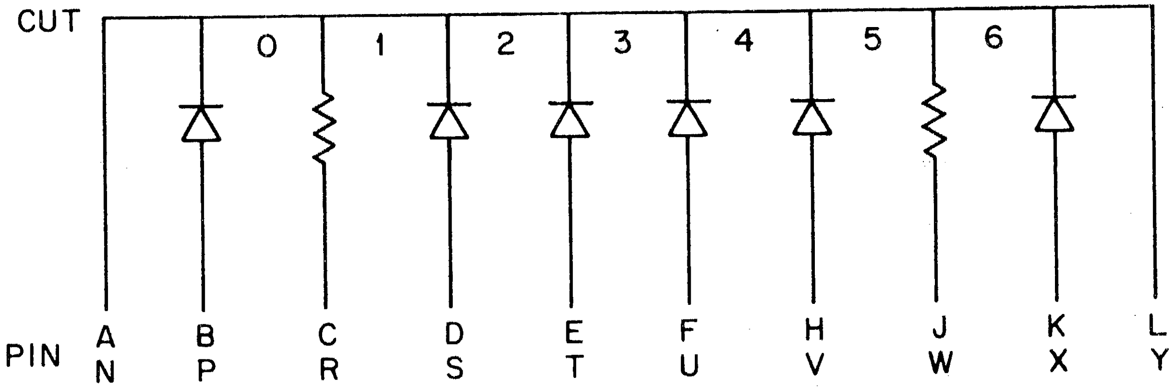
A pair of drawings CBL-2 and CBL-3 show all levels from the logic to indicators and switch registers on the console and on the bay indicator panels. CBL-1 shows the locations of all plugs and jacks throughout the four bays; CH shows the configuration of the main cable harness and its many branches. There are also a large number of cable lists CL that show the connections made from cables to module connector pins and identify the signals carried on the various cable lines.

All connections made among the module connector pins are shown in a series of wiring diagrams WD. Each drawing shows three mounting panels and is identified by the panel letters. Because of the complexity of the wiring in the processor, each set of three panels is shown in two drawings, one showing for example the grounds and pulses, the other the levels and flip-flops outputs (there are also other combinations).

Power wiring is shown in the PW drawings, ACPW and DCPW. Each shows the location of power controls and power supplies on the inside of the rear plenum doors viewed from the front, i.e., as though one were looking right through the mounting panels from the wiring side. ACPW shows all ac connections from the power controls and fans (for machines using 240 volts, 50 cycles, drawing 50ACPW is substituted). The interlock circuit in the lower right is associated with the plenum doors in bays 3 and 4 and may be used for example in the high-voltage line to a display mounted in the console. The interlock switches have three positions including an override the technician may use during maintenance. DCPW shows all of the dc wiring from the supplies to the terminal strips. All wiring is color coded +10 red, −15 blue, ground black. As shown in the lower left, all terminal strips and the terminals on the 728 and 734 Supplies use the same color code as the wiring; the terminals on the 778 Supply are different but the wires connected to them use the standard coding. The strip between the pair of representations for each bay is a power connector bracket, whose layout and wiring are shown at the lower left and right. These are mounted horizontally about halfway up the sides of the bays. The smaller horizontal strips in the right block for each bay are actually vertical and located at one end of every mounting panel.' The green and yellow terminals and wires carry the positive and negative variable voltages for marginal check (the wires are color coded identically). In the upper right are the connections to the marginal-check control panel and the marginal-check bus. This bus carries the variable voltages for marginal checking to other equipment at the site and also carries the −15-volt turnon signal for the memories. A wiring diagram, MCLT, shows the connections from the controls on the marginal-check panel to the variable power supply at the rear of the same bay. Connections to the elapsed time meter are at the right.

Many processor drawings, such as the PWs, also present information about the in-out control units included in this manual. However, each device has its own UML, WD, and various other cable drawings as listed in the appropriate MDL. For the card reader there are a number of Burroughs logic schematics not included in the card reader manual. These include drawings H-1182627 to H-1182640, H-11900107, H-11900115, H-11877446, H-11877453, and H-11877461. The last three are lists of equations for the reader logic; the others are computer-printed block schematics. These schematics show signal flow and give pin numbers and part designations of electrical components, plug-in units, and connectors, all of which are shown by rectangles. Wires are indicated by dashed lines; a "0" at an intersection point of two wires represents a common connection. The information is laid out on a rectangular coordinate system, with all inputs to the drawing in the input column at the left, all outputs at the right. The columns between are numbered and all horizontal rows are identified by a letter and a number. All input signals are identified by at least two rows of information, the top row giving the signal name, the bottom row indicating its source by page and row. Supplementary data may be included between the top and bottom rows. Output information is identified in the same way, giving the destination of the signal by page and row.

All boxes are eight character positions wide. The first two positions are used exclusively for the pin connections of input lines, and in all but the bottom row, the final two positions are used for pin connections of output lines. At the top center of each box is a part designation, and the bottom row gives the location. Section 2.2 of the card reader manual explains the system used to designate the locations of all electrical parts. The location of a plug-in unit is given in the form AAB5L6, which means that the unit is in rack A, panel A, row B, position 5 and that its key pin (the one larger than the others) plugs into row L, column 6 of the socket. The part designation is usually the initial letters of the words describing the part or some standard symbol such as K for relay; some of the part designations are given in Burroughs Section 2.5, and wherever the meaning of the part designation cannot be deduced, the location designation can be used to find and thus identify the part. If the box represents a pluggable unit, such as a photo amplifier, low-speed switch, or diode stick, its location also appears in Table 2.5-1. A schematic and description of the part can be found in Chapter 6. The majority of the plug-in units are diode sticks which are not described, but are all identical and are wired as follows:



A diode stick is designated in the drawings by D followed by a letter or number. It may be cut to form two or three diode gates: the top, center, or bottom part of the stick is represented by the box if a T, C, or B follows the D. A number indicates the position of the cut or cuts made (no cut is indicated by NC). Replacements are sent uncut, and the technician must cut them to conform with the configurations required for any given location. In addition to pin connections, the remaining rows may contain other information. Row 2 often indicates the part of the logic the circuit is used in. For a diode stick, rows 3 and 6 give information about the resistors at positions 1 and 6 and the gates associated with them: an R in position 3 indicates that the resistor is tied to −12 volts; a numeral in position 4 indicates the number of diodes used in the gate; A or O in position 5 or 6 indicates the use of the gate as logical AND or OR. Information contained in other rows depends upon the element represented; for example in a box representing a relay, NO or NC in row 5 indicates normally open or normally closed.

APPENDIX 2

SPARES

For a large-scale system such as PDP-6, even moderate downtime is generally far more expensive than the cost of maintaining a fully adequate stock of spares. The system module list SML lists the quantities of all modules, power supplies and power controls used in all system components (i.e., processors, memories, in-out control units) available in PDP-6. However, individual systems may differ considerably in quantity and type of system components, so DEC Field Service supplies each user with a list geared to his particular system. This list gives the total quantity and number of spares recommended for every module, power supply, and power control. Field Service also supplies spares lists of electrical components needed for repair of DEC modules and lists of electrical and mechanical spares for the various in-out devices. It is further recommended that the user keep a spare tape reader, tape punch, and keyboard-printer, and stock the following miscellaneous items for the system.

Quantity	Item	Part Number	Vendor
1	Switch for bat-handle key	Telever Switch 16006 Cat. No. S-302 Reworked per DEC drawing MA-C-01510	Switchcraft Inc.
5	Subminiature toggle switch SPDT	6AT1	Micro Switch
1	Lockout switch DPDT	1575-L	Arrow Hart
2	Rotron fan	53E168 Type CFG	Rotron Mfg. Co.
2	Rotron filter	34-X1431	Rotron Mfg. Co.
10	Indicator lamp	MC48-639B	Transistor Electric
1	Howard fan	12-80-15	Howard Industries

APPENDIX 3

GLOSSARY

It is not intended that use of this glossary shall be substituted for reading Chapter 4 or for using the flow charts when following any sequence of events in the logic drawings. This glossary gives the meanings of all prefix codes and all terms used in the signal names for the processor and the four in-out devices. Only a few complete signal names are included because all names are composites of standard terms, and their meanings are usually quite obvious to anyone who knows the meanings of the terms and has read 4.2. No generating conditions are given for logical functions because these are listed in the flow charts. The processor flags are listed and described in 2.1c, and all inputs to the logic from the console are discussed in 3.1a. With each prefix code below are listed in parentheses the numbers of the figures to which the code applies.

A (5-3)	Address cycle.
AC	Accumulator.
AC0	Accumulator whose address is 0.
AC2	Accumulator following the one addressed by an instruction.
ACBM (5-9)	Accumulator bit modify; the logical compare instructions, which test bits of AC as specified by a mask (either E or C(E)) and may modify the masked bits.
ACCP (5-9)	Accumulator compare, i.e., those arithmetic compare instructions that compare AC with either E or C(E).

ACCP ET AL TEST ACCP ETC COND	ACCP ETC COND is the skip or jump condition as specified by bits 7 and 8 in ACBM, ACCP, or MEMAC. Since bit 6 of the instruction code specifies whether the skip or jump is to be made on the presence or absence of the condition specified by bits 7 and 8, the condition signal is exclusive ORed with IR6 to generate ACCP ET AL TEST, which is sensed by the program control logic.
ACK	Acknowledge.
ADDR	Address.
A LONG	Flip-flop set whenever any operations are performed in the calculation of an address.
AOBJP	Add one to both and jump if positive.
AOBJN	Add one to both and jump if negative.
AR (6-4, 5, 6, 7, 8, 9, 10)	Arithmetic register and associated control logic, special inputs, subroutines, and flags.
AR COM CONT	AR complement control flip-flop; set at the beginning of subtraction or decrementing in AR to indicate that result must be complemented following the addition or incrementing used.
AR CRY COMP	AR carry complete.
AR CRY0	Flip-flop set by any carry out of AR0.

AR CRY1	Flip-flop set by any carry out of AR1.
AR = FP HALF	Level asserted when the fractional part of AR as a floating-point number is of magnitude 1/2.
AR OV SET	Condition that determines overflow in AS or MEMAC.
AR SBR	Level that causes the execute cycle to pause at ET4 for an AR subroutine.
AS (5-10)	Add-subtract.
ASH	Arithmetic shift.
ASHC	Arithmetic shift combined.
B	Binary; buffered.
BLK	Block.
BLKI	Block in.
BLKO	Block out.
BLT (6-18)	Block transfer.
BLT DONE	Level that terminates a block transfer either permanently because the block is complete or temporarily to handle an interrupt.
BLT LAST	Inhibits AC storage when block transfer is complete.
BOOLE (5-10)	Boolean instructions.

BOTH

Mode in which result is deposited in both AC and memory.

CAO

Character add one (= IBP).

CBHL

Level to the card reader causing it to place the upper half of a binary column on the output lines.

C( )

Contents of.

C(E)

Contents of location specified by effective address.

CFAC (6-17)

Computer floating-arithmetic connection (subroutine interface).

CH (6-19, 20)

As a prefix, character operations (byte manipulation); also channel.

CHG

Change.

CH INC
CH INC OP
CH ~INC OP

These three levels control the first part in a character operation. CH INC is asserted during the first part of any instruction that calls for incrementing the pointer. The other two levels are mutually exclusive within the first part of a character operation. CH INC OP causes incrementing of the pointer; CH ~INC OP inhibits incrementing either because the instruction does not require it or the first part of an incrementing character operation is being repeated following a priority interrupt between the two parts.

CL	Clear.
CLK	Clock.
CLR	Clear.
CMC	Core memory control; prefix for control signals from memory (core or fast) over the bus.
COM	Complement.
COMP	Complete.
CONI	Conditions in.
CONO	Conditions out.
CONSO	Conditions in and skip if one.
CONSZ	Conditions in and skip if zero.
CONT	Control; continue.
CPA (8-5)	Processor I/O interface. This is the interface at the other end of the bus from IOT control; through it the processor, via IOT control and the bus, controls itself as a device.
CR (8-13, 14, 15)	Card reader.
CREL	Signal from the card reader that indicates a validity or read check error.
CRL	Signal from the card reader that indicates it is not ready for operation.

CRY	Carry.
CYC	Cycle.
DATAI	Data in.
DATAO	Data out.
DC (6-20)	Deposit character.
DCA	Dc adder.
DEP	Deposit.
DIR	Direct, i.e., C(E) rather than E.
DIV	Divide.
DN	Do nothing.
DPC	Deposit character (= DPB).
DPCI	Index pointer and deposit character (= IDPB).
DS (6-25, 26)	Divide subroutine.
E (5-5)	As a prefix, execute cycle; also effective address.
E LONG	Level that causes processor to use second half of execute cycle.
EOF	End of file.
EX (5-13)	As a prefix, executive mode; also examine.
EXCH	Exchange.

EXEC	Execute.
F (5-4)	As a prefix, fetch cycle; also flip-flop.
FA (6-22)	Floating add.
FAD	Floating add.
FC(C(ACLT))	Fetch the contents of the location addressed by the number in ACLT.
FC(C(ACRT))	Fetch the contents of the location addressed by the number in ACRT.
FD (6-22)	Floating divide.
FDV	Floating divide.
FE (6-14, 15)	Floating exponent.
FM (6-22)	Floating multiply.
FMC	Fast memory control.
FMP	Floating multiply.
FP (6-23)	As a prefix, floating-point exponent calculate subroutine; also floating-point instructions.
FP/CH	IR decoder output for the floating-point instructions and character operations.
FS (6-19)	Floating scale.
FSC	Floating scale.

FSB	Floating subtract.
FWT (5-9)	Full-word transfer.
HWT (5-9)	Half-word transfer.
I (5-3)	Instruction cycle.
ILL	Illegal.
ILLEG	Illegal.
INC	Increment.
INH	Inhibit.
INST	Instruction.
INT	Interrupt.
IO	In-Out.
IOB (8-1,2)	In-out bus; IOBi (i = 0, ..., 35) is data line i on the bus.
IOS (5-7)	In-out select.
IOT (8-1)	In-out transfer.
IOT A	IR decoder output for IOT (replaced by a UUO in user mode).
IR (5-7, 8)	Instruction register and associated decoding nets.
JFCL	Jump on flag and clear.
JP (5-10)	Jump and pushdown.

JRA	Jump and restore accumulator.
JRST	Jump and restore.
JRST A	IR decoder output for JRST (replaced by a UUO in user mode if a halt or PI dismiss).
JSA	Jump and save accumulator.
JSP	Jump and save program counter.
JSR	Jump to subroutine.
K (5-2)	Key cycle.
KEY (5-1, 2)	Key.
KEY MANUAL	Level generated when any initiating key is pressed.
LC (6-20)	Load character.
LDC	Load character (= LDP).
LDCI	Increment pointer and load character (= ILDP).
LSB	Least significant bit.
LSH	Logical shift.
LSHC	Logical shift combined.
LT	Left.
MA (7-1, 2, 3)	Memory address.
MAI (7-2)	Memory address interface.

MAJ	Majority gate.
MARK	Teletype signal representing a 1.
MAS	Address switch.
MA SW	Address switch.
MB (6-1, 2, 3, 4; 7-6)	Memory buffer and associated control logic, special inputs, and data interface with memory bus.
MC (7-8, 9)	Memory control.
MD	Multiply-divide.
MEM	Memory; as a mode, result is stored only in E.
MEMAC (5-10)	Memory and accumulator modification and test instructions; these test AC against zero for a jump, and may or may not increment AC, or test C(E) against zero for a skip, and may or may not increment C(E).
MI (7-7)	Memory indicators.
MISC BITS	Miscellaneous bits.
MOVN, M	Move negative or magnitude.
MOV, S	Move or move and swap.
MP (6-21)	Multiply.
MQ (6-7, 11,12, 13)	Multiplier quotient register and associated control logic and special inputs.

MQ36	Extra MQ bit for use in multiply.
MR (5-1, 2)	Master.
MR CLR	Pulse that prepares processor for each main sequence or console function.
MR START	Pulse that clears entire computer at power turnon or when IO RESET is pressed.
MS (6-24)	Multiply subroutine.
MSB	Most significant bit.
MUL	Multiply.
MULT	Multiply.
NEGATE	Form the arithmetic 2's complement.
NR (6-27)	Normalize return.
NXT	Next.
OP	Operation.
OV	Overflow.
PC (5-11, 12)	Program counter and control.
PDL OV	Pushdown list overflow.
PI (8-3, 4)	Priority interrupt.
PIA	Priority interrupt assignment.
PICH	PI channel address.

PIH	PI hold.
PIO	PI on.
PIR	PI request.
POP	Pullout.
POPJ	Pullout and jump.
PR (7-4)	Protection.
PSE	Pause.
PTP (8-9, 10)	Paper tape punch.
PTR (8-6, 7, 8)	Paper tape reader.
PUSH	Pushdown.
PUSHJ	Pushdown and jump.
PWR	Power.
PWR CLR	Pulses that clear computer at power turnon.
RD	Read.
REL	Relocation.
REM	Remainder; floating-point mode that stores low-order half of result in AC2.
REQ	Request.
RIM	Readin mode.
RLA (7-5)	Relocation adder.

RLR (7-5)	Relocation register.
ROT	Rotate.
ROTC	Rotate Combined.
RPT	Repeat.
RQ	Request.
RS	Restart.
RST	Restore.
RT	Right.
RUN (5-1)	Run.
S (5-6)	Store cycle.
SBR	Subroutine; also a mnemonic for subroutine card 1260.
SC (6-14, 15, 16)	Shift counter.
SEL	Select.
SH (6-20)	Shift.
SH AC2	Double-length shift.
SHC (6-7)	Shift connection.
SPACE	Teletype signal for 0.
SR	Shift register.
ST	Start.

STATUS	Any IOT instruction that examines status, i.e., CONI, CONSO or CONSZ.
STB	Strobe.
SW (5-1)	Switch.
SWAP	Interchange the left and right halves of a word.
SUB	Subtract.
T	Time.
TST	Test.
TTI (8-11, 12)	Teletype input.
TTO (8-11, 12)	Teletype output.
TTY (8-11, 12)	Teletype.
UUO (5-10)	Programmed operator (unused op code); a UUO is performed when one is programmed or when it replaces an illegal user instruction.
UUO A	IR decoder output for UUO.
WR	Write.
XCT (5-10)	Execute.
2XX	Level indicating that IR contains an octal code beginning with 2.
25X	Level indicating that IR contains an octal code beginning with 25.

APPENDIX 4

INSTRUCTION CODES

Octal	Mnemonic	Octal	Mnemonic	Octal	Mnemonic
000		172	FDVM	242	LSH
⋮	UUO	173	FDVB	243	
		174	FDVR	244	ASHC
077		175	FDVRL	245	ROTC
100		176	FDVRM	246	LSHC
⋮		177	FDVRB	247	
		200	MOVE	250	EXCH
131		201	MOVEI	251	BLT
132	FSC	202	MOVEM	252	AOBJP
133	IBP	203	MOVES	253	AOBJN
134	ILDB	204	MOVS	254	JRST
135	LDB	205	MOVSI	255	JFCL
136	IDPB	206	MOVSM	256	XCT
137	DPB	207	MOVSS	257	
140	FAD	210	MOVN	260	PUSHJ
141	FADL	211	MOVNI	261	PUSH
142	FADM	212	MOVNM	262	POP
143	FADB	213	MOVNS	263	POPJ
144	FADR	214	MOVM	264	JSR
145	FADRL	215	MOVMI	265	JSP
146	FADRM	216	MOVMM	266	JSA
147	FADRB	217	MOVMS	267	JRA
150	FSB	220	IMUL	270	ADD
151	FSBL	221	IMULI	271	ADDI
152	FSBM	222	IMULM	272	ADDM
153	FSBB	223	IMULB	273	ADDB
154	FSBR	224	MUL	274	SUB
155	FSBRL	225	MULI	275	SUBI
156	FSBRM	226	MULM	276	SUBM
157	FSBRB	227	MULB	277	SUBB
160	FMP	230	IDIV	300	CAI
161	FMPL	231	IDIVI	301	CAIL
162	FMPM	232	IDIVM	302	CAIE
163	FMPB	233	IDIVB	303	CAILE
164	FMPR	234	DIV	304	CAIA
165	FMPRL	235	DIVI	305	CAIGE
166	FMPRM	236	DIVM	306	CAIN
167	FMPRB	237	DIVB	307	CAIG
170	FDV	240	ASH	310	CAM
171	FDVL	241	ROT	311	CAML

Octal	Mnemonic	Octal	Mnemonic	Octal	Mnemonic
312	CAME	367	SOJG	444	EQV
313	CAMLE	370	SOS	445	EQVI
314	CAMA	371	SOSL	446	EQVM
315	CAMGE	372	SOSE	447	EQVB
316	CAMN	373	SOSLE	450	SETCA
317	CAMG	374	SOSA	451	SETCAI
320	JUMP	375	SOSGE	452	SETCAM
321	JUMPL	376	SOSN	453	SETCAB
322	JUMPE	377	SOSG	454	ORCA
323	JUMPLE	400	SETZ	455	ORCAI
324	JUMPA	401	SETZI	456	ORCAM
325	JUMPGE	402	SETZM	457	ORCAB
326	JUMPN	403	SETZB	460	SETCM
327	JUMPG	404	AND	461	SETCMI
330	SKIP	405	ANDI	462	SETCMM
331	SKIPL	406	ANDM	463	SETCMB
332	SKIPE	407	ANDB	464	ORCM
333	SKIPLE	410	ANDCA	465	ORCMI
334	SKIPA	411	ANDCAI	466	ORCMM
335	SKIPGE	412	ANDCAM	467	ORCMB
336	SKIPN	413	ANDCAB	470	ORCB
337	SKIPG	414	SETM	471	ORCBI
340	AOJ	415	SETMI	472	ORCBM
341	AOJL	416	SETMM	473	ORCBB
342	AOJE	417	SETMB	474	SETO
343	AOJLE	420	ANDCM	475	SETOI
344	AOJA	421	ANDCMI	476	SETOM
345	AOJGE	422	ANDCMM	477	SETOB
346	AOJN	423	ANDCMB	500	HLL
347	AOJG	424	SETA	501	HLLI
350	AOS	425	SETAI	502	HLLM
351	AOSL	426	SETAM	503	HLLS
352	AOSE	427	SETAB	504	HRL
353	AOSLE	430	XOR	505	HRLI
354	AOSA	431	XORI	506	HRLM
355	AOSGE	432	XORM	507	HRLS
356	AOSN	433	XORB	510	HLLZ
357	AOSG	434	IOR	511	HLLZI
360	SOJ	435	IORI	512	HLLZM
361	SOJL	436	IORM	513	HLLZS
362	SOJE	437	IORB	514	HRLZ
363	SOJLE	440	ANDCB	515	HRLZI
364	SOJA	441	ANDCBI	516	HRLZM
365	SOJGE	442	ANDCBM	517	HRLZS
366	SOJN	443	ANDCBB	520	HLLO

Octal	Mnemonic
521	HLLOI
522	HLLOM
523	HLLOS
524	HRLO
525	HRLOI
526	HRLOM
527	HRLOS
530	HLLE
531	HLLEI
532	HLLEM
533	HLLES
534	HRLE
535	HRLEI
536	HRLEM
537	HRLES
540	HRR
541	HRRI
542	HRRM
543	HRRS
544	HLR
545	HLRI
546	HLRM
547	HLRS
550	HRRZ
551	HRRZI
552	HRRZM
553	HRRZS
554	HLRZ
555	HLRZI
556	HLRZM
557	HLRZS
560	HRRO
561	HRROI
562	HRROM
563	HRROS
564	HLRO
565	HLROI
566	HLROM
567	HLROS
570	HRRE
571	HRREI
572	HRREM
573	HRRES
574	HLRE
575	HLREI
576	HLREM
577	HLRES
600	TRN
601	TLN
602	TRNE
603	TLNE
604	TRNA
605	TLNA
606	TRNN
607	TLNN
610	TDN
611	TSN
612	TDNE
613	TSNE
614	TDNA
615	TSNA
616	TDNN
617	TSNN
620	TRZ
621	TLZ
622	TRZE
623	TLZE
624	TRZA
625	TLZA
626	TRZN
627	TLZN
630	TDZ
631	TSZ
632	TDZE
633	TSZE
634	TDZA
635	TSZA
636	TDZN
637	TSZN
640	TRC
641	TLC
642	TRCE
643	TLCE
644	TRCA
645	TLCA
646	TRCN
647	TLCN
650	TDC
651	TSC
652	TDCE
653	TSCE
654	TDCA
655	TSCA
656	TDCN
657	TSCN
660	TRO
661	TLO
662	TROE
663	TLOE
664	TROA
665	TLOA
666	TRON
667	TLON
670	TDO
671	TSO
672	TDOE
673	TSOE
674	TDOA
675	TSOA
676	TDON
677	TSON
7-00	BLKI
7-04	DATAI
7-10	BLKO
7-14	DATAO
7-20	CONO
7-24	CONI
7-30	CONSZ
7-34	CONSO

APPENDIX 5

TELETYPE CODE

The 8-bit codes are listed below. An asterisk indicates a code that has no effect on the Model 35. Alternate characters are listed in parentheses. The characters actually contain only seven information bits. The eighth bit may be used for parity, but currently all machines are set up so that the eighth bit is a mark, and thus the codes generated from the keyboard are 200_8 greater than the corresponding ASCII codes.

Octal Code	ASCII Character	Key Combination	Remarks
200	NULL	SHIFT CTRL P	Null.
201*	SOM	CTRL A	Start of message.
202*	EOA	CTRL B	End of address.
203*	EOM	CTRL C	End of message.
204	EOT	CTRL EOT	End of transmission; shuts off TWX machines.
205	WRU	CTRL WRU	"Who are you?" Triggers "Here is...," at remote station.
206*	RU	CTRL RU	"Are you...?"
207	BELL	CTRL BELL	Rings the bell.
210*	FE	CTRL H	Format effector.
211	HT	CTRL TAB	Horizontal tab.
212	LF	LINE FEED	Line feed.
213	V TAB	CTRL VT	Vertical tab.
214	FF	CTRL FORM	Form feed.
215	CR	RETURN	Carriage return.
216*	SO	CTRL N	Shift out.
217*	SI	CTRL O	Shift in.
220*	DC0	CTRL P	Device control reserved for data linc escape.

Octal Code	ASCII Character	Key Combination	Remarks
221	DC1	CTRL Q	Turns reader on.
222*	DC2	CTRL TAPE	Turns punch on.
223	DC3	CTRL XOFF	Turns reader off.
224*	DC4	CTRL ~~TAPE~~	Turns punch off.
225*	ERR	CTRL U	Error.
226*	SYNC	CTRL V	Synchronous idle.
227*	LEM	CTRL W	Logical end of media.
230*	S0	CTRL X	Separator, information.
231*	S1	CTRL Y	Separator, data delimiter.
232*	S2	CTRL Z	Separator, words.
233*	S3	SHIFT CTRL K	Separator, groups.
234*	S4	SHIFT CTRL L	Separator, records.
235*	S5	SHIFT CTRL M	Separator, files.
236*	S6	SHIFT CTRL N	Separator, miscellaneous.
237*	S7	SHIFT CTRL O	Separator, miscellaneous.
240	Space	Space bar	
241	!	SHIFT !	
242	"	SHIFT "	
243	#	SHIFT #	
244	$	SHIFT $	
245	%	SHIFT %	
246	&	SHIFT &	
247	´(')	SHIFT ´(')	
250	(	SHIFT (	
251	)	SHIFT)	
252	*	SHIFT *	
253	+	SHIFT +	
254	,	,	
255	-	-	
256	.	.	

Octal Code	ASCII Character	Key Combination	Remarks
257	/	/	
260	Ø	0	Zero, prints with a slash
261	1	1	
262	2	2	
263	3	3	
264	4	4	
265	5	5	
266	6	6	
267	7	7	
270	8	8	
271	9	9	
272	:	:	
273	;	;	
274	<	SHIFT <	
275	=	SHIFT =	
276	>	SHIFT >	
277	?	SHIFT ?	
300	\(@)	SHIFT \ (@)	
301	A	A	
302	B	B	
303	C	C	
304	D	D	
305	E	E	
306	F	F	
307	G	G	
310	H	H	
311	I	I	
312	J	J	
313	K	K	
314	L	L	

Octal Code	ASCII Character	Key Combination	Remarks
315	M	M	
316	N	N	
317	O	O	
320	P	P	
321	Q	Q	
322	R	R	
323	S	S	
324	T	T	
325	U	U	
326	V	V	
327	W	W	
330	X	X	
331	Y	Y	
332	Z	Z	
333	[	SHIFT K	
334	¬ (\)	SHIFT L	
335	]	SHIFT M	
336	∧ (↑)	SHIFT ∧ (↑)	
337	_ (←)	SHIFT _ (←)	
340-373*			Lower case letters; codes cannot be generated from keyboard and should not be used in programs for reasons of compatability.
374*	ACK		Acknowledge; code cannot be generated from keyboard and should not be used in programs for reasons of compatability.
375*	①	ALT MODE	May be used for any desired control purpose.
376*	ESC		Escape; code cannot be generated from keyboard and should not be used in programs for reasons of compatability.

Octal Code	ASCII Character	Key Combination	Remarks
377*	DEL	RUB OUT	Delete
		REPT	Causes any other key that is struck to repeat continuously until REPT is released.
		LOC LF	Local line feed.
		LOC CR	Local carriage return.
		BRK RLS	Not connected.

APPENDIX 6

CARD READER CODE

6-bit Code	Character	Column Punch	6-bit Code	Character	Column Punch
00		Any invalid	40	-	11
01	1	1	41	J	11 1
02	2	2	42	K	11 2
03	3	3	43	L	11 3
04	4	4	44	M	11 4
05	5	5	45	N	11 5
06	6	6	46	O	11 6
07	7	7	47	P	11 7
10	8	8	50	Q	11 8
11	9	9	51	R	11 9
12	0	0	52		11 0
13	= [#]	8 3	53	$	11 8 3
14	' [@]	8 4	54	*	11 8 4
15		8 5	55		11 8 5
16		8 6	56		11 8 6
17		8 7	57		11 8 7
20	Space	None	60	+ [&]	12
21	/	0 1	61	A	12 1
22	S	0 2	62	B	12 2
23	T	0 3	63	C	12 3
24	U	0 4	64	D	12 4
25	V	0 5	65	E	12 5
26	W	0 6	66	F	12 6
27	X	0 7	67	G	12 7
30	Y	0 8	70	H	12 8
31	Z	0 9	71	I	12 9
32		0 8 2	72		12 0
33	,	0 8 3	73	.	12 8 3
34	([%]	0 8 4	74	) [⌑]	12 8 4
35		0 8 5	75		12 8 5
36		0 8 6	76		12 8 6
37		0 8 7	77		12 8 7

Invalid Punch Combinations

0 1 11
0 2 11
0 3 11
0 4 11
0 5 11
0 6 11
0 7 11
0 8 11
0 9 11

0 1 12
0 2 12
0 3 12
0 4 12
0 5 12
0 6 12
0 7 12
0 8 12
0 9 12

1 2
1 3
1 4
1 5
1 6
1 7
1 8
1 9

2 3
2 4
2 5
2 6
2 7
2 8*
2 9
3 4

3 5
3 6
3 7
3 9
4 5
4 6
4 7
4 9

5 6
5 7
5 9
6 7
6 9
7 9
8 9
11 12

*except 2 8 0

digital

EQUIPMENT
CORPORATION

MAYNARD, MASSACHUSETTS

Cambridge, Mass. • Washington, D. C. • Parsippany, N.J. • Rochester, N.Y. • Los Angeles
Palo Alto • Chicago • Ann Arbor • Pittsburgh
Denver • Huntsville • Orlando • Carleton Place
and Toronto, Ont. • Reading, England • Paris,
France • Munich, Germany • Sydney, Australia