Digital Equipment Corporation
Maynard, Massachusetts

digital

PDP-10
Maintenance Manual

KA10
CENTRAL PROCESSOR

Volume I

PDP-10
KA10 CENTRAL PROCESSOR
MAINTENANCE MANUAL
VOLUME I

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

TABLES (Cont)

ILLUSTRATIONS

This publication is one of a series covering the operation, theory, and maintenance of the Programmed Data Processor PDP-10, manufactured by Digital Equipment Corporation, Maynard, Massachusetts.

Because the PDP-10 System may take various forms, depending upon the options chosen, there is no single maintenance manual for the entire system.

This manual covers the KA10 Central Processor of the PDP-10 System. References to other manuals in the series are made when necessary, and duplication of content is avoided when possible.

Volume I of this manual contains the operating instructions, principles of operation, and maintenance instructions for the KA10 Central Processor. Volume II contains a set of engineering drawings, flow diagrams, logic diagrams, and other reference material, all of which supplements the text of this volume.

This manual assumes that the reader is familiar with the PDP-10 System as described in the PDP-10 System Reference Manual (DEC-10-HGAC-D); therefore, it does not cover the fundamentals of data processing systems' or what system instructions are but, rather, how they are implemented by the system hardware.

CHAPTER 1
GENERAL INFORMATION

The PDP-10 is a general purpose computer system consisting, typically, of a KA10 Central Processor (Figure 1-1), a memory, and peripheral equipment such as: a paper-tape reader and punch, teletypewriter, card reader, line printer, DECtape, disk file, and display. The KA10 Central Processor is the primary control unit for the PDP-10 System. The processor handles 36-bit words, which are stored in a memory, with a maximum capacity of 262,144 words.

1.1 DESCRIPTION

1.1.1 Physical Description

The central processor occupies three 31-in. DEC cabinets. Bay 3 (on the right) includes the paper-tape reader and punch and the operator's console. Two TU55 DECtape units may also be located in Bay 3. At the left of the reader and punch is the margin check and maintenance panel, which serves as the master control for system margin checks and as the control panel for performing maintenance checks on the central processor.

Logic wiring is accessible behind the hinged front doors of the cabinets. As many as 16 module panels, designated A through T (G, I, O, Q omitted), from top to bottom, can be mounted. Panels in the 31-in. cabinets hold 44 DEC FLIP CHIP plug-in modules (numbered from left to right, front view), while those in the 19-in. cabinets hold a maximum of 32. Local margin check switches are located beside each module panel. Power supplies are bolted to a hinged frame on the inside of the rear door. Fans or blowers within the cabinets provide the necessary heat removal.

1.1.2 System Configurations

The PDP-10 has an extensive variety of available peripherals. Standard peripherals, the paper tape reader and punch, and the console teleprinter are provided upon request of the user. Table 1-1 lists the hardware options packaged in the central processor. Table 1-2 lists the standard peripherals furnished with the central processor.

Although there are five basic PDP-10 systems (PDP-10/10 through PDP-10/50), the flexibility afforded by hardware/software interaction and the variety of optional devices precludes identifying standard equipment configurations.

Table 1-1
Central Processor Options

Designation	Option
KE10	Extended Order Code
KT10 or KT10A	Memory Protection and Relocation
	Double Memory Protection and Relocation
KM10	Fast Registers

Table 1-2
Peripherals Furnished with Central Processor

Model	Name
KSR35 (LT35A) or KSR37 (LT37A)	Teleprinter (10 characters/second)
	Teleprinter (15 characters/second)
DEC Type PC09 (Modified)	Paper Tape Reader (300 character/second)
	Paper Tape Punch (50 characters/second)

1.2 CHARACTERISTICS

The operating characteristics of the PDP-10 are as follows:

Word length:	36 bits
Core Memory Capacity:	Expandable to 262,144 words
Accumulators:	16
Index Registers:	15
I/O Capacity:	7 priority interrupt channels; 128 I/O device assignments

1.3 INSTALLATION

The PDP-10 System incorporating the KA10 Central Processor is installed by DEC field service personnel; therefore, installation procedures are not covered in this manual. The PDP-10 Site Preparation Guide can be used to plan the installation site. Installation procedures for the various system options are provided in the respective maintenance manuals.

1.4 RELATED DOCUMENTS

The following documents either supplement the information contained in this manual or are prerequisite to understanding the material presented.

Document	Title
DEC-10-HGAC-D	PDP-10 System Reference Manual
DEC-10-HIFB-D	PDP-10 Interface Manual
DEC-10-HIGA-D	MA10 Core Memory Maintenance Manual
DEC-10-BIBA-D	MB10 Core Memory Maintenance Manual

Document	Title
DEC-00-HZTA-D	TU55 DECtape 55 Instruction Manual
DEC-08-I2BA-D	PC02 Paper-Tape Reader Instruction Manual
DEC-9/L-H2AA-D	PC09C High Speed Paper Tape Instruction Manual
Royal McBee Corp	Model 500 Paper Tape Punch Maintenance Manual
Bulletin 281B (Teletype Corp)	KSR35 Teletype Technical Manual
Bulletin 333 (Teletype Corp)	KSR35 Teletype Parts Manual
Bulletin (Teletype Corp)	KSR37 Teletype Technical Manual
Bulletin (Teletype Corp)	KSR37 Teletype Parts Manual
Bulletin 273B (Teletype Corp)	KSR33 Teletype Technical Manual (Volume I and II)
Bulletin 1184B (Teletype Corp)	KSR33 Teletype Parts Manual
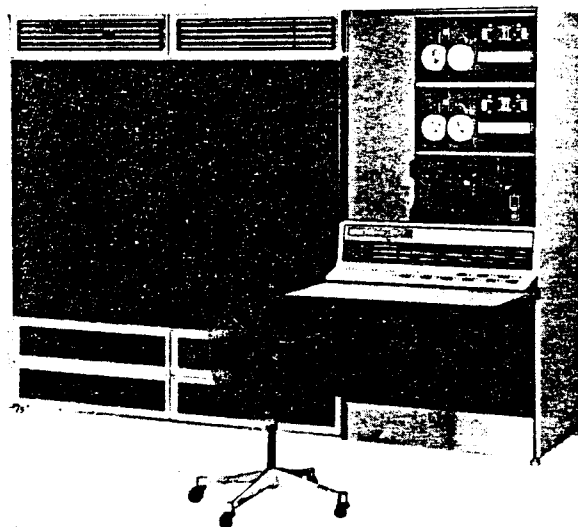
Figure 1-1   PDP-10 General Purpose Computer

This manual is primarily concerned with the KA10 Central Processor. However, a description of basic system operation is provided so that the reader can more clearly see how the KA10 operation relates to the system.

The major components of any PDP-10 System are the arithmetic processor, core memory, and input/output (I/O) subsystems. A functional configuration of system elements, representing a typical basic installation, is presented in Figure 2-1.

As the name implies, the arithmetic processor performs system arithmetic and logical data manipulations. It connects to the memory and I/O subsystems through buses. Standard I/O devices furnished with

the processor are the paper tape reader/punch and the console teleprinter. Optional processor hardware comprises the memory protection and relocation, extended order code, and fast memory options.

The core memory is the basic memory element. It is accessed directly by the processor, and can be accessed via a data channel by high-speed devices such as disk files. It is made up of separate modules, but appears to the processor as one homogeneous unit. Memory functions include recognizing addresses, reading and writing data, and putting appropriate control information and data on the bus for the processor. Once a memory module has received its commands, it performs these functions independent of the
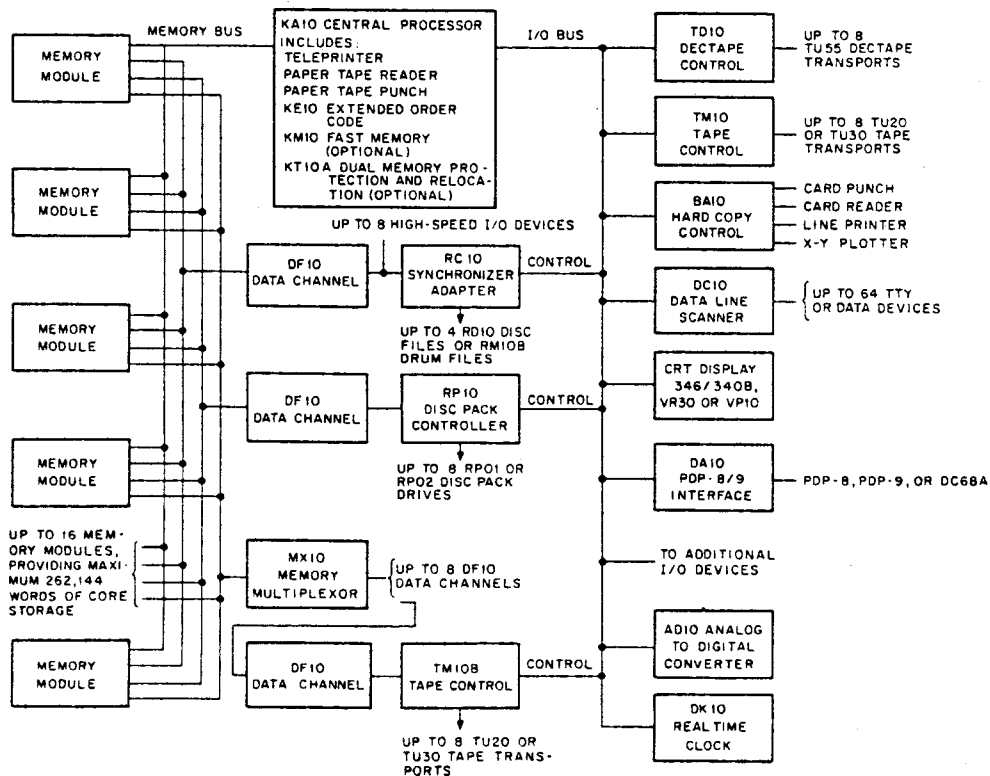
Figure 2-1 PDP-10 System Diagram

processor and of other modules. A number of different types of core memories are available for the PDP-10 System, and the details of operation, maintenance, and theory of each can be found in the appropriate manual.

2.1 ARITHMETIC PROCESSOR

The arithmetic processor performs all arithmetic and logical operations, controls all transfer of data to and from the input/output devices, and makes service requests to memory. It operates asynchronously using hardware subroutines, whereby the start of each operation is triggered by the completion of the previous operation rather than by a trigger from a synchronous timing chain.

Registers in the arithmetic processor differ considerably in size. Generally, data registers contain 36 bits; address registers, 18 bits; floating-point exponent registers, 9 bits; memory protection and relocation registers, 8 bits; and registers of the priority interrupt system, 7 bits.

The processor has 16 general-purpose registers that are implemented as the first 16 core memory locations or high-speed integrated circuit registers (fast memory option). In either configuration, the general-purpose registers may be used as accumulators, index registers, and/or normal memory locations. This functional interchange ability offers a number of advantages. For example, the contents of one register can be transferred to another with a single instruction by addressing one register as a memory location and the other as an accumulator.

When the general-purpose registers are to function as memory locations, the address is carried in the standard memory address field (bits 18 through 35) of the instruction word. When used as accumulators, they are addressed in the accumulator portion (bits 9 through 12) of the instruction word. When several operations are required in a computation, a number of consecutive operations can be performed and their results stored in the different accumulators. The contents of the accumulators, each representing a partial result, can then be combined quickly for the final answer to the computation.

When used as index registers, the general-purpose registers are addressed by the index field (bits 14-17) of the instruction word. The contents of the addressed register are added to the memory address portion of the instruction word before the operation specified in

the instruction word is carried out. Only 15 of the 16 general-purpose registers (addresses 1_8 through 17_8) can be used for indexing purposes. When a 0 appears in the index field, no indexing occurs.

In systems employing the fast memory option, 16 high-speed integrated circuit registers replace the first 16 core memory locations and are so addressed. When the option is included, the replaced core locations are not program-accessible. Instruction-operating times can be reduced by executing iterative program loops from consecutive fast memory locations.

Time-sharing systems employ optional memory protection and relocation hardware, which includes four 8-bit registers. Basically, this equipment stores constants used by the processor to prevent one program from infringing upon areas of memory assigned to another. The relocation hardware automatically reassigns user programs to available memory locations, rather than allowing them to be stored in the locations for which they are written. (Essentially all user programs are written to run from memory address 00000.) Consequently, a program written for locations 0 through 3777_8, for example, may be relocated to run from locations 12000_8 through 15777_8, or any other block of 4000_8 contiguous locations starting at an integral multiple of 2000_8. The protection hardware inhibits a user program from calling a memory location that exceeds the total number of assigned locations. These concepts of protection and relocation are discussed further in Paragraph 2.5, Programming.

The standard processor contains 366 wired instructions which include the optional extended-order code hardware for implementing byte and floating-point instructions. The optional hardware provides 35 floating-point and 5 byte instructions. Floating-point logic contains complete arithmetic capability, including two instructions for double precision calculations (that is, calculations involving numbers larger than 36 bits) and two for simplifying the conversion from fixed to floating point. Byte instructions operate on bytes of any size. Systems without the extended-order code option use programmed simulators of the byte and floating-point instructions to exactly simulate the unimplemented instructions.

2.2 MEMORY

Standard PDP-10 memory systems (Figure 2-2) are fabricated from 8K (8,192 words) or 16K (16,384 words) memory modules. The modules of a memory system connect to the associated central processor

(or other high-speed device) through a common memory bus. As many as sixteen 16K modules (262,144 memory locations) can be grouped into a single memory system.

Access to all storage locations in a module is available through each of four ports. A module can connect to, and constitute a part of, as many as four separate memory systems through these ports. Each system, of course, has a separate memory bus; however, the use of a memory bus and its associated memory system can be apportioned between as many as eight high-speed devices by the optional multiplexor equipment.

2.2.1 Memory Bus

The memory bus connects the central processor in parallel with each memory module on 72 lines (37 bidirectional, 3 memory-to-processor, and 32 processor-to-memory). Although a particular memory module may not use every bus line, all lines are strapped across the module interface and are available to subsequent modules on the bus. Module interfaces have two sets of bus connectors; one receives incoming lines, the other accommodates outgoing lines.

Twenty-eight of the memory bus lines carry address selection data. Eight of them carry the 1 and 0 states of the four most significant address bits. These lines enter the modules through local switches that are configured to assign a different address to each module. A module responds only to bus inputs that include its address. The remaining address bits specify one storage location in the selected module.

In addition to being used to select a memory location, the least significant address bit (both the 1 and 0 states) is applied to the memory modules through local switches which can be used to assign all odd and all even addresses to alternate modules in a memory network. Such an arrangement allows the processor to interleave memory cycles between alternate modules, thus reducing processor idle time, which results from waiting for a module to complete one cycle before requesting another cycle. In general, interleaving is possible over an entire memory network made up of an even number of modules. If a network has an odd number of modules, the last module cannot participate in the interleave scheme.

2.2.2 Memory Priority

Because a module may constitute a portion of as many as four discrete memory systems (Figure 2-2), every module has a priority network that grants access through only one of its ports at a time. Ports with the first and second priorities are always serviced in that order. The third- and fourth-priority ports are serviced in order also, except when they simultaneously request memory access. In this case, access is granted to the port (third and fourth) least recently serviced (assuming neither the first- or second-priority ports are requesting access).

When a port gains access to a memory module, the processor or other high-speed device on the memory bus dictates the type of cycle the module undergoes. After one cycle, the port loses its access rights and the priority scheme is re-examined to determine which memory system is granted the following memory cycle.

2.2.3 Memory Cycles

The three basic memory cycles are: read, write, and read-modify-write. The read cycle reads a word into the processor, then rewrites it into the original memory location. A write cycle clears the specified memory location, then writes data from the processor into that location. A read-modify-write cycle reads a memory word into the processor, pauses while the processor uses the word, then stores the result of the processing in the original memory location.

The processor is connected to memory only when necessary during a memory cycle. During a read cycle, for example, the processor is connected only long enough to access the memory word. It is then free to continue operations or access a location in a second memory module, while the first module restores the data to its original location. This facility for overlapping memory cycles, or overlapping a memory cycle with computation, contributes to overall access times shorter than the period of one memory cycle.

2.2.3.1 Read Cycle – When the processor wants to read data from memory, it places the address on the memory bus (Figure 2-3), and brings up the memory cycle request (REQ CYC) and read request (RD RQ) lines. If the addressed module is not already in use, and if a request is not being made through a higher-priority port, the module reads the address into its input buffer and acknowledges the request with an address acknowledge (ADR ACK) pulse. It then fetches the requested data and stores it in its data buffer.
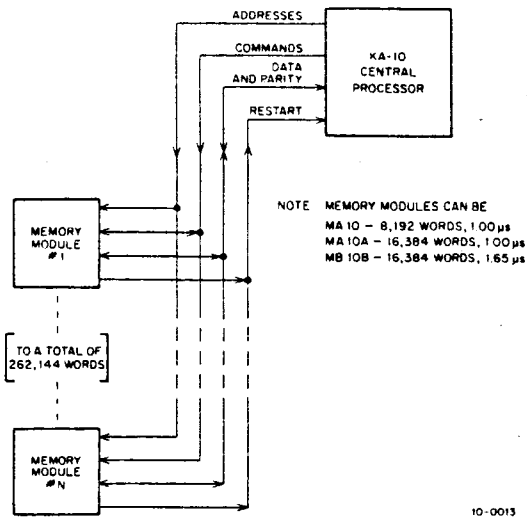
Figure 2-2 Memory System

The data is also sent down the bus, and the module
sends a read restart (RD RS) pulse to the processor.
Having performed the required functions, the module
disconnects itself from the bus and restores the data in
its data buffer to the original memory location.

2.2.3.2 Write Cycle

2.2.3.2 Write Cycle - To initiate a write cycle, the
processor places an address on the memory bus
(Figure 2-3) and brings up the REQ CYC and write re-
quest (WR RQ) lines. If the addressed module is not
busy, it responds with an ADR ACK signal and reads
the address into its buffer register. It then clears the
addressed location, during which time the processor
loads the data into the buffer register of the module.
Upon receipt of the data, the module disconnects it-
self from the bus and initiates the writing of the data
into the addressed location.

2.2.3.3 Read-Modify-Write Cycle

2.2.3.3 Read-Modify-Write Cycle - The processor
initiates a read-modify-write cycle (Figure 2-3) by
sending the address REQ CYC and both RD RQ and
WR RQ. If not otherwise employed, the memory mod-
ule performs a normal read cycle. Instead of discon-
necting itself from the bus and restoring the data in
the buffer register to its original location, the module
clears its buffer register and waits. When ready, the
processor sends new data to the module and restarts
memory action with a write restart (WR RS) pulse.
After receiving WR RS, the module disconnects from
the bus and writes the new data into the location from
which it read the old data.

2.2.4 Parity

Standard PDP-10 memory modules have 37 bits per
word, with one bit providing parity storage. They do
not have parity-inserting or checking circuits; odd
parity is inserted by the processor when writing, and
is checked by the processor when reading.
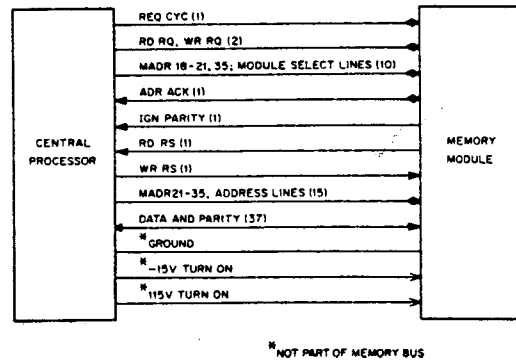
Figure 2-3 Memory Bus

Nonstandard memory modules without provisions for
parity storage, or standard modules which, for some
reason, want to inhibit the parity check function of
the processor, must accompany their ADR ACK pulses
with the IGN PARITY (ignore parity) signal.

2.2.5 Multiplexor

Devices accessing memory through a multiplexor share
a common data bus. The multiplexor assigns use of
the data bus to one device at a time on a priority
basis.

When a device requires memory access, it raises its
REQn signal (Figure 2-4). If the multiplexor is not
being used, and if a similar signal is not being sent
by a higher-priority device, the multiplexor returns
an ACKn signal. The acknowledged device now has
control of the multiplexor bus, and proceeds to com-
municate directly with memory through the associated
memory bus.

When the device places a request on the memory bus,
the multiplexor enters a mode in which it waits for
memory to respond. If the request addresses a non-
existent memory location, the multiplexor can wait

indefinitely for a response. To prevent such a condition, the transmitting device can relinquish control of the multiplexor by generating an MPX CLR (multiplexor clear) signal if memory does not respond within approximately 100 μs.
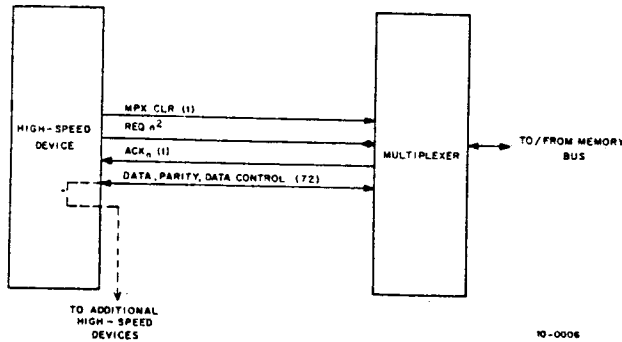
Figure 2-4 Multiplexor Bus

2.3 INPUT/OUTPUT

Various I/O techniques can be employed in a PDP-10 System. Some approaches generally considered standard, are discussed in the following sections.

2.3.1. I/O Bus and Control Device

The bidirectional I/O bus (Figure 2-5), comprising 72 separate lines, connects in parallel the arithmetic processor and all I/O devices. Thirty-six of the lines carry data and status information: 14 carry addresses for selecting specific devices; the remainder carry control-type signals. Although a particular device may not use every bus line, all lines must be strapped across its interface and be made available to subsequent devices on the bus. Device interfaces have two sets of bus connectors: one for incoming lines; the other for outgoing lines.

To provide the isolation necessitated by the parallel nature of the bus, each device is equipped, within a control section, with a selection gate capable of decoding a unique 7-bit number. This decoder causes an I/O device to respond only to bus data that includes the device number assigned to the device.

In addition to the selection gate, I/O control devices contain data registers, control registers, and status

registers. The data registers vary in size from 1 to 36 bits. Those with less than 36 bits usually connected to the low-order data bus lines; that is, data enters and leaves the processor in the low-order bit positions.

Control registers vary in size up to 18 bits; status registers vary in size up to 36 bits. The processor is able to specify the function the device is to perform by loading conditions into the control register. It can determine the current state of a device by requesting that the contents of the status register be placed on the data bus lines.
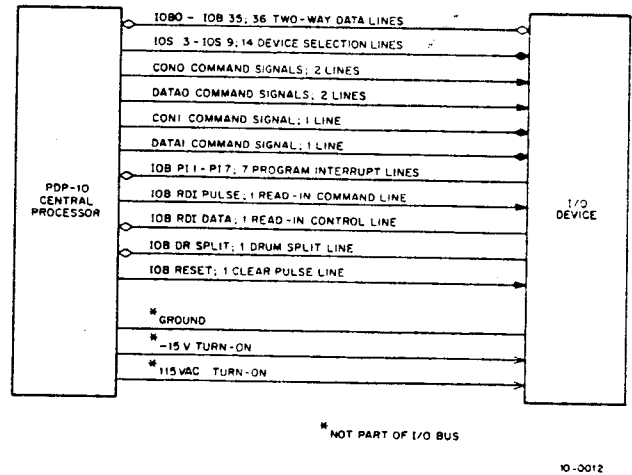
Figure 2-5 Input/Output Bus

2.3.2 I/O Transfers

The processor makes only four demands of I/O devices: take initial conditions from the data lines, place status bits on the data lines, place data on the data lines, and take data from the data lines. For this purpose it uses four basic I/O instructions:

a. Conditions Out (CONO) Transmits the control word to the device control register to specify the desired operation.

b. Conditions In (CONI) Transmits the contents of the device control and/or status register to the processor.

c. Data Out (DATAO) Transmits processor data to the device data register.

d. Data In (DATAI) Transmits the contents of the device data register to the processor.

2-5

2.3.3 Basic Input Transfers

This discussion assumes a system that is not using the priority interrupt system and that is not time-shared.

When the processor requires data from a device, it places the device number on the device selection lines (Figure 2-5), places the control word on the data lines, and sends two CONO pulses. The first pulse prepares the device to receive the control word; the second pulse commands the device to read the 18-bit control word from the data bus. Typically, the control word starts the device and/or defines the type of I/O transaction to take place.

After issuing the CONO commands, the processor proceeds with other tasks until it is ready to take data from the initialized device. At this time, the processor sends a CONI and the device responds by sending a status message across the data bus. If the status message received indicates that the I/O device has data ready, the processor sends a DATAI pulse. DATAI reads the data into the processor over the data bus and initializes the device to send another word. The processor reads successive words of a message by repeating the CONI/DATAI sequence. Of course, the device number must accompany every command to the device.

The processor recognizes the end of a transfer as a result of the program or receipt of an end-of-file status from the input device. When a transfer is complete, the processor deactivates the device with CONO signals.

2.3.4 Basic Output Transfers

As with the basic input transfers, this discussion assumes a system that does not use the priority interrupt system and is not time-shared.

Output transfers in a basic system (Figure 2-5) are similar to input transfers. They begin with CONO pulses that perform device-initializing functions.

When the processor wants to send data, it senses the readiness of the device with a CONI. If the status message returned by the device indicates that the device is ready, the processor can transfer data with DATAO commands. Successive word transfers are accomplished by repeated CONI/DATAO cycles. When an entire message has been transferred, the processor turns off the device by issuing a CONO instruction.

2.3.5 Priority Interrupt in a Time-Shared System

The most efficient use of processor time is made by employing the interrupt system. Using this system, the processor initializes an I/O device and returns to other data manipulations until the device sends an interrupt signal indicating its readiness.

Initially, the processor assigns an I/O device to one of the seven priority interrupt lines by issuing a CONO command containing the channel number in the three low-order bits of the control word. The control word, in addition to specifying the priority channel assignment, typically activates the device and defines the type of I/O transfer to take place. The processor also must turn on the PI (priority interrupt) system, enable the appropriate PI channel, and be able to call the program subroutines required to service the device. With the initializing tasks complete, the processor is free to perform other programmed operations.

When the device is ready to participate in the specified transfer, it interrupts the processor by grounding the assigned priority interrupt lines. To determine which device is requesting the interrupt, the processor repeatedly issues CONI-type instructions to devices assigned to the interrupt channel, placing a different device number on the device selection lines each time. As each device is queried, it responds with a status message. The message from the signaling device contains a set ready flag and flags that define the particular condition within the interrupting device. When the processor detects the active ready flag, it can effect the data transfer with DATAI or DATAO commands. Data transfer clears the interrupt condition in the device. At the end of the I/O transaction, the processor issues CONO signals to deactivate the I/O device. As in all I/O proceedings, the device number must accompany all control and data outputs from the processor.

2.3.6 Hardware Read-In

The user, by means of the console switches, can initiate data read-in from an I/O device. This is a function of processor hardware and of special hardware in each I/O device intended for hardware read-in operation.

Initially, the number of the device is entered on the read-in device switches. When the console READ IN switch is depressed, the processor generates an IOB RESET pulse (Figure 2-5). This pulse halts all motion in the I/O devices and clears all conditions that ini-

tiate interrupts. The devices remain in this null condition until reactivated by CONO commands.

Following IOB RESET, the processor sends an IOB RDI pulse to the device addressed by the console switches, and the special device logic forces the device into the read-in mode. For example, IOB RDI causes the DECtape to prepare to read data, starting at block 0.

When the device is ready with input data, it brings up the IOB RDI data line. The processor responds by reading in the first word and using it to determine the starting address for storing the data words that follow.

2.3.7 Drum Split Signal

The drum split signal (IOB DR SPLIT) is used by high-speed devices that are controlled by the processor and which access memory through a separate memory bus. When active, this line prevents the processor from making memory read-modify-write requests. Instructions requiring read-modify-write cycles are performed instead with separate read and write cycles. Thus, processor control of memory is limited to one memory cycle at a time.

A device brings up the IOB DR SPLIT line whenever a high-speed transfer must be executed to or from memory. In this manner, the device is prevented from

requesting its first memory cycle during a read-modify-write cycle previously initiated by the processor.

NOTE

The DF10 system does not use this feature.

The processor itself may also be assigned a channel. Through this channel, the real time clock and certain processor flags are able to interrupt system operations.

2.4 PRIORITY INTERRUPT SYSTEM

The priority interrupt system permits I/O devices requiring attention to interrupt processor operation. I/O bus interrupt requests, which correspond to channels that the program has enabled, are sampled and stored in flip-flops. The request to be honored is determined by a priority chain. If requests occur at all channels simultaneously (see Figure 2-6), PI request 1 is honored first, while PI request 7 is honored last.

When, as a result of an interrupt, the processor enters a device service routine, the routine cannot be interrupted by another device on the same or on a lower-priority interrupt channel. If an interrupt occurs on a high-priority channel, however, the routine in process is interrupted and the processor enters its service routine for the higher-priority
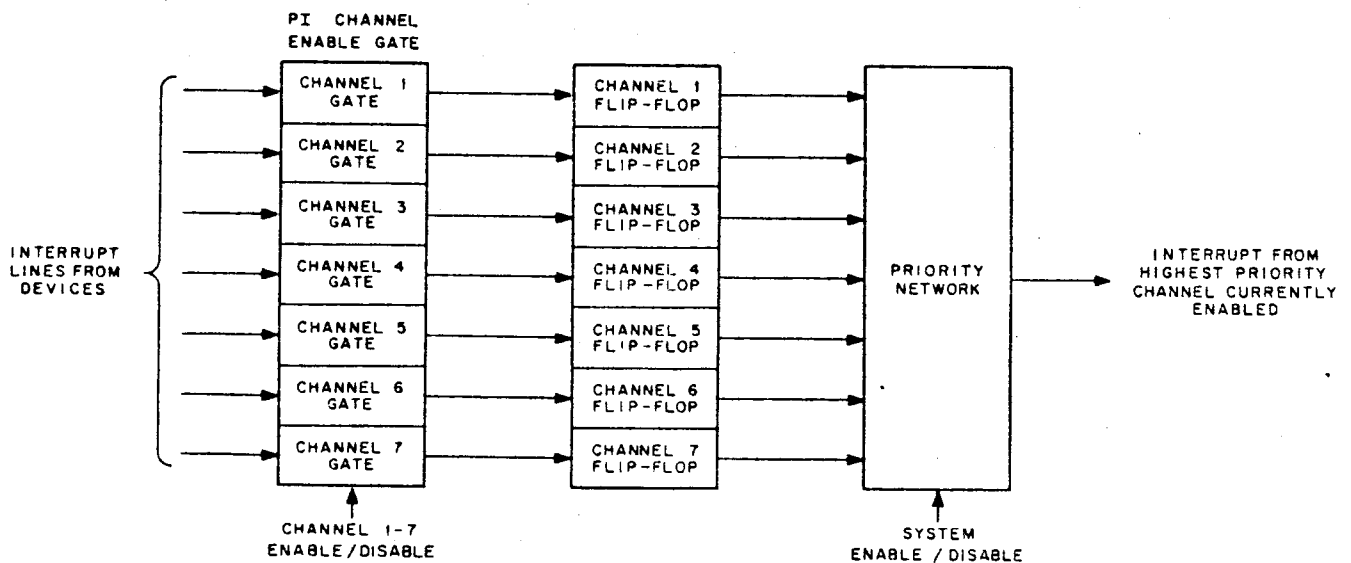
Figure 2-6 Priority Interrupt System, Functional Diagram

2-7

channel. When the latter routine is finished, the processor returns to the place in the lower priority routine at which the interrupt occurred. For example, if the processor is performing a routine for a device on channel 7 and a channel 6 interrupt occurs, the processor stops the channel 7 routine und proceeds to the routine for channel 6. If, while performing the channel 6 routine, it is interrupted by channel 2, the processor interrupts and proceeds to the channel 2 routine. If no other interrupts occur, the processor completes the channel 2 routine, finishes the channel 6 routine and, finally, returns to the channel 7 routine. When all routines are complete, the processor returns to the instruction it was performing at the time of the channel 7 interrupt. The act of interrupting and backlogging successive interrupts having ascending priorities is referred to as nesting interrupts.

2.5 PROGRAMMING

One-user systems operate in a single mode, known as an executive mode. In such systems, the user's program and the monitor program are both able to dictate all machine activities and maintain continuous control. (Generally, the user's program calls the monitor program to perform I/O, but this is not a requirement.) Multi-user systems operate in executive and user modes. Here, machine control alternates between the monitor program and various user programs. The executive program dictates overall system operation, while user programs assume control of the processor to handle jobs for specified users.

The basic purpose of the monitor program in a multi-user system is governing user-program access to the machine to assure most effective machine operation. One monitor program is capable of handling the executive tasks associated with the most complex PDP-10 System. Portions of this program can be deleted as required to provide a monitor that is tailored to the needs of a specific system. Certain portions of the monitor are rarely deleted, however, as they control operations that are standard to most systems. A few of the standard executive tasks are outlined in the following sections.

2.5.1 Time Allotment (Simplified Description)

To provide all user programs with an equitable share of machine time, the monitor establishes a time-sharing sequence (referred to as "queues" in programming). The monitor grants control of the processor to one user at a time for a predetermined period (typically, several

hundred milliseconds). During this time period, the selected user program executes as many of its instructions as possible. When the allotted time expires, the monitor transfers control to another user. The cycle repeats when all users have been granted one control period.

If a user program reaches a point where it cannot continue until it can access a particular I/O device, the monitor removes the user from the time-sharing sequence. The user is returned to the sequence as soon as the I/O transfer is complete.

When an interrupt occurs, the monitor halts the user program in progress, services the interrupt, and returns to the interrupted program. Such a transaction does not significantly affect the interrupted user's allotted time in the time-sharing sequence.

Systems without enough core memory to accommodate simultaneously all of its user programs may expand their storage capacities with mass storage. In such a system, the monitor brings programs stored on the mass storage device into memory before they are needed in the time-sharing sequence.

Using a sophisticated algorithm, the monitor swaps programs, not currently required to be in core, onto the disk (or drum) and swaps programs waiting to run into the core. Normally, this swapping takes place at the same time that some other user's program is running. When a program is to be run again, it is again swapped back into core.

2.5.2 Memory Protection and Relocation

Before being loaded, each user program requests a specific number of core memory locations. As such requests are made, the monitor assigns blocks of locations to users, progressing from the lower to the higher memory addresses. Since all user programs are written to start at memory location 0, the monitor positions these programs in various areas of memory by assigning each one a relocation constant. Memory addresses referenced by a user program are incremented by the associated constant and, as a result, the program is relocated to a specific memory area. Every time the user program runs, the monitor must fetch the appropriate constant. When a program is removed from memory, the monitor, in some cases, reassigns relocation constants and repositions the remaining users so that all unused memory locations exist in the highest memory addresses.

To prevent the memory references of one program from being relocated into another user's area, the monitor stores the highest location assigned to each user. If a user program specifies an address (unrelocated) that exceeds the highest location in its memory area, the memory cycle is inhibited. As with relocation, the memory protection operation is performed whenever a user program addresses memory.

2.5.3 Input/Output

User programs are not ordinarily allowed to control I/O devices directly. If this were possible, one user might try to operate a device already carrying out an order from another user.

As an alternative, all user requests for I/O operations are made by means of UUOs, which cause the machine to store immediately the requesting instruction in monitor memory address 40_8. The machine then traps to location 41_8 from which the monitor accesses its library of I/O subroutines. This library comprises subroutines for performing all I/O operations including such tasks as: determining whether or not an I/O device is available, performing input or output operations on an available device, and controlling a user-selected substitute for a device. When the I/O task has been accomplished, the processor returns to the step in the user program that follows the point from which the call occurred.

I/O functions that take place as a result of interrupts are also handled by the monitor program. A pair of monitor locations, within addresses 42_8 - 57_8, is assigned to each of the seven interrupt lines. When active, an interrupt line traps to its assigned pair of locations. From these locations, the monitor accesses the subroutines that service the active interrupt. Upon completing the I/O transaction, the processor returns to the program step in process at the time of the interrupt.

Table 2-1 details the memory locations in the central processor hardware reserved for I/O and other special functions.

Table 2-1
Reserved Memory Locations

Location (Octal)*	Function
00-17	General-purpose registers
0	Program read-in pointer word

Table 2-1 (Cont)
Reserved Memory Locations

Location (Octal)	Function
20-37	Reserved for expansion or loader
40	Programmed operator storage
41	Programmed operator trap
42-57	Priority interrupt trap
60,61	Byte and floating point trap locations (if the extended order code option is not included)
62-77	Reserved for expansion

*Additional locations are reserved by the monitor for user-specified monitor data.

User mode IOT is a special user mode in which the user program has direct access to I/O devices. Of course, since the priority interrupt system operates as an I/O device, an IOT user program can prevent all interrupts, including the executive time-sharing interrupt. Therefore, user programs requiring the IOT mode must be assembled and used with great care because once such a program gains control of the processor, it effectively becomes an executive program.

2.5.4 Conditions Storage

Whenever the monitor switches control from one user to another, or interrupts a user to perform an I/O operation, it stores certain machine conditions so that they can be duplicated upon return to the current user program. Data stored for use on return includes the contents of the accumulators, the contents of the program counter, certain processor flags indicating processor operating conditions, and the protection and relocation constants. Storage is effected in the first few locations of the associated user program. It is restored when the user program is recalled or re-entered from an interrupt.

2.6 INSTRUCTIONS

2.6.1 Instruction Word Formats

There are two types of instruction words: the basic instruction word that defines processor operations

not involving I/O, and the I/O instruction word that defines processor operations that involve I/O operations. A number from 0 through 6, contained in bits 0 through 2 identifies the word as a basic instruction (Figure 2-7). In this case, bits 0 through 8 are the operation code and define the specific operation to be performed. The remaining bits decode as shown in Table 2-2.

Table 2-2
Remaining Bits

Bit Numbers	Function
9-12 (AC)	Specify 1 of 16 accumulators to be used in executing the instruction.
13 (I)	When this bit is set, the operand is fetched, not from the location addressed, but from the address specified by the data stored at the addressed memory location.
14-17 (X)	If not 0, specify the index register whose contents must be used to alter the memory address in bits 18 through 35. If 0, indicate no indexing.
18-35 (Y)	Indicate an 18-bit quantity, which may be the operand or the memory address in which the operand is stored.

If bits 0 through 2 contain 7, the word is identified as an I/O instruction (Figure 2-8). In this case, bits 3 through 9 specify the address of the I/O device, and bits 10 through 12 define the I/O operation to be performed. The remaining bits (13 through 35) decode as in a basic instruction.

2.6.2 Unimplemented User Operations - UUOs (formerly called Unused Operators)

Operating codes 040_8 through 077_8 are called programmed operators or unimplemented user operators

(UUOs) and are employed by user programs to communicate with the time-sharing monitor. When an instruction containing one of these codes appears in a user program, it is stored immediately in location 40_8 of the monitor and the machine traps to location 41_8. From this location, the monitor calls subroutines to determine the nature of the user's call and to execute the desired operations.

Some UUOs request I/O service from the monitor; others request monitor-stored information such as date and time of day; still others call special monitor subroutines such as those that provide software implementation of floating-point arithmetic in systems without optional floating-point hardware. In general, user programs resort to the use of UUOs to request monitor intervention for performing tasks they are incapable of commanding.

Instructions 001 through 037_8 are also considered UUOs and function in the same manner as codes 040_8 through 077_8. However, they are involved with locations 40_8 and 041_8 (relocated) in the user's own program rather than in the monitor program. Subroutines called from these locations are in the user program.

2.6.3 Effective Address Calculation

Without exception, all instructions calculate an effective address, using bits 13 through 35 (Figures 2-7 and 2-8). That is, the memory address, Y is altered according to the indexing and indirect addressing operations specified by the X- and I-fields. The method of calculation is outlined below.

a. Obtain the number in the address field, Y (bits 18 through 35). Any of 262,144 locations can be specified.

b. If the index field, X (bits 14 through 17), is non-zero, add the contents of the specified index register to the number obtained in step a.
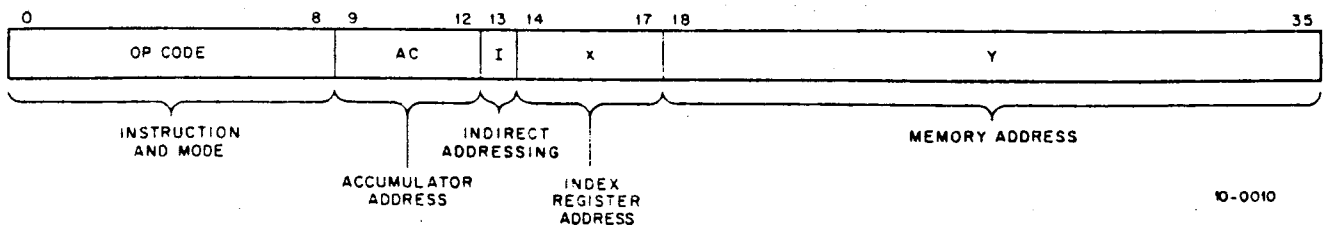
Figure 2-7 Basic Instruction Word

2-10
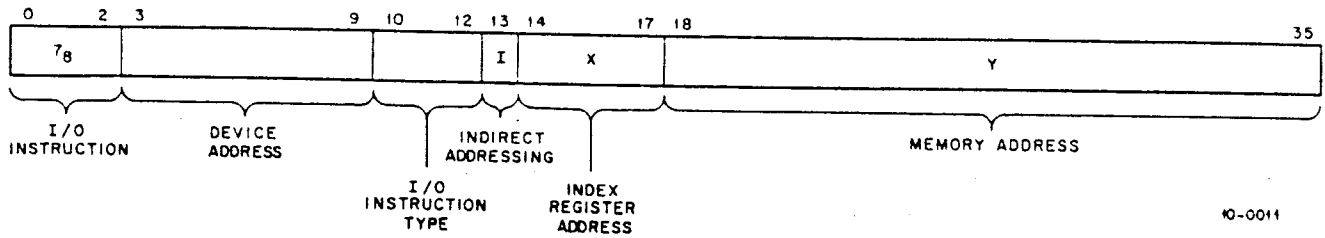
Figure 2-8 I/O Instruction Word

c. Obtain the indirect bit I (bit 13). If it is 0, the calculation is done and the result of a. and b. is the effective address. If it is 1, go to d.

d. Use the address calculated in a. and b. to obtain a new word from memory, and return to a.

The effective address calculation continues until a word is encountered with a 0 in bit 13. At that point, the result of a. and b. is taken as the effective address for the instruction.

Figure 2-9 shows the flow diagram for this process.

10-0008
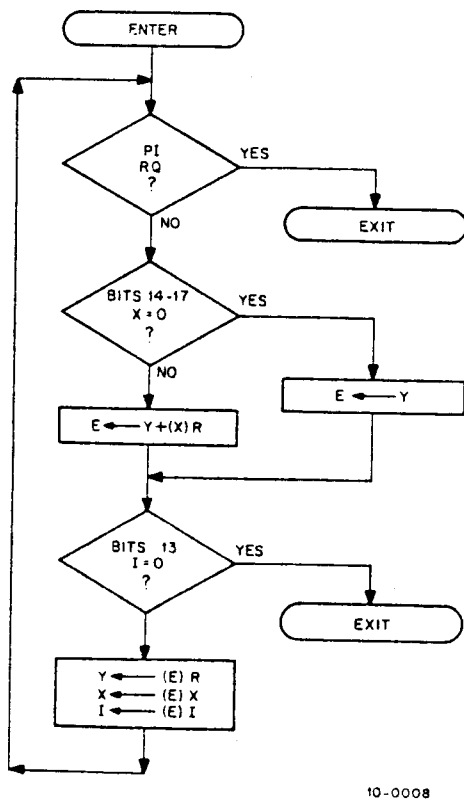
Figure 2-9   Effective Address Calculation

2.6.4 Instruction Classes

It is convenient to consider an instruction as belonging to one of five classes. Four classes (data transmission, arithmetic and logical, executive, and pushdown) are included in the basic instruction format. The fifth class, I/O, is specified in the I/O instruction format.

The description of the instruction contained in this manual is intended to provide a general background for understanding the operation of the KA10 Central Processor. A detailed description of each of the 366 instructions from the system programmer's point of view can be found in the PDP-10 System Reference Manual (DEC-10-HGAC-D).

Each instruction class is composed of numerous instructions. Those in the data transmission, arithmetic and logical, and executive classes are grouped into instruction types. Specific operations and conditions concerning the execution of the instruction types are specified by various modes. The class, type, and mode are all defined by the instruction bits.

2.6.4.1 Data Transmission – Four types of instructions are in the data transmission class: full word, half word, byte manipulation, and miscellaneous. For the most part, they consist of simple data transfers involving core memory and/or 1 or more of the 16 processor accumulators. Full-word instructions transfer 36-bit words, half-word instructions transfer 18-bit words, and byte manipulation instructions transfer any number of contiguous bits in a 36-bit word. The miscellaneous instructions are exchange and block transfer. The former exchanges the contents of the effective address with the contents of the specified accumulator. The latter moves blocks of words from one contiguous group of addresses to another. Modes associated with this instruction class define operating details such as the direction of data transfer, and the form (swapped, negated, set to magnitude) in which the data is transferred.

2.6.4.2 Arithmetic and Logical - Four types of instructions are in the arithmetic and logical class: fixed-point arithmetic, floating-point arithmetic, boolean, and shifting. Arithmetic instructions cause the processor to add, subtract, multiply, or divide. Fixed-point arithmetic involves whole numbers, and floating-point arithmetic involves mantissa and exponents. Both are done in 2's complement binary; that is, carries from the highest order bit are dropped, and negative numbers are the 2's complement of the positive numbers with the same magnitude. The processor is hard-wired for single-precision arithmetic but is easily programmed for double precision.

Boolean instructions combine the effective address or its contents with the specified accumulator on a bit-by-bit basis. They provide for all 16 possible boolean functions of two variables.

Shift instructions perform various types of shifts on the contents of an accumulator or on the combined contents of two consecutively addressed accumulators.

The modes associated with this instruction class specify operating details such as where to store results and what to do with remainders from arithmetic calculations.

2.6.4.3 Executive Instructions - Five types of instructions comprise the executive class: memory and accumulator modification and testing, arithmetic compare, logical compare and modify, jump, and miscellaneous.

Within the memory and accumulator modification and testing instructions are two groups: accumulator jumps and memory skips. The jump instructions test and/or modify the contents of the accumulator. If the conditions specified by the mode are met, program control jumps to the instruction at the effective address. Skip instructions test and/or modify the contents of the effective address. If the conditions specified by the mode are met, the next instruction is skipped.

Arithmetic compare instructions algebraically compare the contents of the effective address with the contents of the accumulator, or vice versa. If the conditions specified by the mode are met, the next instruction is skipped.

Logical compare and modify instructions skip by testing and possibly modifying the bit positions of the accumulator corresponding to bit positions in the

memory word containing 1s. The mode specifies the conditions under which a skip occurs.

Jump instructions are of two basic types: unconditional and conditional. Unconditional jumps cause the processor to take its next instruction from a specified memory location. Conditional jumps cause the jump only if certain conditions exist.

Included in the miscellaneous instructions is the execute instruction which forces the machine to execute the instruction located by the effective address. Essentially, the instruction modes in the executive class detail the type of modifications and comparisons to be performed and the conditions to be tested.

2.6.4.4 Push-down Instructions - Push-down instructions provide a means of inserting or removing data from lists that are stored in consecutive memory locations. They operate in such a manner that the last data placed on the list is the first removed.

2.6.4.5 I/O Instructions - The eight I/O instructions are used for transferring data and control information to I/O devices and for transferring data and status information out of I/O devices.

2.6.5 Instruction Execution

Processor operation is initiated by means of a special key cycle that supplies timing for events associated with intervention from 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.

Most instructions are executed by the five cycles that comprise the main sequence. The five cycles are: instruction, address, fetch, execute, and store. Each main sequence begins when the instruction cycle requests memory access to retrieve an instruction word. When it receives the instruction, the processor enters the address cycle, wherein it calculates the effective address. If an indirect address is encountered, a new address word is retrieved from memory, and the cycle begins again. After repeating the cycle as many times as necessary to produce the effective address, the processor proceeds to the fetch cycle.

In fetch, the processor fetches the operands required for the actions specified by the instruction. After fetching the operands, the processor enters the

execute cycle during which it executes whatever logical, arithmetic, or control functions are necessary to carry out the instruction. The program counter is incremented during the execute cycle. Incrementing the counter by one causes it to point to the next instruction in the sequence.

Finally, the processor enters the store cycle. For most instructions, the store cycle places the results of the execution cycle in an accumulator, in memory, or in both. The processor then returns to the start of another instruction cycle. During the instruction cycle of the new sequence, the incremented program counter is used to obtain the next consecutive instruction in the program.

The main sequence uses a hierarchy of other sequences (built-in hardware subroutines) for performing the operations within its five cycles. These subsequences are called directly by the main sequence or by any subsequence of higher rank within the hierarchy. Thus, the processor operates, using many levels of nested sequences; each sequence stops upon calling a lower rank sequence and restarts upon return from it (although the restart need not be at the point of departure).

Most processor control functions involved in the retrieval and setup of instructions and the retrieval and storage of operands take a negligible amount of time when compared to memory access time. For each memory access the processor must first check for memory protection and relocation and then wait until the addressed memory is free. Approximate instruction execution times may be determined from the flow charts included in Volume II of this manual.

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, program counter incrementation is inhibited in all but the final repetition. In this way the counter points to the next instruction only when the current one can be completed before an interruption can occur.

Because instructions are executed by nests of sequences, performance times vary from one instruction to another. For example, multiplication and division are performed by a series of additions and subtractions. The time required for such major sequences depends upon the number of times the various subsequences must be called.

This chapter describes the operating controls and indicators of the KA10 Central Processor. Included in it are: operating procedures for the arithmetic processor, paper tape reader/punch and teleprinter, and some operator maintenance instructions. Detailed operating instructions for maintenance purposes are provided in Chapter 6 of this volume.

3.1 OPERATOR'S CONSOLE

The KA10 console (Figure 3-1) has 36 data switches, 18 memory address (MA) switches, and 20 control switches. Of the 20 control switches, 10 are momentary contact switches and 10 are latching switches.

The ten momentary contact switches are: READIN, START, CONT (continue), STOP, RESET, XCT (execute), EXAMINE THIS, EXAMINE NEXT, DEPOSIT THIS, and DEPOSIT NEXT.

When depressed, these switches (except STOP) generate some or all of the pulses (KT0 through KT4) in the key cycle that initialize processor operation. Each of these switches sets the corresponding flip-flop that is used to gate functions required for the operations associated with the switch. The flip-flops are cleared at the end of the key function unless a repeat action is dictated by the REPT (repeat) switch. Simultaneously depressing two momentary contact switches may result in erroneous processor operations.

The latching switches are: SING INST (single instruction), SING CYCLE (single cycle), PAR STOP (parity stop), NXM STOP (nonexistent memory stop), INST FETCH (instruction fetch), DATA FETCH, WRITE, ADR STOP (address stop), and ADR BREAK (address break). When a latching switch is operated, its function remains active until the switch is disengaged.

STOP, RESET, XCT, EXAMINE THIS, and DEPOSIT THIS are effective when the processor is running (under program control) or stopped. All other switches, except for REPT, are effective only when the machine is stopped. The REPT switch is effective while the machine is running and used in conjunction with EXAMINE THIS, DEPOSIT THIS, or XCT. When used with any of the other switches, REPT is effective only when the machine is stopped.

Tables 3-1 and 3-2 list and describe the function of each of the console switches and indicators, respectively.

Table 3-1
Function of Console Switches

Switch	Function
READ IN	Clears processor and I/O devices. Initiates the reading-in of data from the I/O device specified by the READ IN DEVICE switches (margin check and maintenance panel). When the
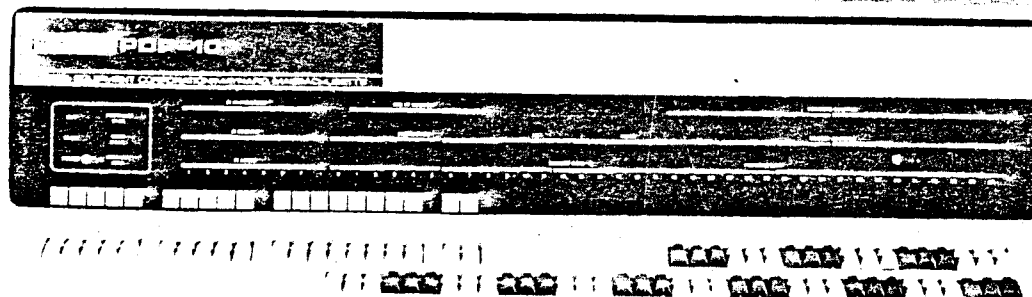
Figure 3-1 KA10 Operator's Console

Table 3-1 (Cont)
Function of Console Switches

Switch	Function
READ IN (Cont)	read in is complete, the processor begins operation by executing the last word read in from the device.

NOTE

Do not depress another function switch while a readin is in progress. If a read-in operation must be interrupted (e.g., due to a crumpled tape), clear the read-in condition, using the RESET switch, before proceeding to another function.

Switch	Function
START	Starts program execution at the location specified by the MA switches. Processor and I/O states are not cleared. Memory relocation and protection may or may not occur, depending on the current state (user or executive mode) of the machine.
CONT	Causes program execution to resume from stops caused by any of the following: SING INST, SING CYCLE, NXM STOP, PAR STOP, STOP, ADR STOP, RESET, SHIFT CNTR MAINT, or stops initiated by the program.
STOP	Halts machine operations by clearing the RUN flip-flop and aborting any unending indirect address calculation. STOP cannot be repeated.
RESET	Same as STOP except that it clears processor and I/O devices.
XCT	Causes the contents of the data switches to be executed as an instruction. Priority interrupts are inhibited during this instruction to be sure that it runs to completion. The XCT switch may be used while the machine is running. However, if the machine is in the user mode, relocation and protection is in effect, and attempted I/O instructions may be trapped.

Switch	Function
EXAMINE THIS	Causes the memory indicators (MI) to display the contents of the word whose address is in the MA switches. MEMORY DATA illuminates above the memory indicators. Protection and relocation are inhibited so that the absolute rather than the relocated address is referenced. EXAMINE THIS may be used while the machine is running, in which case the memory fetch is synchronized between two instructions.

NOTE

Memory protection and relocation are inhibited in all deposit and examine functions.

Switch	Function
EXAMINE NEXT	Increments the contents of the MA by one. Displays the contents of the resulting address in the MI with MEMORY DATA illuminated. This key is not functional unless the machine is stopped.
DEPOSIT THIS	Writes the contents of the data switches into the memory address specified by the MA switches. The data is displayed in the MI with MEMORY DATA illuminated. If the switch is activated while the processor is running, the action is synchronized between two instructions.
DEPOSIT NEXT	Increments the contents of the MA by one and writes the contents of the DATA switches into the resulting address. The data is shown in the MI with MEMORY DATA illuminated. This key is not functional unless the machine is stopped.
SING INST	Halts the processor (with the RUN flip-flop off) after each instruction is completed (prior to ITO for the following instruction). This allows a program to be stepped through one instruction at a time, using the CONT switch. To prevent clock interrupts that would otherwise appear between every instruction while stepping slowly, the SING INST switch inhibits the setting of the processor clock flag. Instructions that never end, specifically non-

Table 3-1 (Cont)
Function of Console Switches

Switch	Function
SING INST (Cont)	terminating indirect-address calculations, are stopped by the STOP or RESET switch but not by the SING INST switch.
SING CYCLE	Stops the processor at the end of each memory control subroutine, prior to the generation of MCRST0 or MCRST1.

NOTE

The memory subroutine is used for all memory references that reference instructions, indirect references, or effective address contents. The memory subroutine is not used in machines having internal fast-memory options when making index register or accumulator references, since this would slow down these operations. However, single-cycle operations involving fast memory references may be performed by turning off the FM ENABLE switch (margin check and maintenance panel), causing core memory locations 0 through 17 to be used as accumulators. As with any STOP, activating the CONT switch, after a single-cycle stop, resumes processor operation.

PAR STOP	Stops the processor at the end of a memory subroutine in which a word containing incorrect parity is read. Running with this switch on slows all memory read operations the amount of time required to check parity. Therefore, it is not usually advisable to operate with PAR STOP on unless memory errors are suspected. In either case, the parity of memory reads is checked and, if an error is detected, parity error flag of the processor is set.
NXM STOP	Stops the processor when reference is made to a nonexistent memory location (i.e., one which is beyond the memory size of the system or one which for some reason does not re-
NXM STOP (Cont)	spond to memory requests). For read references, a word containing 0s is assumed. For write references, the data is discarded. The CONT switch may be used to proceed with the program.
REPT	Causes actions specified by any of the momentary function switches (except STOP) to be repeated. The rate of repetition is dictated by the SPEED CONTROL switches on the margin check and maintenance panel. If, with the processor running, the REPT and CONT switches are both active, the processor stops and then continues through programmed stops (hardware or software). The period of the stop is determined by the SPEED CONTROL switches.
ADDRESS CONDITION:	
INST FETCH	Used in conjunction with ADR STOP or ADR BREAK. Causes a stop (ADR STOP) or break (ADR BREAK) in the execution of the program when an instruction fetch or indirect address operation references the address in the MA switches.
DATA FETCH	Used in conjunction with ADR STOP or ADR BREAK. Causes a stop (ADR STOP) or break (ADR BREAK) in the program on all memory read references except when an instruction fetch or indirect address occurs and the referenced memory address is equal to the contents of the MA switches.
WRITE	Used with ADR STOP or ADR BREAK. Causes a stop (ADR STOP) or break (ADR BREAK) on all memory writes that reference the address in the MA switches.
ADR STOP	Stops program execution when the condition specified by the ADDRESS CONDITION INST FETCH, DATA FETCH, or WRITE switch is encountered.

Table 3-1 (Cont)
Function of Console Switches

Switch	Function
ADR BREAK	Causes a priority interrupt on the KA10 PI channel when the condition specified by the ADDRESS CONDITION INST FETCH, DATA FETCH, or WRITE switch is encountered.

NOTE

The functions of the ADDRESS CONDITION INST FETCH, DATA FETCH, and WRITE switches occur with core memory references or fast memory references if E = FMA. If an ADR STOP or BREAK is required on other fast memory addresses (i.e., AC or XR field of an instruction), the FM ENABLE switch on the margin check and maintenance panel must be disabled. The function can then be performed on the appropriate core memory location.

Switch	Function
Data Switches (0 through 35)	The contents of these switches are written into memory when DEPOSIT THIS or DEPOSIT NEXT is active. They are executed as an instruction when XCT is active. The program can sense the contents of these switches with a DATAI instruction.
Memory Address Switches (18 through 35)	These switches define the memory addresses required for the functions associated with the following control switches: START, EXAMINE THIS, DEPOSIT THIS, ADR STOP, and ADR BREAK. The contents of the switches are continually compared with the contents of the MA register plus the contents of the relocation register (if relocation is in effect) and the fast memory address mixer. When

Switch	Function
Memory Address Switches (18 through 35) (Cont)	comparisons yield equality, logic terms are generated, enabling certain processor functions. The contents of the MA switches may also be jammed into the MA register.

Table 3-2
Function of Console Indicators

Indicator	Function
RUN	Lights when the RUN flip-flop is set, indicating that the program is running or is hung up in a loop.
PI ON	Indicates that the PI system is enabled.
POWER ON	Indicates that system power is on, and the power-up sequence is complete.
PROGRAM STOP	Indicates that the machine has stopped as a result of a halt instruction.
MEMORY STOP	Indicates that the machine has stopped as the result of a memory reference and a condition associated with any of the following switches when the switch is active: SING CYCLE, PAR STOP, NXM STOP, ADR STOP.
USER MODE	Indicates that the machine is in the user mode.
IOB PI REQUEST (7 indicators)	Indicate active PI lines from the I/O devices prior to their entering the PI system.
PI ACTIVE (7 indicators)	Indicate enabled PI channels.
PI REQUEST (7 indicators)	Indicate active PI channels that are being considered by the PI priority network.
PI IN PROGRESS (7 indicators)	Indicate which PI channel, of those set, is being serviced by a subroutine.

Table 3-2

Function of Console Indicators (Cont)

Indicator	Function
	NOTE The subroutine is servicing the highest priority PI channel.
PROGRAM COUNTER (18 indicators)	Display contents of the program counter.
INSTRUCTION/ AC/I/INDEX MEMORY ADDRESS (36 indicators)	The left half displays the contents of the instruction register (operating code, accumulator address, indirect-address bit, and index bits). The right half displays the contents of the memory address register.
MEMORY/ PROGRAM DATA (36 indicators)	Operate under control of the console switches to show the contents of memory locations, or under program control to display various types of data via the AR.
MEMORY DATA	Lights when the MEMORY/PROGRAM DATA indicators display the contents of a memory location as a function of the console switches.
PROGRAM DATA	Lights when the MEMORY/PROGRAM DATA indicators display data as a function of the program.
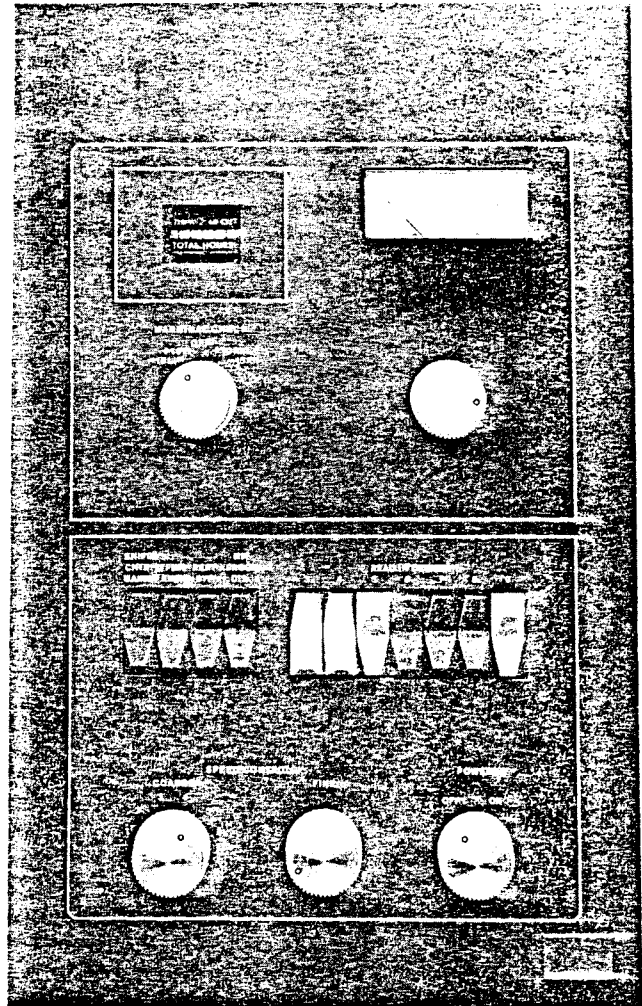
Figure 3-2 Margin Check and Maintenance Panel

Table 3-3
Function of Margin Check and
Maintenance Panel Controls

3.2 MARGIN CHECK AND MAINTENANCE PANEL

The margin check and maintenance panel (Figure 3-2) is used for maintenance purposes. The controls on the margin check section of the panel are used to control application of either margin or standard voltage to the system. The controls of the maintenance section are used to control exercising of the central processor. The function of each of the controls and the indicator on the panel is given in Table 3-3.

Name	Function
MARGIN CHECK -15L/+10L/OFF/+10R/ -15R Switch	Selects the polarity of the margin check power-supply output. Connects the selected voltage to the left or right side of the system. (The processor and all cabinets to the left of it comprise the left side of the system; cabinets to the right of the processor comprise the right side.)

Name	Function
Margin Check Vernier	Varies the output from the margin check power-supply. Approximate limits are: 0 to 20V, unloaded; 20V, 7A, maximum load.
FM ENB Switch	Substitutes the fast memory registers for the first 16_{10} core memory locations.
SHIFT CNTR MAINT Switch	Transfers control of shift-counter stepping from the shift-counter timing chain to the console CONT switch.
MI PROG DIS Switch	Overrides program control of MI register displays, returning control to the MA switches.
REPT BYP Switch	Reinitiates the key cycle selected by the console switches when, due to a malfunction, the selected key cycle hangs up. The rate of the repeat is selected by the SPEED CONTROL switches. The REPT switch must be active to utilize REPT BYP.

NOTE

When using the REPT BYP switch, make certain that the delay selected by the SPEED CONTROL switches is longer than the period of the selected key cycle. Otherwise, the selected key cycle will be reinitiated before it can run to completion.

SPEED CONTROL, COARSE/FINE Controls	Select time delays that determine the rate at which the repeat function reinitiates the key cycles. Six basic delays can be selected by the COARSE SPEED CONTROL.

SPEED CONTROL, COARSE/FINE Controls (Cont)	The FINE SPEED CONTROL varies the six basic delays so that overlap occurs between the range of delays that can be selected by adjacent positions of the COARSE SPEED CONTROL. The positions and corresponding delays are: Position 1: 270 ms to 5.4s Position 2: 38 ms to 780 ms Position 3: 3.9 ms to 78 ms Position 4: 390 µs to 7.8 ms Position 5: 27 µs to 540 µs Position 6: 2.2 µs to 44 µs
READIN DEVICE Switches	Select the I/O device used in a readin operation.
POWER Switch	Controls application of primary power to the system.
TOTAL HOURS Meter	Records power-on time from 00000.0 to 99999.9 hr.

3.3 BAY 1 and 2 INDICATORS

The indicator panels (Figure 3-3) at the top of bays 1 and 2 are provided primarily for maintenance purposes. They display I/O bus data, the status of all control flip-flops, and the contents of all processor registers including the I/O register. Status indicators are also included for many critical enabling levels.

3.4 PAPER TAPE READER/PUNCH AND DECTAPE

Power is available to the reader/punch (Figure 3-4) and DECtape (Figure 3-5) when processor power is on. A description of the switches and controls, and operating data, such as tape-loading procedures, are provided in: Perforated-Tape Reader PC02 Instruction Manual (DEC-08-I2BA-D), and Royal-McBee Model 500 Maintenance Manual and Instruction Manual TU55 DECtape Transport (Document No. DEC-00-HZTA-D).

3.5 TELEPRINTER

The teleprinter provides two-way communication between operator and computer. It is actually two independent devices (keyboard and printer), which may be operated simultaneously. Either a Teletype® Model 35 KSR (Figure 3-6) or Model 37 KSR Teleprinter (Figure 3-7) is provided with the KA10.

® Teletype is the registered trademark of Teletype Corporation.
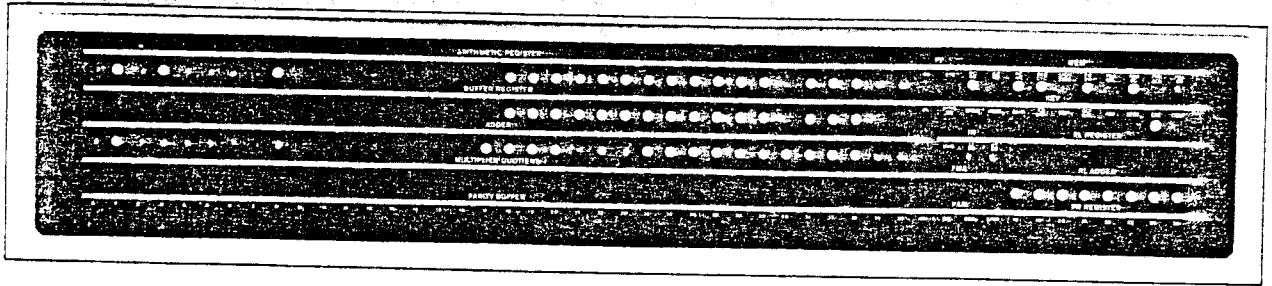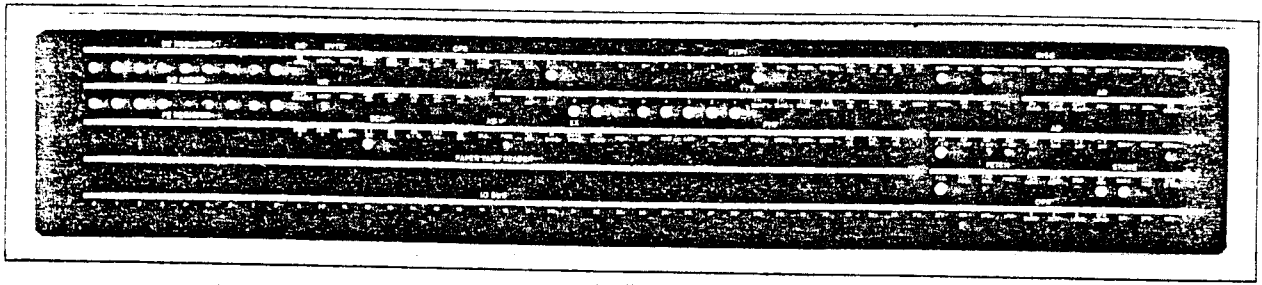
Figure 3-3   Indicator Panels, Bays 1 and 2
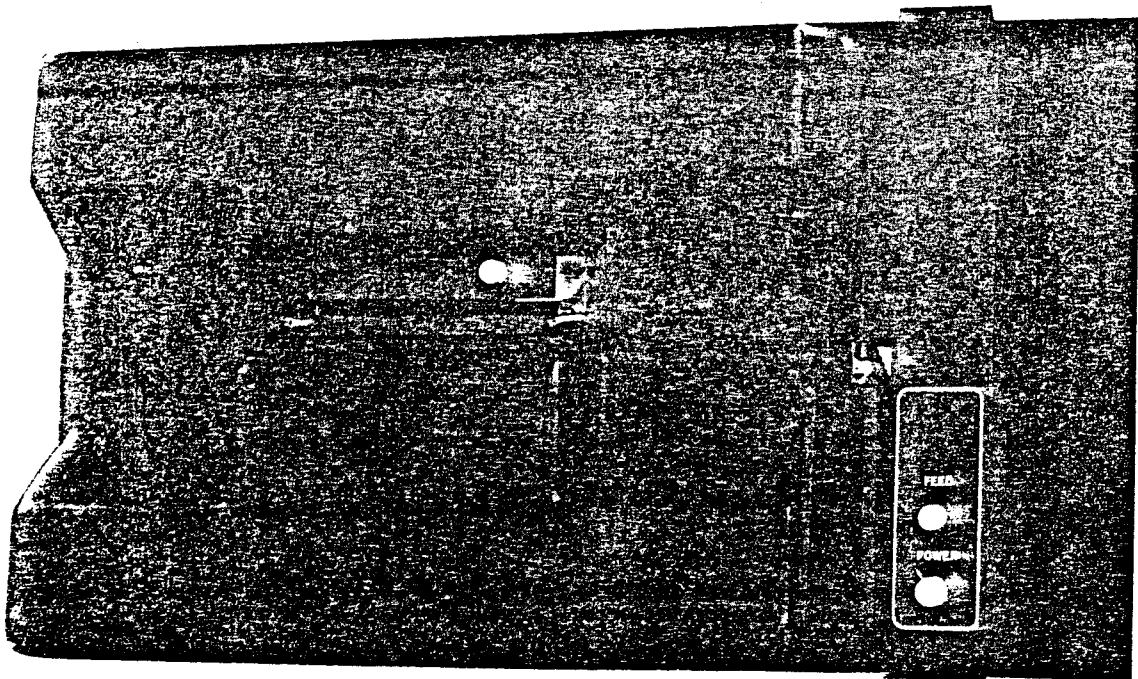
Figure 3-4   Paper Tape Reader/Punch
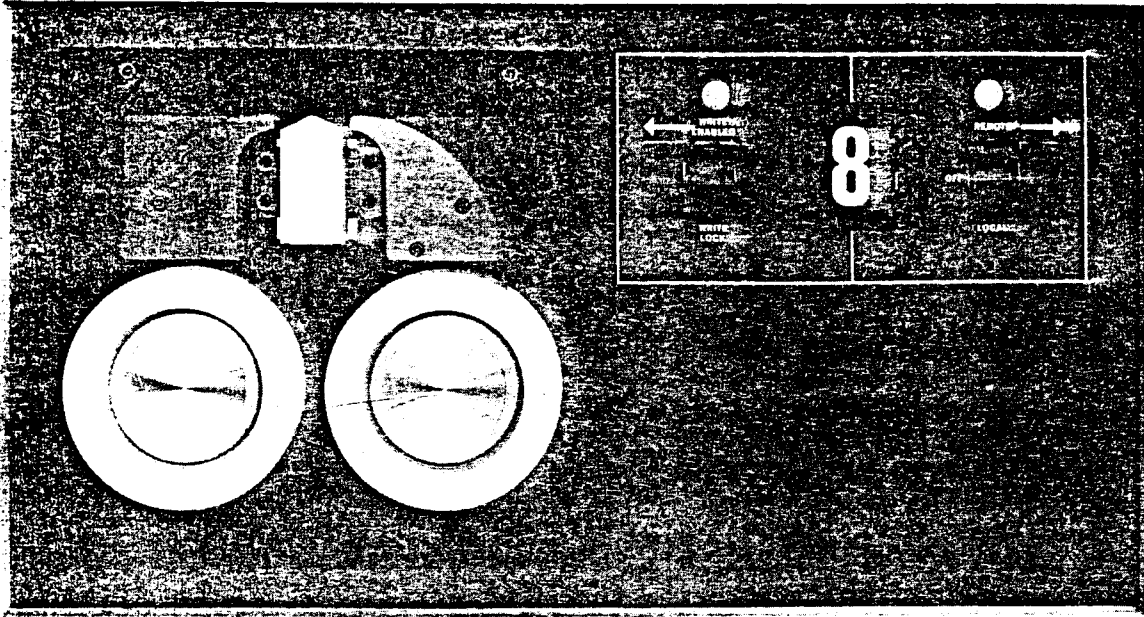
Figure 3-5   DECtape
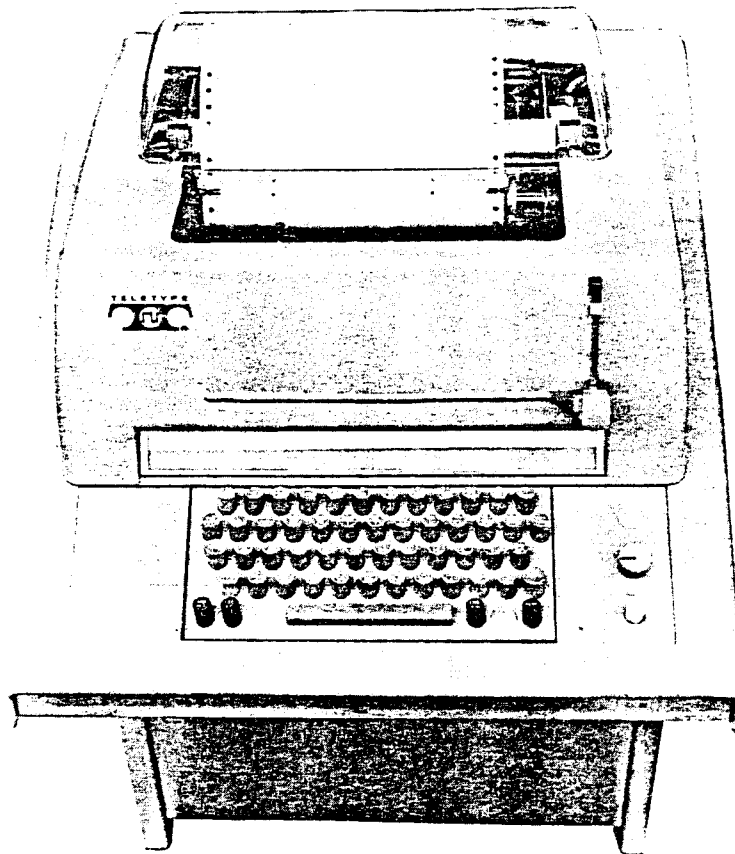
Figure 3-6   Teletype Model 35 KSR
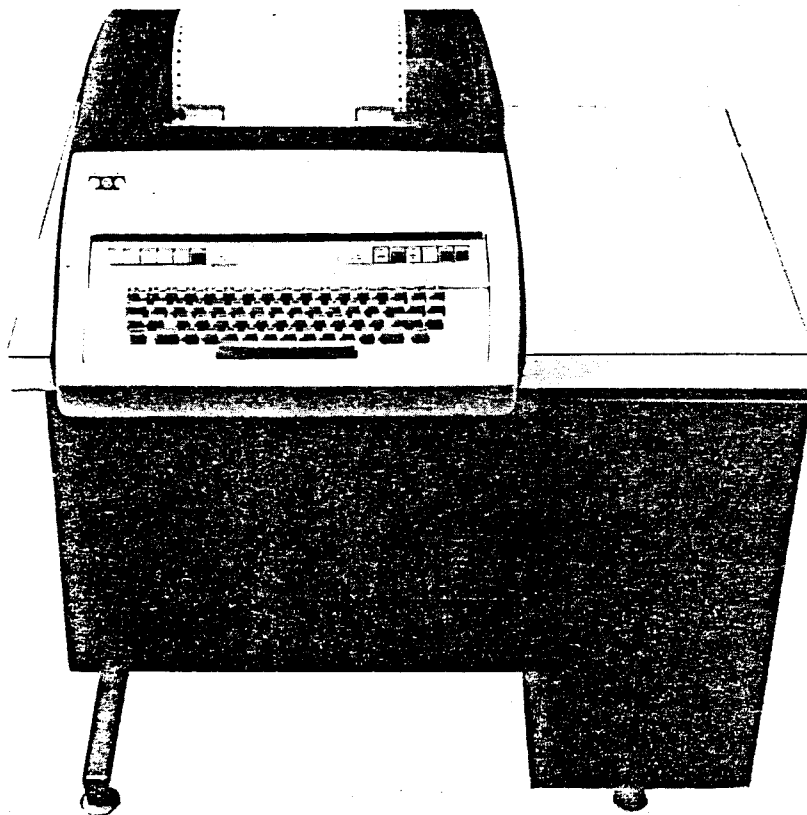
Figure 3-7   Teletype Model 37 KSR

Both models of the teleprinter operate with 8-bit characters plus start and stop control signals transmitted serially. The 35 KSR operates at 10 characters per second and the 37 KSR at 15 characters per second. One of the most significant differences between the two models is that the 37 KSR prints out lower case letters, as well as upper case.

Power is available to the teleprinter when KA10 power is on. When on, the LINE/OFF/LOCAL switch is set to LINE, the unit is "on line", and it goes on and off with system power.

The keyboard of both the 35 KSR and 37 KSR resembles that of a standard typewriter in that it has four rows of keys and a space bar. Striking a key transmits a character to the teleprinter control logic. The character is printed or the function executed by the teleprinter only if the processor sends it back to the printer.

Striking only the character keys transmits the codes for the characters on the lower part of the keys.

Striking the character keys while holding down the shift key transmits the codes for printable characters on the upper part of the keys (punctuation, ampersand, percent sign, etc.). On the 37 KSR, lower case is normally transmitted and the shift key gives upper case. 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. However, codes for some of the control functions have no effect on the printer when they are sent back.

The code used is the American Standard Code for Information Interchange (ASCII) with adaptations for the PDP-10. The operational differences between the 35 KSR and the 37 KSR are cited in the remarks column of the table.

Depressing the REPT button while striking a character key causes repeated transmission of the corresponding

3-9

code for as long as REPT is held down. Characters that require the SHIFT or CTRL key may also be repeated in this manner. (On the 37 KSR, repeated transmissions are accomplished by depressing and holding down the key.) The red button at the left, BRK RLS, is not operative. LOC LF and LOC CR are the local line feed and carriage return buttons. They affect the printer directly and do not transmit codes to the control logic. The white button, BREAK (on the far right), opens the line, sending continuous space.

Details of paper installation, ribbon replacement, tab setting procedures and maintenance procedures for the teleprinter are in Teletype Bulletin 281B.

3.6 READ IN

The read in mode of operation permits placing information in memory without relying on a program in memory or loading one word at a time manually.

Console initiation of read in from the paper tape reader, DECtape, or any other device so equipped, is accomplished in the read in mode. With the address of the appropriate input device entered in the READ IN DEVICE switches, depressing the READ IN switch causes the processor to read the loader program from the input device. If the program is self-starting, the last instruction causes the processor to jump and execute the loader, thereby reading in the remaining tape data. If the program is not self-starting, the processor can usually be induced to execute the loader program by pressing the CONT key.

Details of loaders are in the MACRO-10 Assembler Programmer's Reference Manual (DEC-10-AMD0-D).

To read tapes in either the RIM10 or RIM10B loader, proceed as follows:

a. Load the tape onto the input device as directed in DEC-10-AMD0-D.

b. Enter the device address in the READ IN DEVICE switches.

c. Press the RESET switch.

d. Press the READ IN DEVICE switch.

When using the READ IN DEVICE switches (Figure 3-8), note that the least significant octal digit of the device code is selected by a single switch (switch 9). This final digit can only be a zero or a four.
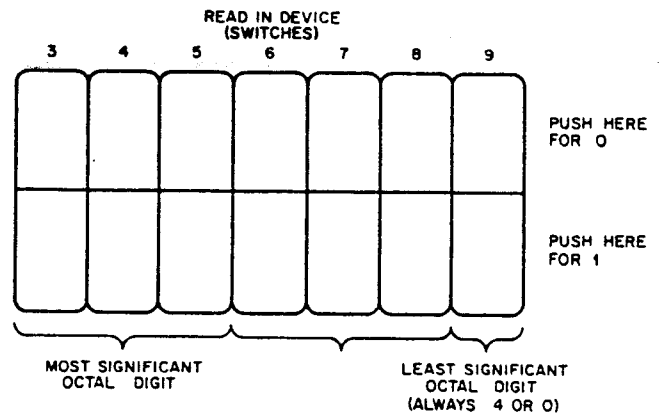
Figure 3-8   READ IN DEVICE Switch

CHAPTER 4
CENTRAL PROCESSOR ORGANIZATION

This chapter is divided as follows:

4.1 KA10 REGISTERS AND ADDERS

Brief descriptions of the registers and adders of the processor and the data paths between them are presented below.

4.1.1 AR (Arithmetic Register, 36 bits)

The AR is the most active register in the processor because it serves as an internal accumulator and as a memory buffer. I/O transfers also take place through this register. It can be loaded from the adder, either directly or with a one-bit shift to the left or right; its halves may be swapped. Other inputs to this register include the SC, IR, memory bus, fast memory, console data switches, and the arithmetic flags.

4.1.2 BR (Buffer Register, 36 bits)

The BR is used primarily to hold the second operand of an instruction. It is loaded only from the AR.

4.1.3 MQ (Multiplier Quotient, 36 bits)

As the name implies, this register holds the multiplier during multiplication, and the quotient during division. This register can be considered as a right-hand extension to the AR, and it can shift right or left; its external input comes from the adder.

4.1.4 AD, ADR (Adder, 36 bits)

The adder is not a register, actually, but a network of gates that generate various logical functions. The primary function of the AD is to add arithmetically, but due to a very flexible system of control gating,

several other functions are possible. The AD can internally produce either the arithmetic sum, or the logical equivalence of its two input branches. (Equivalence is the complement of exclusive-OR.) One input is from the AR, the other from the BR. Each of the two input branches may select one of four functions:

a. all 0s

b. all 1s

c. the register

d. the 1's complement of the register

There is also a +1 input separately selectable, and a +1 or -1 input to the left half.

4.1.5 MI (Memory Indicator, 36 bits)

The sole function of the MI is to drive the console indicators. This register is loaded from the AR.

4.1.6 PB (Parity Buffer, 37 bits)

The PB holds all words going to and from the memory bus for the purpose of computing their parity. Its input is from the memory bus.

4.1.7 MA (Memory Address, 18 bits)

This register is the primary source of address to the memory system. This register is loaded from the AR, PC, and the console address switches.

4.1.8 PC (Program Counter, 18 bits)

This register holds the program address. It is equipped with a high-speed counting gate and is loaded from the MA.

4.1.9 IR (Instruction Register, 18 bits)

This register holds the instruction during execution. Its primary input is from the memory bus; however,

the READ IN DEVICE switches also provide an input. The instruction register is divided into a 13-bit left part and a 5-bit right part; these parts are handled separately during address calculation.

4.1.10 SC (Shift Counter, 9 bits)

In addition to counting shifts, the SC is used for exponent calculation in floating-point arithmetic, and for position calculation in byte operations. For these operations, the SC works in conjunction with the SCAD. The SC has a separate high-speed counting gate of the same type used in the PC. The shift counter gets inputs from the AR, BR, FE, SCAD, and several "magic numbers" used to determine the number of shifts in multiply and divide operations, or as constants for floating point and byte operations.

4.1.11 SCAD (Shift Counter Adder, 9 bits)

The SCAD is a smaller version of the AD, which is used in conjunction with the SC. One of the SCAD inputs comes from the SC; other inputs are provided by selected bits of the AR, BR, and more "magic numbers".

4.1.12 FE (Floating Exponent, 9 bits)

This register holds the exponent of floating point numbers when all other registers are busy. The register is loaded from the SCAD.

4.1.13 PR, PR2 (Protection Registers, 8 bits each)

These registers hold the upper address bounds for the memory protection feature of the KT10A options; these registers are loaded from the I/O bus.

4.1.14 RL, RL2 (Relocation Registers, 8 bits each)

These registers hold offsets that are added to the MA to form the actual memory address, as a feature of the KT10A options; they are loaded from the I/O bus.

4.1.15 PIH, PIR, PIO (Priority Interrupt Hold, Request, and On, 7 bits each)

These registers are part of the priority interrupt system, and are explained in Section 4.7.

4.2 KA10 BASIC CYCLES

During the execution of each instruction, the central processor goes through at least five basic cycles. These basic cycles are:

I   Instruction Cycle - used to retrieve the instruction from memory.

A   Address Cycle - used to compute the effective address.

F   Fetch Cycle - used to fetch the operands from memory and ACs.

E   Execution Cycle - used to perform a basic instruction or to begin execution of a more complicated instruction.

S   Store Cycle - used to return the results to memory or ACs.

The term "cycles" as used in the KA10 has a different meaning than in most computers. In the KA10 the machine is said to be in a given cycle when the active timing pulse is finding its way through the delay lines, gates, and PAs that control that cycle. There are no flip-flops or other multi-stable circuit elements that determine when a fixed group of clock pulses is currently producing a given cycle. Each of the following cycle descriptions is referenced to a flow chart. The I and A cycles are shown on the IAC flow diagram; the F cycle on FC flow diagrams, and the E and S cycles on the ESC flow diagram. These diagrams are included in Volume II of this manual. For those readers not familiar with DEC logic standards and flow diagrams, Appendix A contains a brief review.

4.2.1 Instruction Cycle (See Drawing KA10-0-IAC)

The instruction cycle has only two time pulses, IT0 and IT1. The IT0 pulse begins each instruction, either as a result of the completion of the previous instructions or from some console action, coming from KT4. IT0, through MR CLR (called "mister clear"), clears all registers, and loads the MA with the address of the next instruction.

This address normally comes from the PC, but may, in some cases, already be in the MA, or be a trap address determined by the priority interrupt system. The IR input gates are also enabled to allow the left-most 18 bits of the instruction word, which is retrieved from the memory system, to enter the IR. The entire instruction word also enters the AR, as it does during all memory read operations.

Having established the above conditions, the IF0 flip-flop is set. This flip-flop informs the memory control logic where it should return when the instruction word has been read from the memory system. This is an example of the "hardware subroutine" concept. Control is passed to the memory control logic via the read request input. At the completion of the memory operation, an MCRST1 pulse is generated. (Because this pulse is common for all memory operations, it will be discussed later. The timing will return to IT1 since only the IF0 flip-flop is set; this flip-flop is only one of a group of flip-flops that gate the MCRST1 pulse.)

At IT1, the MA is cleared in case it is required later, and the input gates to the IR are disabled. The left-most 13 bits of the IR hold the operation and accumulator parts of the instruction word and do not change during the address calculation. The right-most five bits are the indirect address and index bits and are zeroed by the address calculation algorithm. The signal flow then branches in one of three ways, depending on whether a priority interrupt is requested and whether indexing is called for. The priority interrupt system is covered in detail in Section 4.7 of this chapter. The normal course is to enter the address cycle.

4.2.2 Address Cycle (See Drawing KA10-0-IAC)

The first part of the address cycle computes the sum of the address and the index register (if indexing is called for). Because the KM10 fast registers are an optional feature of the KA10, each time data is fetched from a register, a decision must be made as to whether the register is in memory or is one of the fast registers. AT1 (Address Time 1 pulse) prepares for this register data fetch by saving the address part of the instruction in the BR and clearing the AR. The control signal path then branches, depending on whether MC FM EN is false or true (memory control, fast memory, enable); if false, the register data fetch is made from a memory register by a fast memory read request; if true, the contents of the index register are loaded into the AR by AT2, after a short delay to

allow the internal address decoders to set up. Previously, the fast memory address decoders were set to the XR part of the IR by the MR CLR pulse. Both paths produce AT2 with the same result.

The delay after AT2 allows the adder to set up the sum of the address in the BR and the index register in the AR. (The appropriate adder enable conditions were previously set by the MR CLR pulse.) At AT3, the sum is loaded into the AR, if indexing was called for, and the original contents of the AR are saved in the BR for later use by the JRST instruction. The left half of the AR is cleared if the address is to be used as an immediate operand, and the fast memory decoders are set to examine the AC part of the IR, in preparation for fetching data from an accumulator.

After a delay to allow the contents of the AR to settle, the indirect bit of the IR is examined. If the indirect bit is set, pulse AT4 occurs next. This pulse loads the MA with the computed address and requests a new instruction to be fetched. The only difference between this and IT0 is that the MR CLR pulse is not produced and the left part of the IR is not disturbed, thereby retaining the same instruction. If the indirect bit is cleared, the address cycle is terminated by AT6, which loads the MA with the calculated effective address. Byte instructions (134-137) set the IR gating to allow the index and indirect bits of a pointer word, which are fetched during the fetch cycle, into the IR. Control of the processor is then given over to the fetch cycle.

4.2.3 Fetch Cycle (See Drawing KA10-0-FC)

The fetch cycle retrieves operands in the following order:

a. memory

b. accumulator

c. miscellaneous operands

After AT6, a decision regarding the memory operands is made. If the instruction requires a memory operand that will not be returned to memory, the FCE condition (Fetch C(E)) will be true, and FT0 will occur; however, if an operand, which will be modified and then returned to memory, is required, the condition, FCE PSE (Fetch C(E)) Pause, will be true and FT1 will occur.

The pulse FT1 occurs after a delay to allow the MA18-31 = 0 gate to set up. Because the fast registers do not have the destructive readout characteristics of core memory, no read-modify-write cycle is provided for them. Hence, any reference to an accumulator is made as a straight read reference. The MC SPLIT CYCLE SYNC flip-flop forces all read-modify-write cycles to be made as separate reads and writes. Instructions that do not require memory operands, and most immediate mode instructions, proceed directly to FT1A; however, floating-point immediate instructions take the immediate mode operand in the left half; therefore, at FT8, the halves of the AR are swapped.

At FT1A, the memory operand, if any, is placed in the BR (except for a JRST instruction, which preserves the previous contents of the BR to be used in restoring the arithmetic flags, if they are required). The next decision that must be made is whether or not an accumulator is needed. (By coincidence, any instruction that does not need an accumulator does not need a miscellaneous operand.) If no AC is needed, FAC INH (Fetch AC Inhibit) is true, and signal flow proceeds directly to FT9, the end of the fetch cycle. If an AC operand is required, it is picked up by FT2 and FT3, in a manner similar to AT1 and AT2 in the address cycle. At FT3, if the next higher AC will also be required, the fast memory address decoder is set to examine the accumulator part of the IR incremented by one. If no further operands are required, control again is given over to FT9.

The remaining fetches are in three categories, as follows:

a. FAC2 (Fetch AC2)

b. FCCACLT (Fetch the $C(C(AC)_{left})$)

c. FCCACRT (Fetch the $C(C(AC)_{right})$)

These additional operands are placed in the MQ. However, the only path from memory or the fast registers is into the AR; hence, as a first step, the AR is saved in the MQ (via the adder). The operand is then fetched, and the AR and MQ interchanged. For the second category of fetch, the FAC2, FT4, FT5, and FT4A accomplish the task. For FCCACRT, the address is in the right half of the AR, which was the AC just fetched. This address is placed in the MA at FT7, and an ordinary read cycle is then requested, returning to FT4A. For FCCACLT, FT6 interchanges the halves of the AR to get the left-hand

address into the MA at FT7. FT4A restores the fast-register address to the first AC in anticipation of the store cycle, and then goes to FT9.

FT9, the end of the fetch cycle, normally increments the PC unless the instruction specifically asserts PC+1 INH. The pulse also sets up gating for the adder, as necessary, for the individual instructions. The KEY SYNC RQ flip-flop is sampled at this time to determine whether there will be a key cycle at the end of this instruction. Control now advances to the execute cycle.

4.2.4 Execute Cycle (See Drawing KA10-0-ESC)

Flow through the execute cycle, though simple, accomplishes the functions of most instructions. This description concerns flow only; the other actions are covered in the detailed description for each instruction. The first condition examined in the execute cycle is EF0 LONG; this condition determines whether the subsequent ET0 pulse is to be delayed for a long or short time after FT9. If the adder is being used for a full-add function (and, hence, needs to propagate carries), EF0 LONG is asserted, choosing the longer delay. ET0 sets the adder enables to the state necessary during the store cycle if the next pulse will be ST1. Two conditions determine what takes place next: E LONG calls for ET1 and ET2 to occur; ST INH (store time inhibit) means that the current instruction has an additional timing chain of its own which occurs as the result of one of the execution pulses. Unless the ST INH condition is asserted, control progresses to produce ST1, after either ET0 or ET2, with the adder output holding the contents of the BR.

4.2.5 Store Cycle (See Drawing KA10-0-ESC)

The store cycle is entered from the execute cycle at ST1, or at ST0 from the individual instruction timing chains. The ST0 pulse is provided to set up the adder for subsequent store cycle operations. The store cycle attempts to skirt around as many pulses as possible, consistent with what has to be done; hence, there are many paths from ST1 to ST9. ST1 stores the AR in the fast registers, if they are present, and AC store is not inhibited.

If no other registers are to be stored, control passes directly to ST9; however, if the current instruction was performed as the result of a priority interrupt, pulse ST1A and a delay are squeezed in to allow the

4-4

PIOV and PICYC flip-flops to clear before progressing to IT0.

If the content of an AC is to be stored, and the fast registers are not present, pulse ST1 requests a fast memory write, and control eventually returns to ST2 after the memory subroutine.

Control also reaches ST2 when the content of a second AC is to be stored, or when the result to be stored in memory is in the BR. (The BR data is not the same as the AR data: $SAR \neq BR$). When a second AC is stored, the fast register address is changed to that of the second AC. When a result is to be stored in memory from the BR, the AR is loaded from the adder, which contains the BR.

ST5, ST6, and ST6A perform the storing of a memory result. ST5 is used when the memory result is modified from a word previously used from the same location (at FT1). Pulse ST5 may be produced as the result of ST2, or directly from ST1. Pulse ST6 performs ordinary memory writes and also may be produced as the result of either ST2 or ST1. After returning from the memory subroutine, the store cycle ends if the second AC was not required. When data is stored in a second AC (SAC2), ST7 retrieves the data from the MQ and, after a delay, ST8 either puts this data into the fast registers or calls the memory subroutine to store the data in memory.

ST2 also has a direct path to ST9 in the event that ST2 was necessary only because of the lack of fast registers. At ST9, a decision is made whether to start a new instruction or to enter the key logic cycle.

4.3 KA10 BASIC INSTRUCTIONS

The basic instructions are so called because they require no timing pulses other than those from the basic cycles. These instructions are covered as they appear on basic instruction flow diagrams BIF1, BIF2, and BIF3 in Volume II of this manual.

4.3.1 Boolean

The Boolean group includes 16 instructions in 4 modes. These instructions perform all possible functions of two variables; they are decoded as BOOLE 0 through BOOLE 17 (octal). The mnemonics appear below the relevant flow chart section. The E LONG condition is selected for those functions requiring the complement of the AC and/or the other variable. Operands

are fetched only when necessary because some cases are degenerate. The adder is set by FT9 to give a true or complemented BR, or the complemented AR, or the equivalence (XOR) of the AR and BR. At ET0, the result is either jammed, ORed, or ANDed into the AR. For all, except the four long instructions, control goes directly to ST1; for the long instructions, ET1 and ET2 form the complement of the AR.

4.3.2 Add, Sub

The distinction between these two groups of four instructions is made at FT9, when a subtract selects the 2's complement of the BR by taking its complement plus one. The result is then put into the AR at ET0, and, if required, the overflow and carry flags are set.

4.3.3 Full Word Transfer (FWT)

The FWT group is composed of the four variations of MOVE. If negation is possible, EF0 LONG is selected to allow enough time. AD AR NEGATE sets the adder gating for the complement of the AR plus one. This result may be strobed into the AR at ET0, or may be ignored if the instruction is MOVMX and the AR is already positive.

4.3.4 Half-Word Transfer (HWT)

The HWT group is divided into two independent sections. The second section, requiring E LONG, includes only the following 4 of the 64 instructions in the group:

a. HRL

b. HLR

c. HRLI

d. HLRI

These four instructions have the property of being a function of both the BR and AR, and that the BR (or memory) argument must have its halves swapped. In this group, ET0 interchanges the AR and BR, and, at ET1, the bits are shuffled in the AR as necessary.

The first HWT section contains the remaining 60 instructions, which, for the most part, require only one operand. If all 1s are needed for a half word, the adder is first set up with both BR+ and BR− inputs

active, thus guaranteeing a 1 in every bit position. At ET0, either the destination half of the AR is already correct (HLLXX or HRRXX), or the other half of the AR is loaded into it (HLRXX or HRLXX). The other half either is available from the adder, or will be 0. For HXXEX instructions, a special term called HWT E TEST is used. This term looks at the correct bit of the AR to determine the sign of the result.

4.3.5 Exchange (EXCH)

This instruction interchanges the memory and AC operands and uses the SAR ≠ BR signal to have the two results handled separately. This instruction requires only ET0, which interchanges the AR and BR.

4.3.6 Unimplemented User Operation (UUO)

All operation codes not assigned as specific instructions (except 247 and 257) are executed as UUOs. In such cases, the word given as an instruction is trapped and must be interpreted by a routine included for this purpose by the programmer.

Operation codes 000 through 127 are always treated as UUOs. If the KE10 extended order code is not implemented in the processor, the IR FP TRAP SW is on and extended order codes 130 through 177 are also treated as UUOs. IOTs and some JRST instructions are illegal and are also treated as UUOs in the non-privileged user mode (i.e., when the EX ALLOW IOTS signal is not present). For example, the JRST 4 instruction that halts the processor and the JRST 10 instruction that restores the PI system are both illegal if the system is in the non-privileged user mode. Furthermore, all IOT instructions (operation codes 7xx) are illegal if the system is in the non-privileged user mode or if no PI cycle is in progress.

Any of the above conditions cause the IR UUO level to be true. As a result, the given instruction, with its calculated effective address, is stored in a trap location, and the processor executes the instruction at the next location. The trap location used by the UUOs varies. In user mode, a UUO with an operation code in the range 001 through 037 traps to user location 40 (relocated 40). All other UUOs, in user mode or otherwise, trap to absolute lower memory locations. The absolute trap location will be 40, 60, 140, or 160, depending upon operation codes and processor configuration.

In a single processor system, the MA TRAP OFFSET switch is not set. UUOs with operation codes 000 through 077 (except 001 through 037 in user mode) trap to absolute location 40. UUOs, with operation codes 100 through 177, trap to absolute location 60.

In a dual processor system, where it is undesirable to have both processors trap UUOs to the same locations, the MA TRAP OFFSET switch is set in one of the processors. For UUOs executed by that processor, absolute trap locations are offset by 100. UUOs with operation codes 000 through 077 trap to absolute location 140, and UUOs with operation codes 100 through 177 trap to absolute location 160.

The MA register is preserved in its 0 state throughout the fetch cycle, and, hence, at ET0, the trap address bits can be ORed in, but are controlled by the conditions explained above. All non-relocated UUOs set the EX ILL OP flip-flop, which suspends relocation to allow the instruction at the non-relocated address to preserve the state of the user mode condition. The EXCTF flip-flop is set to signify that the MA is not to be loaded from the PC by the next IT0 because the MA contains the address of the next instruction to be executed, less one. The EUUOF flip-flop causes the one to be ORed into the MA at IT0. The UUO instruction is stored during the normal store cycle because MA contains the correct address. The PC+1 operation at FT9 is inhibited for the UUO; thus, in normal operation, a JSR instruction at the trap address causes one PC increment, and stores the address of the instruction following the UUO. If an interrupt occurs between the UUO or JSR instruction ordinarily found at the trap address, the UUO is restarted from the beginning upon return from the interrupt routine.

4.3.7 Jump

4.3.7.1 Jump to Subroutine (JSR)

The JSR instruction stores the PC and flag word at the effective address, and jumps to that address, plus one. The incrementation is done at ET2. For reasons which are explained later, all instructions that store the flags clear BYF6, a flip-flop in the byte instruction control.

4.3.7.2 Jump and Save PC (JSP)

JSP saves the PC and the flags in the AC.

4.3.7.3 Jump and Save AC (JSA)

The JSA is another subroutine-calling instruction. This instruction stores the AC at E, and stores the PC and E in the AC. Having E in the left half of the AC allows the JRA instruction (refer to Paragraph 4.3.10) to find the storage location of the AC and thus restore the AC when returning from the subroutine. ET0 saves the AC (which was loaded into the AR during the fetch cycle) in the BR. The right half of the AR is replaced by the old PC, which was incremented by FT9. The PC is loaded from the MA. At ET1, the old PC is moved to the left half of the AR, and the new PC (containing E) is loaded into the right half of the AR. At ET2, the halves of the AR are swapped, and the PC is incremented to get around the word currently being stored.

4.3.7.4 Jump and Restore AC (JRA)

The JRA instruction is intended for use as a return from a subroutine previously entered by JSA instructions. If the same AC is specified in the JRA and JSA instructions, the left half of the AC points to the location where the old contents of that AC were stored. The FCCACLT level then causes the fetch cycle to retrieve the word whose address is contained in the left half of the AC, and to place it in the MQ. The effective address is usually indexed by this same AC, thereby getting an address relative to the location from which the subroutine was called.

The effective address will not be in the MA, because the MA was previously changed by the FCCACLT, but will be found in the BR. The effective address is sent to the PC via the MA and AR, using the three ET time pulses. The content of the AC is retrieved from the MQ during ET1.

4.3.7.5 Jump on Flags and Clear (JFCL)

JFCL tests for any of the four flags being on, and clears those that have been tested. The four flags are specified by the four AC bits of the instruction. If a detected flag is on, the PC is loaded from the MA during ET0. If all AC bits are off, this becomes a "no-op" instruction and is the fastest "no-op" instruction in the instruction repertoire.

4.3.7.6 Jump and Restore (JRST)

JRST is the simplest unconditional jump instruction. This instruction can do several useful things when

called for by bits in the AC field of the instruction. IR10 specifies halt, which clears RUN. This option causes E LONG in order to allow the console lights to be set up to show the old PC in the MA lights. Halting is not allowed in user mode, so IR10 changes the JRST into a UUO when present. IR9 restores the interrupt system, and is discussed with the priority interrupt system. IR9 also changes the JRST into a UUO in user mode.

The IR11 specifies that the flags be restored from the left half of the last word referenced in the address calculation. For the IR11 option to be meaningful, the JRST instruction should be indexed or indirected. The BR is saved throughout the fetch cycle so that it is available for the index or indirect operation. The flags are in the same stored format as the JSR, JSP, and PUSHJ instructions.

IR12 sets the EX MODE SYNC flip-flop which puts the machine into user mode for the next instruction. This flip-flop can also be set by restoring the flags (with IR11) with bit 5 on in the flag word.

4.3.8 Test

The test instructions are another 64-instruction group. The memory operand is used as a mask to select bits in the AC operand. The selected bits may be tested, and then either cleared, set, complemented, or left unchanged. The TDXX and TRXX groups are the same except that TRXX is an immediate group. The TSXX and TLXX are also the same except that TLXX is an immediate group. At FT9, the adder enablers are set up to provide the true mask for TXOX, the complement of the mask for TXZX, and the exclusive OR of the mask and AR for TXCX.

Because the halves of the mask in the BR cannot be swapped, the AR is swapped for TSXX and TLXX by both ET0 and ET2; this accomplishes the effect of swapping the BR halves. At ET1, the AR is ANDed into the BR, and the adder is set so that the true BR is available from the adder at ET2. Simultaneously, at ET1, the new result is formed in the AR: TXOX ORs the adder into the AR, TXZX ANDs the adder to the AR, and TXCX jams the adder into the AR. ET2 increments the PC when the test condition is true, as determined by the instruction and whether or not the adder is zero.

4.3.9 Add One to Both and Jump (AOBJ)

There are two instructions in this group: AOBJP (add

one to both halves of the AC and jump if positive) and AOBJN (same as AOBJP but jumps on negative result). The addition of one to both halves is accomplished by a special adder enable that simulates a one in bit 17 of the BR input of the adder. Note that, if the right half of the AC is 777777 before this instruction, two is added to the left half: an explicit one, and a carry from the right half. This phenomenon takes place in all instances in which both halves of the AR are incremented.

4.3.10 Skips

The Skip group includes instructions in the AOS, SOS and SKIP classes. AOS and SOS use the FCE PSE mode because they add and subtract one, respectively, from memory locations. Some of the logic conditions specified by this group are common to the JUMP group which performs similar actions with respect to ACs. The FT9 pulse sets the adder enablers for the true AR (either plus one, minus one, or unchanged).

The adder result is jammed back into the AR at ET0, and the skip condition is tested. The skip conditions for the skips, jumps, and comparisons are all encoded by PC conditions P, Q, and R. These conditions are logically derived from the skip or jump conditions (as described in the PDP-10 System Reference Manual) and have been reduced to an OR of six 3-way ANDs.

Conditions Q and R are similar: condition R is used for the skips and jumps, and depends only on the adder sign bit (AD0); condition P depends on whether the adder output is equal to zero and also includes the "always" skip case. The AOSX and SOSX instruction groups set the AR flags as indicated.

4.3.11 Jumps

The Jump group includes AOJ, SOJ, and JUMP classes. These classes are similar to the corresponding skip classes, and operate on ACs, jumping when the specified conditions are met. The jump logic is largely common with the Skip group described previously.

4.3.12 Compare

The Compare group, CAMX and CAIX, have the same modifiers as the Skip and Jump groups previously mentioned. The CAMX compares the content of the AC to memory and, hence, uses the FCE. CAIX is immediate; the comparison is accomplished by subtracting the memory operand from the AC, and examining the result in the adder. The conditions P, Q, and R, mentioned in the Skip group, also apply to the Compare group. Condition Q is used in place of condition R because a subtraction may overflow and, therefore, AD COND, which corrects for overflow, is used in place of AD0. Condition P remains the same.

4.3.13 PUSH

This instruction is representative of the PUSH, POP, PUSHJ, and POPJ instructions in that they all use an AC as a pointer word. The left half contains a count that is tested during each of these instructions to detect whether it has passed through zero; if it has, the CPA PDL OV flip-flop is set, usually indicating an error in the program.

The PUSH instruction specifically adds one to both halves of the AC, and uses the resultant right-half address as the location in which to store the contents of the effective address.

4.3.14 Push and Jump

After incrementation, as in PUSH, the PUSHJ instruction stores the PC and flag word at the address specified by the AC. Here, because the path from the PC and flags is into the AC, the result of the pointer word incrementation is loaded temporarily into the MQ during ET0. At ET1, the AR is moved to the BR, and the AC is brought back to the AR from the MQ. ET2 loads the address part of the pointer into the MA. The results are stored as with the PUSH instruction.

4.3.15 POP

POP is the opposite of PUSH; this instruction uses the FCCACRT part of the fetch cycle to retrieve the word pointed to by the AC. FT9 sets the adder to subtract a one from both halves of the AR containing the pointer. This is accomplished by setting the BR + and BR − inputs to get the quantity −1; also the AD − 1 LH, which introduces a zero at bit 17. These actions add the number 777776777777 to the content of the AR. At ET0, this result is put in the MQ, and the word retrieved from memory is brought into the AR from the MQ. At ET1, the contents of the AR and BR are exchanged, leaving the data word in the BR, the effective address in the AR, and the pointer in the MQ.

At ET2, the effective address is placed in the MA and the pointer is brought back to the AR.

4.3.16 POPJ

POPJ is the subroutine return used to exit from a subroutine called by the PUSHJ instruction. Its operation is identical to that of POP (above) through ET0. At ET1, the address part of the data word is loaded into the MA, and the pointer is brought back to the AR. At ET2, the PC is loaded from the MA, causing the next instruction to be taken from the address specified in the right half of the word removed from the pushdown list.

4.3.17 Execute (XCT)

The execute instruction causes the contents of the effective address register to be executed as an instruction without changing the normal instruction sequencing (unless a jump-type instruction is executed). Hence a skip instruction skips, relative to the execute instruction, and is independent of its own location. The only function of the execute cycle during an execute instruction is to set the EXCTF flip-flop, which causes the next IT0 to use the address already in the MA when fetching the instruction. The PC is not incremented unless the instruction being executed increments it.

4.4 KA10 ADDITIONAL INSTRUCTIONS

4.4.1 Extended Instructions

4.4.1.1 Shift and Rotate
The Shift and Rotate group includes ASH, ASHC, LSH, LSHC, ROT, and ROTC. The flow of this group of instructions is shown on the "SMF" flow chart. The XXC instructions cause the second AC to be fetched and stored in the MQ during the FAC2 part of the fetch cycle. The direction of shift is determined by the sign bit of the effective address (bit 18). A positive number causes a left shift. Because the SC counts up from a 2's complement negative number, the correct representation of the number is loaded into the SC from the BR at ET0 if the effective address is negative. If the BR address is positive, ET0 sets up the SCAD to negate the SC.

Because these instructions specify ST INH, ET0 is the last main-flow time pulse until this group returns to STO. The next pulse will either be SCT0 or SRT1; SRT1 is used to negate the SC, if necessary, and, then, it goes to SCT0 also.

SCT0 provides entry to the shift and count subroutine and is explained in detail below. The various shift instructions differ only in determining where the bits at the end of the registers go. The shifting pattern is shown on the "SCAF" flow diagram. Both the AR and MQ are shifted for all instructions in this group because shifting the MQ does no harm, even though nothing of interest is contained in the MQ. The shift and count subroutine returns at SCT4 and is fed directly to ST0 for this class.

4.4.1.2 Shift and Count Subroutine
The shift and count subroutine includes the timing for all places where a known number of shifts are to be executed. Because shifting of the AR is accomplished through the adder, the shift and count logic also performs the inner loop of multiply and divide routines. In addition to saving logic, this permits the shift speed to be set in only one place; logic is shown on the "SCBT" flow diagram.

The shift and count subroutine is made up of pulses SCT0 through SCT4. The first pulse, SCT0, is the common entry point to the subroutine; this pulse always sets the adder for the true AR. The BR inputs are controlled as a function of the instruction calling this subroutine. The SCAD SC + 1 setup is accomplished (mostly for historical reasons), although a new high-speed counting circuit has been added to circumvent the SCAD. The SC STOP switch is tested and, if on, the SC STOP flip-flop is set, thus allowing single stepping through the shift iterations. A delay then allows the adder gating and the gating which determines bit position to set up. Control passes to SCT4 immediately, if no shifts are necessary. SCT1 is generated if the SC is still negative and the SC STOP is zero. SCT1 counts the SC using the counting gate (B166s).

The branch after SCT1 determines whether any additions or subtractions are being done this time around; if so, it squeezes in SCT2 and a delay, which lengthens the loop time to 280 ns. SCT3 does the actual shifting as gated by the particular instruction. SCT3 also sets the adder for the next time around the loop if the instruction is a multiply or divide. The basic shift-loop time is set to 150 ns. This loop is closed by SCT3, and the sign of the SC is again examined to determine whether the loop must be repeated. If the SC STOP is on, control is resumed by pushing CONT on the console. SCT4 is the subroutine exit, which also clears the SC STOP, and MSF1, for a multiply.

4.4.1.3 Fixed-Point Multiply – The two kinds of fixed-point multiplication in the PDP-10 System differ only in how results are treated. Integer multiply discards the high-order result word and sets overflow if the high-order word contains any significant bits. Multiplication is accomplished directly with 2's complement numbers. The algorithm is explained in detail in the System Reference Manual.

Both the MULX and IMULX instructions start by enabling the BR + input to the adder, at FT9, which is used at MST0. ET0 loads the SC with the number 735$_8$ (equivalent to -35 decimal), the number of times around the shift loop. Control passes from ET0 to MST0, which rearranges the operands. The multiplier, or memory operand, is loaded into the MQ from the BR via the adder. The multiplicand, or accumulator operand, is sent to the BR. The MPF2 flip-flop remembers whether both operands are negative. This condition is used later to check for a possible overflow.

MST1 sets the MSF1 flip-flop, which controls adder gating. Whenever MQ34 and MQ35 are equal, the BR inputs to the adder are disabled. If MQ34 and MQ35 are 01, the AD MD+ level is generated; this causes the BR to be added to the quantity in the AR in the adder. When MQ34 and MQ35 are 10, AD MD- is generated; this subtracts the BR from the quantity in the AR. These conditions affect the adder at each SCT3 pulse in the shift and count subroutine. The initial adder setup depends only on the state of MQ35 and is done by MST1. The AR is cleared at this time to initialize the product.

The product is developed in the AR, and is shifted into the MQ as bits are vacated by the multiplier being shifted off the right-hand end of the MQ. This process is carried out by the shift and count subroutine (SCT0 through 4). During fixed-point multiply, control returns from SCT4 to MPT2. Here, the last result is loaded from the adder into the AR while the MQ is shifted right, thus bringing the sign of the result into the MQ sign bit. If the result is negative, and both the operands are negative, the AROV flip-flop is set, since two negative numbers should have a positive product. (The only possibility of this case is 400 000 000 000 times itself.)

If IR6 is on, the instruction is a MULX, and control passes directly to the store cycle via ST0. If IR6 is not on, the instruction is IMULX, and the AR result is tested for being either all 0s or all 1s; either condition means that there is no significance in the AR. The adder enables are set for only the AR+ input, and if the AR is negative, the AD CRY INS effectively complements the output of the adder. The AD = 0 condition is tested at MPT4 and, if false, AROV is set. MPT4 also brings the result in the MQ back to the AR. IMULX stores only one AC, while MULX also stores the second (unless the instruction was a MULM, in which case no ACs are stored.)

4.4.1.4 Fixed-Point Divide – As with multiply, there are also two kinds of divide instructions, DIVX and IDIVX. IDIVX takes a single-word dividend and processes it as the low-order word. Because there is no simple direct 2's complement division algorithm, a method is used which takes the magnitude of the dividend but keeps the divisor in complement form. The fixed-point divide instructions are shown on the "DIVF" flow chart. The divide instructions take separate paths until they combine during DST1, the divide subroutine which is also used for floating point. IDIVX sets up the proper dividend in the AR and MQ registers during the normal execution cycle. At FT9, the adder is set up to give the negative of the dividend in the AR. At ET0, the negative dividend is read back into the AR, if the dividend was negative in the first place. The SC is set up for a count of -35 (decimal) and the adder gates are set for the true AR.

Then, at ET1, the dividend is moved into the MQ, and the AR is cleared. Thus, the AC operand is in the position of the low-order word in a regular divide instruction. ET2 shifts the MQ one place left to remove the sign bit. Control then is given over to DST1 (see below).

Because DIVX may have to negate two words of the dividend, a special timing chain, involving DIVT0 through DIVT4, is provided. The timing chain logic is also used for floating-point division. FT9 sets the adder for the complement of the high-order dividend in the AR. ET0 sets the SC as in IDIVX and also causes DIVT0, which has effect only if the dividend is negative.

DIVT0 swaps the MQ and the complemented AR, thereby leaving the 1's complement high-order dividend in the MQ and the original low-order dividend in the AR. The adder CRY 36 input is turned on to set the adder for the 2's complement of the AR. The DSF7 flip-flop is set to remember that a negative dividend was originally present.

Nothing further takes place until pulse ET2, allowing the adder to set up. Then, if the dividend was originally negative, control passes to DIVT1, which again swaps the MQ and AR (this time the AR is negated),

leaving the correct low-order dividend in the MQ where it belongs, and the complemented high-order dividend in the AR. The AR answer will be correct, unless the low-order part was all 0s, in which case, the 2's complement should propagate a carry all the way to the high-order word. This condition is checked at DIVT1 through DIV LOW ZERO COND, and is true whenever a carry propagates all the way through the adder and the instruction is a fixed-point division, or whenever bits 8 through 35 of the adder are 0 and the instruction is a floating-point division. Because the AD CRY 36 was previously set, it is cleared, unless the DIV LOW ZERO COND is true. The other adder enables are set for the positive AR. Hence, DIVT3 either reads the identical AR or the AR+1 into the AR.

Control reaches DIVT4 either from DIVT3, or directly from ET2 if the dividend was originally positive. DIVT4 shifts the sign bit out of the MQ, and then proceeds to DST1 for a fixed-point divide, or returns to additional special logic for floating point.

The divide subroutine consists of DST1 through DST7, and is used for all divides in the KA10. DST1 sets the DSF1 flip-flop, which controls the direction of shifting (to the left) and the conditions governing AD MD+ and AD MD-. If the result of a division step is positive (AD0 (0)), the magnitude of the divisor should be subtracted from the dividend; if the result of a division step is negative (AD0 (1)), the magnitude of the dividend should be subtracted from the divisor. Because the divisor is in 2's complement form, addition and subtraction of the divisor must be exchanged when the divisor is negative. Hence, BR0 is effectively exclusive ORed with AD0 to determine whether an addition or subtraction must be made. DST1 sets the adder gates to initially subtract the magnitude of the BR from the AR. After a suitable delay, the sign of the initial result is checked. If the result is still positive, the divisor is too small, relative to the dividend to hold the answer in the current word size; therefore, DST7 sets the appropriate overflow flip-flops and proceeds immediately to ST9, bypassing the store cycle, so that the initial values are still available in the ACs or memory to provide data for determining the cause of the overflow.

If the first result is negative, DST2 calls the SC subroutine, which performs the next 35 steps of the division (27 for floating point). The bits of the quotient are the complement of the adder sign bit and enter MQ35 when the combined AR and MQ are shifted left. The shift pattern is shown on the "SCAF" flow chart. At the completion of the shift and count sub-

routine, control returns to DST3, which shifts the last quotient bit into the MQ, and loads the last remainder bit into the AR, this time without shifting. The adder enables are once more set according to AD MD+ and AD MD-. At DST4, the remainder is corrected if the last step of the division made the remainder negative. DST4 also sets up to give the remainder the correct sign; that of the original dividend, stored in DSF7. At DST5, the remainder, having the correct sign, is loaded into the MQ, and the quotient is brought back to the AR from the MQ. If this was a fixed-point divide, the adder is set up to negate the quotient if it is necessary. For a floating-point divide, the adder is set to round the quotient. Control then returns to either the fixed or floating-point divide logic, whichever is appropriate.

The fixed-point divide logic ends at DIVT5, at which time the negative quotient is loaded into the AR if the exclusive OR of the original divisor and dividend signs were 1. Control then passes to ST0 and the store cycle.

4.4.2 Block Transfer

The BLT instruction is unique in that it may make a large number of memory cycles and, hence, needs to be interruptable at any point. When interrupted, the routine must be able to restart where it left off. The flow for BLT is on the "SCBT" flow diagram.

The BLT instruction initially specifies the PC + 1 INH condition so that the PC is not incremented until all words in the block have been transferred. The FCCACLT level causes the fetch cycle to pick up the first data word, whose address is in the left half of the specified AC. At FT9, the AR contains the pointer word from the AC with its halves swapped (to, from), the MQ contains the first data word, the MA contains the first "from" address, and the BR has the final address (the effective address of the BLT). FT9 sets the adder enables, which are used at ET2. ET0 swaps halves of the pointer word back to the original AR configuration (from, to). At ET1, the "to" address is loaded into the MA. Then, at ET2, the data word is brought back to the AR from the MQ, and the pointer, with both its addresses incremented, is read into the MQ from the adder. A memory "write" cycle is requested and the control goes to the memory subroutine.

The memory control returns to BLT T1, which sets up the test by bringing the final "to" address back to the AR from the MQ and clearing the left half of the AR. The adder is set to subtract the BR from the AR. Be-

cause CRY 36 was cleared by BLT T1, the first time AD17 will be zero is when the final address equals the current "to" address. Because no bits are present in the left halves of the AR or BR, AD17 just shows the carry-out of the right half. When BLT instruction is complete, control goes to BLT T2, which increments the PC and goes to ST1A, allowing the PC to settle before going to the next instruction.

If more words are to be transferred, the next pulse is a BLT T3, which clears the AD BR-EN so that the adder controls are the same as they are during a fetch cycle. The full-pointer word is loaded back into the AR, and the priority interrupt system is checked for waiting interrupts. If no interrupt is present, control returns to the fetch cycle at FT6, the pulse which prepares for FCCACLT. If an interrupt is waiting, control passes to the store cycle via ST0, which stores the pointer word back into the AC.

4.4.3   Byte Manipulation

The byte manipulation group, part of the KE10 extended-order code option, consists of five instructions: LDB, DPB, ILDB, IDPS, and IBP. Byte instructions are performed in two parts: updating the byte pointer word and generating a mask, and isolating the actual byte in the data word. The flow of all byte instructions is shown on the "BYTF" flow diagram.

The two parts of the byte instructions are distinguished by the state of the BYF5 flip-flop; this flip-flop is 0 for the first part and 1 for the second part. The levels BYTE PTR INC and BYTE PTR NOT INC divide the byte instructions into two groups during the first part of byte instructions, depending on whether or not the pointer is to be incremented. The byte instructions can be interrupted during the middle of a byte operation. The BYF6 flip-flop is set at the end of the first cycle, where pointer incrementing occurs. Should an interrupt occur before the second part can take place, the state of flip-flop BYF6 is stored in memory along with the PC. When the interrupt is dismissed, BYF6 is restored and the byte instruction restarted. However, on the second time through, incrementing the pointer again would be erroneous; therefore, flip-flop BYF6 causes ILDB and IDPB to become part of the BYTE PTR NOT INC group. The IBP instruction does not involve the above problem because it is completed in only one part.

Byte instructions begin with the BYF5 flip-flop off (shown at the left side of the BYTF flow diagram). During the fetch cycle, the index and indirect parts

of the instruction register are enabled to receive the same parts of the pointer word, when this word is retrieved from memory. If the pointer word is to be incremented it is fetched, using the FCE PSE. The PC is not incremented on any but the IBP instruction, and the ST INH flip-flop holds off the store cycle until all the byte logic steps are complete. The adder will be set to increment the AR for BYTE PTR INC instructions. Control leaves the execution chain at ET0.

The BYTE PTR INC instructions go to BYT1, which loads the SC with the position part of the pointer, from AR bits 0 through 5. SC-EN calls for the second input of the SCAD to be subtracted from the SC. This second input is selected to be AR bits 6 through 11, the size part of the pointer. If the result is negative, the pointer has moved off the right end of the word, and will have to be moved on to the next word; hence, control goes to BYT2. BYT2 loads the AR with its old contents plus 1, and clears the SC. BYT3 loads the SC with 44 (octal), the position corresponding to the leftmost bit in the word; this becomes an input to the SCAD by virtue of its being in the SC. Either this new position, or the one computed from BYT1, is loaded into the AR from the SCAD output at BYT4, thus replacing the original position. The SC is cleared, and the pointer is written back into memory. If this has been an IBP instruction, it has been completed, and the memory control will return to ST1A in the store cycle. If the instruction has been an ILDB or IDPB, control goes to BYT7 from the memory control.

The BYTE PTR NOT INC instructions bypass BYT1 through BYT4, and join at BYT7, after going through BYT6 to set the SCAD enables in the same configuration as they were at BYT1. BYF4 serves the dual function of being the return control for the memory subroutine and for the shift-and-count subroutine when making up the byte mask. BYT7 loads the SC with the negative of the size because the size has been available in its negative form from the SCAD for some time. BYT7 then calls the shift-and-count subroutine, which shifts 1s in the MQ, to the left, making a mask of as · many bits as the byte size calls for. BYT7A loads the SC with the position of the byte and sets the SCAD to negate it. Flip-flops BYF5 and BYF6 are set, to indicate that the second part of the instruction is to be entered, and that incrementing has been performed (if called for). The main adder enables are set in the same manner as by IT0 and control is returned to IT1 where the effective address of the byte pointer is computed. This is possible because the index and indirect bits of the IR are loaded during the earlier fetch cycle.

If, during the address calculation, an interrupt occurs the byte operation is aborted, but the state of the BYF6 flip-flop is preserved, as previously described. If no interrupt occurs, the second part of the byte instruction takes place; this time, the instruction is divided into BYTE LOAD (ILDB and LDB) and BYTE DEPOSIT (IDPB and DPB). The BYTE LOAD instructions fetch the contents of E and go immediately into the shift-and-count subroutine from ET0, which also picks up the negative of the SC from the SCAD. SCT3 shifts the AR (which contains the byte desired) to the right, moving the rightmost bit into bit 35. The LBT1 pulse causes the mask in the MQ to be ANDed into the AR, preserving only those bits of the AR that are part of the desired byte. The BYF6 flip-flop is cleared and control is sent to the store cycle for the completion of the instruction.

The BYTE DEPOSIT group of instructions starts out in a fashion similar to the BYTE LOAD group of instructions. However, this group also fetches the AC. The shift-and-count subroutine is used to shift the mask and byte in the AR to the left, simultaneously positioning both byte and mask. The desired result is that the C(E) is unchanged when the mask has 0s and contains the bits of the AC when the mask has 1s.

DBT1 changes the adder enables to the condition of BR+ only. At DBT2, the three main registers are permuted such that the AR contains the mask, the BR contains the byte, and the MQ contains the result word. The adder is then set for the complement of the AR, or mask. At DBT3, the mask, in the AR, is ANDed into the BR, with the result that extraneous bits in the original AC are set to 0; consequently, the only 1 in the BR will be in the desired byte. The AR is loaded from the adder, producing the complement of the mask. The adder enables are then changed again to give the BR+. At DBT4, the MQ is ANDed into the AR, thereby preserving only 1 bits only in the area of the result word where the byte is no longer. Then, at DBT5, the byte is ORed into the AR, from the adder. Hence, the function performed is: (Byte ∧ Mask) ∨ (C(E) ∧ ~Mask). This group of instructions then returns control to ST0, where the result is stored back in E. This instruction group does not use FCE PSE because it is possible to hold up the memory for several microseconds during the shifting operation.

4.4.4 Floating Point

The floating-point instructions are the second half of the KE10 extended order code option. There are four primary subgroups in the floating-point logic, and three additional special instructions. The four primary subgroups are FAD, FSB, FMP, and FDV, and the three special instructions are UFA, FSC, and DFN. The UFA instruction is covered with the FAD group. In each of the main groups, rounding and immediate mode are options.

4.4.4.1 Floating Add and Subtract - This group includes all variations of FAD, FSB and UFA. The "FAF" flow diagram covers the operation of this group of instructions. The basic scheme of operation of all instructions in this group is as follows.

a. Find the operand with the larger (in magnitude) exponent.

b. Subtract the magnitude of the smaller exponent from the larger exponent.

c. Shift the fraction of the number with the smaller exponent to the right by the number of places found in step b.

d. Add or subtract the two fractions.

e. Unless performing a UFA instruction, renormalize the fraction of the result by shifting the fraction to the left until its magnitude is between 1/2 and 1, and subtract 1 from the exponent of the result for each step of normalization.

f. Combine the resultant exponent (which is the original, larger exponent, less the number of steps taken in normalization) with the normalized fraction.

The FAD and FSB instruction groups each has an immediate mode in which the effective address is used as the operand; however, because the significant part of a floating-point number is in the left half, FT8 in the fetch cycle swaps the two halves of the AR, providing an immediate operand having significance in the left half, and 0 in the right half. FSB uses the main execution cycle to negate the memory operand, which starts in the BR. The operands are interchanged between the AR and BR, but this is immaterial because A-B = -B+A.

Instruction UFA causes the fast memory address decoder to advance to the second accumulator at ET0 because the UFA has the added function of storing its result in the second AC. All three operations join at FAT1.

Because floating-point numbers are in 2's complement form, a complication results. This complication is that the exponents of negative floating-point numbers are in 1's complement form. (Because a fraction must have some bit set, the +1 inherent in 2's complementation does not propagate all the way down to the exponent part of the word; hence, the exponent, considered as a separate number, will be in 1's complement form.)

FAT1 begins computation of the difference in magnitude of the exponents by loading the AR exponent into the SC. If the two operands are of the same sign, the SCAD is set to subtract the BR exponent from the SC. If the operands are of different sign, the SCAD is set to add the exponent in the BR to the SC. The main adder is set to hold the BR+. At FAT2, the result of the previous operation is loaded into the SC. The desired result is the 2's complement difference of the exponent magnitude because this result is needed to control the shift-and-count loop. At FAT2, four classes of result are possible.

a. If the result is negative (and the signs of the AR and BR are equal), the correct result already exists because a 2's complement subtraction was performed. In this case, all SCAD functions are disabled, causing the true SC to appear at the SCAD output when the next time pulse occurs.

b. If the result is positive (and the signs agree), the result must be 2's complemented to obtain the correct negative difference.

c. If the result is negative (and the signs are different), a 1's complement number is added to the positive number and, hence, a 1 must be added to the result to get a 2's complement result.

d. If the result is positive (and the signs are different), the result is too small by 1; therefore, 1's complementing this number produces a correct 2's complement result. (Note: $-(x+1) = (x+1)' + 1 = x' + 1' + 1 = x'$ where ' = 1's complement.)

At FAT3, the final correct difference is loaded into the SC, and FAT3A is produced if the sign of the SC is the same as the sign of the AR. FAT3A causes the AR and BR to be interchanged, bringing the number with the smaller exponent into the AR. This is true because the sign being examined in the SC is that loaded by FAT2 and is the result of a straight subtraction of exponents. If the signs are equal, then the number originally in the AR is larger because the subtraction does not change its sign; hence, the numbers in the AR and BR must be swapped.

The branch following the delay after FAT3 determines whether it is necessary to shift the smaller number to the right. If the exponents differ by an amount greater than 54, all significance in the smaller number is shifted off the end of the MQ, so that it is not necessary to take time to shift. Because the number 64 is easier to test for, the logic tests whether the SC has a negative number of the form 7XX or a positive number (which can only be 0 because of the logic from FAT1 through 3) before operation goes to FAT5 and the shift routine. FAT4 occurs when this condition is not met, and clears the AR and SC to set up the adder for the add to come. FAT5 spreads the AR sign over the AR exponent because the exponent would cause problems if shifted with the rest of the AR. With this operation done, the fraction in the AR is effectively a 36-bit, fixed-point number. Bits leaving the right end of the AR register during floating-point shifts enter bit 8 of the MQ because the high-order bits of the MQ are used for determining the exponent in long-mode operations. The AR sign is unchanged, thereby propagating itself into all vacated bits to the left of the fraction. The shift path is shown on the "SCAF" flow diagram.

When shifting has been completed, control returns directly to FAT6 (where FAT4 also goes). FAT6 picks up the BR exponent, the exponent of the answer. The SC will have been cleared by FAT4, or the shift-and-count subroutine, so the SC+BR setup gives just the BR exponent. When FAT7 occurs, the exponent is loaded into the SC, and the exponent part of the BR is once more smeared with the BR sign to make the BR number appear as an effectively fixed-pointed number. The SCAD is set to complement the SC (in case the BR was negative and the exponent was in 1's complement form). FAT8 finally performs the actual addition of fractions and, if necessary, picks up the complement of the resulting exponent so that the exponent is always positive when going into the normalization operation. Control then passes to the common normalization routine used for all floating-point operations.

4.4.4.1.1 Normalization – The normalization logic, shown on flow diagram "NRF", performs several functions for the floating-point operations.

a. The normalization logic takes care of the case where the sum of two fractions exceeded 1 in magnitude, thereby overflowing into bit 8.

b. This logic brings the first significant bit into bit 9 from the right, if any significant bits exist in either the AR or MQ.

c. For all but division operations, this logic handles rounding.

d. This logic inserts the exponent into the result word.

e. This logic fixes up (modifies) the MQ for long mode, except for division.

The normalize logic begins at NRT0 (called NR for normalize – return because it returns to the main timing sequence). The SCAD is set to count up by one, and the main adder is set to have the AR+ enabled. After a delay, if the AR result (shown in the adder) is all 0s, and either MQ bits 9 through 35 are all 0, or if this is a floating-point divide operation, the NR ALL ZERO condition is true. In this instance, control is sent directly to NRT99 and then to STO. If the NR ALL ZERO condition is false, control passes to NRT1. In addition, NR SH RT COND will also be true if the AR sign differs from AR bit 8 (the overflow bit for the fractional addition), or if bit 8 is set and bits 9 through 35 are all 0. This latter condition is necessary to detect the case of exactly –1 as a fraction, which must be turned into –1/2 with an increased exponent. When NR SH RT COND is true, the AR is shifted right one place, and the incremented exponent is loaded into the SC from the SCAD. The MQ is also shifted, unless in a floating-point divide instruction, because, in that case, the MQ does not contain an extension of the quotient, but rather contains a remainder.

For all non-trivial cases, flow proceeds to NRT1, which sets the SCAD to complement the SC. This step is necessary for normalization operations going to the left because the exponent needs to be decremented. Exponent decrementing is accomplished by first complementing, incrementing as necessary, and then recomplementing. The output of the delay after NRT1 combines with the output of the delay from NRT2 and tests for the NR NORMAL condition. A number is normalized when AR0 is different from AR9, or the fraction is 400 000 000 (i.e., AD9 (1) $\wedge$ AD 10 through 35 = 0); also, the UFA instruction forces the NR NORMAL condition to be true.

If the number in the AR is not yet normalized, the NRT2 delay loop is circled (each time, loading the SC from the SCAD), shifting the AR left and shifting the MQ left if not in FDV. The first time through NRT2, the complement of the exponent is loaded into the SC; however, because NRT2 does a SCAD SC+1 SETUP, all successive times through the loop give increments. When the number is normalized, control passes to NRT3, which performs the final increment

into the SC and sets up to recomplement the exponent. If the exponent was normalized to begin with, the NRT3 pulse picks up the complement, which was set up at NRT1.

Following the delay after NRT3, the rounding condition is examined. NRF1 is set when rounding has already occurred, or when rounding is not desired, when it might otherwise occur; in particular, during FDVR, rounding is done by a different algorithm. The NR ROUND condition is NRF1 (0) AND IR6(1) AND MQ8(1) AND NOT (MQ9-35 = 0 AND AR0(1)). In this situation, IR6(1) indicates that an "R" mode instruction is present, MQ8(1) indicates that the magnitude of the fraction in the MQ is at least 1/2 and that, if the AR is negative, the magnitude of the fraction in the MQ is not exactly half. The last condition (NOT (MQ8-35 = 0 AND AR0(1))) is necessary because negative numbers truncate in the more negative direction in 2's complement. An additional consequence of 2's complement arithmetic is that the fraction in the MQ is always positive, so that a negative number in the AR (which is inexact and therefore has additional bits in the MQ) is an additional "one more" negative. As an example, consider the number –1 in the AR: 777 777 777 777. Then –1-1/4 is 777 777 777 7776 in the AR and 001 400 000 000 in the MQ, using MQ8 as the most significant bit. This number, when rounded, becomes –1 and, hence, 1 has to be added to the AR. Note that MQ8 was a 1. By a similar argument, –1-3/4 should become –2. In this case, the MQ would contain 000 400 000 000, with MQ8 a 0, which will not cause the AR to be changed. The MQ holds 001 000 000 when –1-1/2 is represented. Here, MQ8 is a 1, but the desired result is –2. This special case is picked out by the logic to prevent incrementing of the AR.

If NR ROUND is true, control proceeds to NRT6, which loads the SC with the recomplemented exponent (now in positive form), sets the adder to add one to the AR, and sets NRF1, indicating that rounding has occurred. NRT7 loads the AR with the incremented fraction, and returns control to NRT0. This loop is necessary since the fraction may no longer be normalized because the one which has been added may have caused overflow in bit 8. The easiest way to handle the problem is to proceed through the entire renormalization process.

If NR ROUND is not true, control goes from NRT3 to NRT4. NRT4 loads the positive exponent into the SC, and if the AR is positive, disables all inputs to allow SCAD to hold the true number in the SC. The overflow and underflow conditions are checked on the

contents of the SC, prior to being recomplemented. If SC0 is 0, then the true exponent would have a 1 in bit 0, and would, therefore, be invalid. This condition sets the overflow flag and the floating-point overflow flag. If both SC0 and SC1 are 0, then the exponent underflowed, meaning that the sum of two exponents is negative. NRT5 returns the exponent to the AR, via the SCAD. If the AR is negative, then the SCAD SC COMP enables are left on at NRT4, and the complement of the exponent is used.

Following NRT5, floating-divide instructions return to their own special timing chain. Long-mode floating-point operations, other than FDVL, proceed to NLT0. All other operations are completed. NRT99 provides a common exit for the normalize subroutine and then returns to ST0.

The "long" mode of the floating-point operations returns a second accumulator which contains additional significance for addition, subtraction, and multiplication, and the remainder of divisions. The division case is handled separately. The major task of the "NL" timing chain is to give the low-significance word an exponent that is 27_{10} less than the exponent of the word in the AR.

NLT0 sets the SCAD to subtract 33_8 (27_{10}) from the exponent in the SC. At NLT1, this exponent is loaded back into the SC, and the high-order result is temporarily moved to the MQ. If the reduced exponent is still positive, the MQ (which contains the low-significance bits) is brought into the AR; if the reduced exponent underflows, the AR is cleared. The underflow of the low-order exponent is not considered an error. NLT2 shifts the AR to the right one place, moving the most significant bit into bit 9. All floating-point shift operations use bits 8 through 35 of the MQ to preserve significance and, hence, the result must be moved right one place at the end. The SCAD enables are cleared by NLT2, leaving the true low-order exponent at the SCAD outputs. At NLT3, the exponent is loaded into the AR if the fraction is non-zero. Lastly, NLT4 interchanges the MQ and AR, returning the result to its proper position. Control then passes to NRT99 and ST0.

4.4.4.2 Floating-Point Multiply – The steps that perform floating-point multiplication are:

  a. Compute the resulting exponent as the sum of the magnitudes of the exponents of the operands, less 200 (octal).

  b. Perform multiplication, using the standard multiplication algorithm.

  c. Normalize the result, using standard NR logic and "long" mode logic, if necessary. The exponent calculation is a common routine used for both floating-point multiply and divide instructions.

4.4.4.2.1 Floating-Point Exponent Calculation – Floating-point exponent calculation is complicated by the fact that exponents can be either true or complemented, depending on the sign of the full word containing them. The problems here are similar to those encountered in the floating-point add exponent situation. For floating-point multiply, the FP routine is entered directly from ET0, but floating-point divide instructions may use additional logic before reaching FPT0. FPT0 loads the SC with the exponent of the AR operand. If the AR is negative, the SCAD is set to complement the SC to get the true exponent. At FPT1, the true exponent is loaded into the SC, and the BR input to the SCAD is enabled. The condition FP EXP ADD, determines whether the BR exponent is added to, or subtracted from, the SC. The FP EXP ADD condition is true for divides if the BR is positive, and is true for multiplies if the BR is negative. Unfortunately, this signal is misnamed because, if FP EXP ADD is true, the exponents are subtracted, and, if false, the exponents are added. The SCAD +1 EN is set for divides because the desired result is the difference between the magnitudes of the exponents, and if the FP EXP ADD condition causes the exponents to be added, it is because the BR is negative. Therefore, the exponent in the BR is in 1's complement form and needs a 1 added to get a 2's complement result. For the subtraction case, the addition of 1 is necessary in 2's complement as usual.

FPT2 loads the result of the previous calculation into the SC and cuts off the BR inputs to the SCAD, enabling the number 200_8 instead. The 200_8 is needed because exponents are kept in excess 200 notation (which makes it possible to use fixed-point comparison operations on floating-point numbers). If two exponents are added, it is necessary to subtract out 1 of the excess 200; if exponents are subtracted, the excess 200 must be added again. For divisions, the SC+EN is set to add the 200, and, for multiplies, SC-EN causes the 200 to be subtracted (in 2's complement). At FPT3, the exponent parts of the AR and BR are reduced to sign bits because no further use is made of them, and having them as sign bits simplifies the

multiplication or division that follows. The final exponent is loaded into the FE register from the SCAD, where it stays while the SC is being used to count iterations of the multiply or divide. For multiplication, the SC is cleared prior to being loaded with the number of shifts. For division, the exponent is loaded into the SC because it may have to be manipulated once more. The SCAD is set to increment the SC, and main adder enables, which may have been set earlier in division, are cleared. Multiplication flow proceeds to FMT1 and division flow to FDT1.

4.4.4.2.2 Multiplication and Normalization - After returning from the floating-point exponent calculation routine, FMT1 sets the SC to 745, which calls for 27 (decimal) shifts. This is the number of significant bits in floating-point fractions. Because the shift path between the AR and MQ enters the MQ at bit 8, 27 shifts are sufficient to align the product correctly in the AR and MQ. The adder BR + EN is set (prior to permuting the operands); this is necessary to get the memory operand into the MQ and the AC operand into the BR. At this point, floating-point multiplication joins the fixed-point multiplication logic through SCT4. This operation has been explained in detail in Paragraph 4.4.3. After SCT4 (this is a floating-point multiply), control returns to pulse FMT3, where the final result in the adder is loaded into the AR, and the last multiplier bit (contained in MQ35) is cleared. The SC is 0 on return from SCT4, so the exponent can be ORed into the SC prior to going to the normalization return routine (which completes the floating-point multiplication).

4.4.4.3 Floating-Point Divide - The floating divide instructions in the PDP-10 System are the most complicated in the machine. The FDVL instruction (floating divide long) has considerable additional complication because it requires a double-length operand and gives a remainder as part of the result. The following steps are taken as part of a floating divide.

Step	Procedure
1	Take the magnitude of the dividend, remembering its original sign.
2	Compute the exponent of the quotient.
3	Test to determine if the magnitude of the dividend is greater than the magnitude of the divisor; if it is, shift the dividend right one place and increase the quotient exponent by one.

Step	Procedure
4	Set the shift count for the divide loop, taking one extra step when rounding is called for.
5	Call the divide subroutine to perform the actual division.
6	Round the quotient on the basis of an extra bit developed in the quotient, when called for.
7	Normalize the result, using the normalize subroutine.
8	Recomplement the quotient, if necessary, and return to ST0, unless the instruction is FDVL.
9	Refetch the dividend to get its exponent.
10	Subtract 27 (decimal) from this exponent (or 26 if step 3 above, was necessary).
11	Affix the exponent to the remainder unless the exponent underflowed or the remainder is zero.
12	Return the results to the usual registers and return to ST0.

The description of the floating divide routine is shown on the "FDVF" flow chart. The FDVL instruction fetches both ACs, and uses the early part of the fixed-divide timing chain (DIVT1 through DIVT4) to take a two-word magnitude. This operation can be followed in the description of fixed-point divide in Paragraph 4.4.1.4. All other floating-point divide instructions have a one-word operand whose magnitude is taken by FT9 and ET0. At this point, all floating divide instructions go through the floating-point exponent calculation subroutine previously explained in the floating multiplication description, Paragraph 4.4.4.2.1. The floating-point exponent calculation takes the difference between the magnitude of the exponent of the dividend and that of the divisor. After returning from FPT3, FDT1 sets up to subtract the magnitude of the divisor from the dividend to check for overflow. For the remainder to be correct, the number of quotient bits which must be developed to yield a normalized result must be precomputed. Assuming normalized arguments (in the range 1/2 to 1), the quotient must lie between 1/2 and 2. If the dividend is larger than the divisor, yet still normalized, the quotient must lie between 1 and 2. If this is so, an equivalent answer is obtained by shifting the dividend to the right

one place and increasing the exponent of the quotient by 1, yielding a fractional quotient between 1/2 and 1. Through this stratagem, the quotient of any two normalized numbers will itself be normalized, and hence, the remainder will be correct.

If bit AD0 is 0 following FDT1, the result of the subtraction is still positive, indicating that the dividend in the AR was larger; hence, FDT2 is generated, resetting the adder to hold only the AR, and clearing the FE. FDT3 shifts the entire dividend in the AR and MQ right one place and also reloads the FE with the increased exponent. The SCAD was previously left holding the increased exponent by FPT3. Flip-flop FDF3 remembers that this step was taken, for later use in handling the exponent of the remainder for a FDVL instruction. The two paths merge at FDT4, which clears the SC in preparation for FDT5. FDT5 loads the count of the number of shifts to make during actual division. An extra step is called for if the rounded mode is used. This extra bit is squeezed out later after it is used to round the result.

FDT5 calls the division subroutine (whose operation is described in Paragraph 4.4.1.4). Following DST5, control returns to FDT6 (which achieves rounding because DST5 previously set the adder to add one to the quotient). FDT6 shifts the quotient right one place because one extra quotient bit is developed when rounding is indicated. If this bit is 0, the plus 1 would have made the bit a 1; however, the 1 would be shifted off the right end anyway. If the extra bit is a 1, then a +1 would have incremented the quotient. FDT6 also retrieves the exponent of the quotient from the FE and sets NRF1 to prevent the normalizing routine from rounding. Control is then given over to the normalizing routine, for the following reasons.

a. If the operand was un-normalized, the result will be normalized.

b. The normalizing routine has the logic required to insert the exponent into the quotient. For floating divide instructions, the normalizing routine returns control to FDT7 from NRT5. FDT7 sets up to negate the quotient and, at FDT8, this negative is taken if one and only one of the original operands is negative. The adder enables are set back to the true AR only. After FDT8, control is sent to ST0 for non-long-floating divides because the correct quotient is now in the AR.

FDT9 moves the quotient to the BR, clears the AR, and clears the SC in preparation for again retrieving the original dividend. This step is necessary because

the remainder must be given an exponent of as many powers of two less than the dividend as there were steps of division. The AC is retrieved in the usual way by FDT10 or by the memory subroutine. At FDT11, the exponent of the dividend is loaded into the SC. The remainder is brought into the AR from the MQ, and the SCAD is set to either add or subtract 32 or 33 (octal). Addition occurs if the dividend and, hence, its exponent, is negative; subtraction occurs if the dividend and, hence, its exponent, is positive.

The number 32 is selected if the FDF3 flip-flop is set, indicating that the dividend was shifted right one place at FDT3. FDT12 reads the exponent back into the remainder, if the exponent still has the same sign as the remainder, and the remainder is non-zero. If the sign associated with the exponent changed, the exponent underflowed, and, hence, the remainder is set to 0. FDT13 returns the remainder to the MQ and prepares the adder to show the quotient in the BR, which is returned to the AR at FDT14. Control then returns to ST0.

4.4.4.4 Floating Scale - The floating scale instruction multiplies floating-point numbers by powers of two by adding the power of two to the exponent. The instruction flow is found on the "FSDN" flow chart. ET0 begins the operation by loading the SC with the effective address, taking bit 18 as the sign, and bits 28 through 35 as the rest of the significance. The AR, containing the AC, is moved to the BR because the input gates to the SCAD come from the BR. Depending on the sign of the AR, the SCAD is set to either add or subtract the exponent (which will be the BR to or from the SC. This is done because the exponent may be either in true or 1's complement form. If the number is negative, and the exponent is complemented, the usual +1 associated with subtraction is disabled because the number is in 1's complement form. At ET1, the exponent part of the AR is cleared to sign bits (a requirement of the normalization logic). The new exponent is loaded into the SC at ET2 and control passes to the normalize return logic (where the number is normalized if necessary and the exponent part is re-inserted into the word). An additional reason for using normalization logic is that the exponent will not be inserted in a word with a zero fraction.

4.4.4.5 Double-Floating Negate - This instruction was necessitated by the double-precision floating-point format used by the PDP-10. In this format, the low-order word is kept with positive sign and exponent, independent of the sign of the high-order word.

To negate a double-precision quantity, it is necessary to negate only the fractional part of the low-order word and then complement the entire high-order word, propagating a carry from the low-order fraction when necessary. This instruction is shown on the "FSDN" flow chart.

In DFN, the AC contains the high-order word, and the C(E) contains the low-order word. These arguments are in the AR and BR, respectively. FT9 sets the adder for the 2's complement of the BR and sets the SCAD for the exponent of the BR. At ET0, the low-order exponent is loaded into the SC, the negated low-order part is loaded into the AR (from the adder), the high-order part is moved from the AR to the BR, and, if the fractional part of the adder (bits 9 through 35) is not all 0, the carry 36 input to the adder is cleared. This last operation causes the carry to continue to propagate into the high-order word only if the low-order fraction is all 0.

ET0 also clears all miscellaneous inputs to the SCAD so that the contents of the SC are available at the SCAD output. At ET1, the exponent is restored to the low-order word, unchanged. ET2 moves the low-order word back to the BR, and the complemented (and possibly also incremented) high-order word is brought back to the AR, thus completing the operation.

4.5 KA10 CONSOLE KEY LOGIC

The logic associated with the console is known as the KEY logic. This logic is primarily concerned with the "action" keys: stop, reset, start, continue, XCT, examine, examine next, deposit, and deposit next. Logic is included to allow these functions to be repeated at varying speeds (when appropriate). The KEY operations can be followed on the "KO" flow diagram.

4.5.1 STOP Key

The simplest action is caused by the STOP key. This key, through an initial transient detector, generates the KST 1 pulse providing KEY RUN CLEAR which clears the RUN flip-flop. The machine is given a 100-μs delay to stop (which it usually does). The machine examines the RUN flip-flop between instructions. After this 100-μs delay, the address time chain is broken by IF0 being held in the 0 state for 100 μs by the KEY AT INH delay. Because all instructions (except BLT) must go through that point in the logic at least once every 100 μs, the machine is likely to stop.

4.5.2 RESET Key

The RESET key is similar to the STOP key, except that it may be repeated and, therefore, has a flip-flop which holds the information that the key has been pressed. Flip-flops are provided for all keys whose functions may be repeated. Pressing the RESET key first clears KEY REPT SYNC, stopping any key functions. KEY RESET is an input to the KEY MANUAL initial transient detector, which gives a single transition from potentially bouncy mechanical contacts. KEY MANUAL triggers a 30-ms delay to allow bouncing to stop, and then gives KEY FCN STROBE. This signal reads the state of the repeatable action keys into a set of flip-flops. It is assumed that only one key at a time was pressed.

After 1 μs, KT0 occurs. This pulse starts the repeat delay if the REPEAT BYPASS switch is set. REPEAT BYPASS causes repeatable actions to be reinitiated even if they never reach a normal conclusion. If RUN is on during a RESET operation, control passes to KST1, as if STOP had been pressed. Following the second 100-μs delay (KEY AT INH), KST2 gives the MR START pulse, which clears all internal control states and I/O device control registers.

4.5.3 REPEAT Key

If the REPEAT key is on when RESET is pressed, the KEY REPT SYNC flip-flop is set at KEY FCN STROBE (or KT0, too). KST2 gives the KEY DONE pulse, which either clears the key function flip-flops if KEY REPT SYNC (0), or triggers the KEY REPT DLY if KEY REPT SYNC (1). If the REPEAT switch is off, KEY REPT SYNC is cleared. The repeat loop is closed by the off-going transition of the repeat delay, which triggers KT0 if KEY REPT SYNC is still on. If not, KEY FCN CLR is generated, which clears the key function flip-flops.

4.5.4 START Key

The START key operation goes through the KEY MANUAL logic, as described above for RESET. KT0 triggers KT0A, a logically identical pulse but, electrically, a B-series pulse. If the RUN flip-flop is set, the START function is not allowed, and control goes immediately to KEY FCN CLR. If RUN(0), KT1 EN causes entry into the KT1 through KT4 chain, which, for the START operation, gives MR CLR, clears the AR, reads the address switches into the MA, then reads the MA into the PC, and turns on RUN, exiting through KT4 to IT0 to start a normal instruction.

4.5.5 CONTINUE Key

The operation of the CONTINUE key depends on how the machine came to a stop, if, indeed, it has stopped. The KEY MID INST STOP level is true when the machine has stopped in the middle of an instruction; i.e., MC STOP (1) or SC STOP (1). If MC STOP (1), MCRST0 is given to restart the machine at KT0A. If SC STOP (1), SCT1 is triggered from KT0A. RUN is set on CONTINUE if KEY MID INST STOP is false. Unless RUN (0), and KEY MID INST STOP was false, the CONTINUE function is complete, and KEY DONE occurs. In the remaining case, the instruction sequence needs to be restarted, and, therefore, KCT0 is generated followed by a delay to allow RUN to settle on (it was turned on at KT0A) and then KT4 returns to the main instruction sequence.

4.5.6 EXAMINE, DEPOSIT, and EXECUTE Keys

EXAMINE, DEPOSIT and EXECUTE can be considered as a group known as KEY SYNC OPS, because their function occurs, even while the machine is running, by being so synchronized as to have their effect occur between instructions. When RUN (0), these keys go through the normal KT0-through-KT3 timing chain. When the RUN (1), however, they set the KEY SYNC RQ flip-flop at KT0A, and then wait for the following sequence. The next FT9 sets the KEY SYNC flip-flop, followed by an ST9 when EXCTF (0) diverts the normal path from ST9 to IT0 into KT1, which then performs the required action.

EXAMINE reads the address from the address switches into the MA, and then proceeds to make a memory read request. MI PROG is cleared, allowing the new reference to show in the MI lamps. DEPOSIT clears the AR and then reads in the DATA switches, and picks up the address switches into the MA. It then makes a memory write request. The KEY F1 subroutine return flip-flop inhibits relocation on key-controlled memory references. EXECUTE causes the DATA switches to be read into the AR, and then transfers the AR to the IR by doing an MC WR RS, sending the data over the memory bus. The IR LT and RT enables are set by KT2. Because the instruction cannot be restarted in case of interrupt, PIs are inhibited by the KEY PI INH flip-flop for one instruction. EXECUTE enters the normal time chain via KT3A at IT1, where the processor treats it as though it has just retrieved a new instruction from memory.

EXAMINE NEXT and DEPOSIT NEXT are allowed only when the machine is stopped because the examine or deposit address information is kept in the MA. These operations increment the MA, and then perform the corresponding simpler operation. The only complication is that the MA cannot be directly incremented and, therefore, the key-next time chain, KNT1 through KNT3, is employed to use the PC counting circuit. Because the original PC cannot be lost, it is saved in the AR when the MA (examine or deposit address) is sent to the PC at KNT1. KNT2 increments the PC (examine or deposit address) and brings the AR (original PC) back to the MA. KNT3 sends the PC (examine or deposit address, incremented) to the AR, and restores the original PC from the MA. KT1 retrieves the examine or deposit address, incremented to the MA from the AR.

4.5.7 READ IN Key

READ IN operation follows the "RIMF" isolated flow diagram. The READ IN key puts the machine in a special mode for initially loading programs into a cleared machine. It is allowed only when RUN (0). At KT0A, an IOB RESET pulse is sent to all I/O devices by MR START, and a 1.5-μs delay is set at KT0. The IR is loaded with a DATAI to location 0 of the device set in the READ IN DEVICE switches. At the end of the KEY RDI DLY, mentioned above, the IOT RDI PULSE is sent, selecting the device whose code was just read into the IR, and initiating that device to perform its read in operation. Each time the selected device has a word of data, it pulses the IOB RDI DATA line, causing the KT1-through-KT3A time chain to run, exiting to IT1. On the first data word, the same DATAI instruction is executed (for a second time, although the first one has no effect), reading the count and address into location 0. On successive IOB RDI DATA pulses, the disappearance of the IOB RDI DATA pulse sets the KEY RDI PART 2 flip-flop. A BLKI to location 0 is loaded into the IR, which uses the pointer word read in earlier to read in data from the selected device.

4.6 KA10 MEMORY CONTROL

4.6.1 Memory Subroutine

The memory control section of the KA10 is a hardware subroutine with multiple entries. Its function is to communicate with the memory system over the memory bus and to sort out request involving the optional KM10 fast memory. A complete description of the memory bus itself is found in the PDP-10 Interface

Manual (DEC-10-HIFB-D). The reader should become familiar with the bus system before proceeding.

The memory control flow is shown on the "MCFM" flow diagram. The calls to the memory control are shown at the top of this drawing. They fall into four broad groups:

a. Read Request

b. Read/Write Request

c. Write Request

d. Read/Write Restart

The fast memory-read request is used when it is necessary to fetch a word which would always be located in the fast registers (if they were available), but, in this instance, they are either not implemented or disabled. Therefore, the main memory system is used for the first 20_8 locations. This group of requests is further characterized by the fact that the addresses required are not in the MA, but are available at the output of the fast memory-address mixer.

The MC FM RD RQ signal sets the MAI FMA SEL flip-flop, which forces the high-order 14 bits of the address presented to the memory bus to be zeros, and selects the FMA mixer as the low-order 4 bits. This type of request then becomes an ordinary read request.

4.6.2 Read Cycles

The MC RD RQ PULSE sets the read flip-flop and clears the write flip-flop, signifying a simple read cycle. It also clears the AR, which receives the data. The next pulse, MC RQ PULSE, does the following:

a. Clears the MC STOP

b. Clears the MC PAR STOP

c. Clears the PB (parity buffer), which will be used to compute the parity of the incoming data word

d. Sends a pulse to the priority interrupt system to synchronize requests to that system.

If the machine is not in USER mode, or if the address in the MA is less than 20_8, the MC RQ flip-flop is set 45 ns after the MC RQ PULSE. The MC RQ flip-flop is what actually starts the memories on the bus.

If the machine is in USER mode, 140 ns are allowed for the relocated address to set up. Then the MC RQ is set only if the address is valid (-PRA ILL ADR). If the address is illegal, the MC ILLEG ADR pulse is fired, setting the CPA MEM PROT FLAG, and aborting the memory cycle. In this instance, control returns to ST9, unless this was an instruction or address calculation fetch, in which case, control returns to IT1.

The MC RQ signal only reaches the memory bus if two other conditions are fulfilled:

a. At least one of the read or write requests is set

b. Either the fast registers are disabled or this is not a request for a location below 20_8.

Assuming these conditions are fulfilled, the request (now called MC REQ CYC) is sent to the bus. The processor now stops and waits for a response from the memory; in particular, address acknowledge (called MAI CMC ADR ACK in the drawings). This pulse, regenerated as MC ADR ACK, clears the MC RQ flip-flop and the parity flip-flop. On a read cycle, no further action results from this pulse.

After the arrival of ADR ACK, the memory should send the data bits, along with RD RS. If neither the MC STOP nor the MC PAR STOP flip-flops were set (by the MC STOP SET pulse, which occurs after a delay from the MC RQ PULSE), MCRST0 (memory control restart, time 0) is generated from MAI CMC RD RS. MAI CMC RD RS is reshaped as MC RD RS, which is then delayed 140 ns to allow the parity network to set up. If the console PARITY STOP switch is on, the MC PAR STOP flip-flop is set, requiring verification of correct parity before triggering MCRST0. If the parity is acceptable (-PN PAR EVEN OR MC IGNORE PARITY(1)) and MC STOP (0) and MC PAR STOP (1), MCRST0 occurs after the delay.

If the parity is even, the MC PAR ERR pulse occurs, setting the MC STOP flip-flop if the PARITY STOP switch is on, causing the console MEMORY STOP lamp to be on when the machine stops. The CPA PAR ERR flip-flop is set on any parity error, independent of the parity stop setting. The PARITY STOP switch, therefore, gives two choices: proceed rapidly, but set the "parity error flag", causing an interrupt on a parity error; or wait until each word has its parity checked before proceeding, and stop on an error. The latter mode is normally used only for maintenance, or if memory failures are suspected as the cause of an otherwise unexplained problem.

Following MCRST0, a 65-ns delay allows the data to settle in the AR (and possibly IR) before control is returned to the calling routine by MCRST1.

An alternate path for a read cycle occurs if the address is not in the range of existing memory: a nonexistent memory reference. This is handled by an integrating one-shot (R303) set for 100 µs. If this time goes by following a memory request with the MC RQ flip-flop still on and the MC STOP off, the MC NON EX MEM pulse is fired. This pulse sets the CPA NON EX MEM flag, and if the console NXM STOP switch is on, sets the MC STOP flip-flop again to light the MEM STOP console indicator. Control proceeds to the MC NXM RST pulse if the NXM STOP switch is off. This pulse acts as if ADR ACK is present. In the case of a read cycle, MC NXM RD occurs, and if MC STOP has not been set for some other reason, it simulates an RD RS, triggering MCRST0. In this case, the machine proceeds as if the word addressed contains zero; however, the NON EX MEM flag causes an interrupt, if enabled.

The remaining type of memory read cycle addresses one of the first 20 registers, assuming the KM10 fast memory is enabled. For this type of request, the MC REQ CYC must remain false to avoid signaling the core memory system that a word is required; this gating is provided, as explained above. As was previously mentioned, the fast registers also have their own addressing network, independent of the MA; hence, it is necessary to feed the MA outputs to the fast memory address inputs. This is accomplished by a signal called FMA MA EN, which overrides the normal FMA (fast memory address) source. FMA MA EN is the AND of MC RQ (1), MA 18-31 = 0 and MC FM EN. If this signal is true after a delay following the setting of MC RQ, then the request is not going out on the bus, but allows FMAT1 to fire, loading the AR from the fast memory, if in a read cycle.

The next problem is that the word retrieved from the fast memory may also be required in the IR. Since gates from the fast memory are provided only to the AR, the memory bus data lines are used to convey the information to the IR, if required. After a delay following FMAT1, FMAT2 hits MC ADR ACK, clearing the MC RQ. Because it has been postulated that FMA MA EN is true, the MC WR RS pulse is generated and causes the MC MEM BUS FM AR pulse, which takes the contents of the AR and puts it on the bus. Normally, this function is used only during a write cycle, but here it serves to route the information to the IR. Because no memory on the bus can be logically connected to this processor at this time, pulsing various

bus lines has no effect on the memory system. MC WR RS feeds into MCRST0, which is in the normal return path.

4.6.3 Write Cycles

Normal write requests enter through the MC WR RQ PULSE. In the case where the fast registers would be used if functioning, the MC FM WR RQ input is available to set the MAI FMA SEL, such as is done by the MC FM RD RQ. The MC WR RQ PULSE sets the MC WR and clears the MC RD flip-flops, calling for a simple write cycle. The flow-through to MC RQ SET is the same as for a read cycle, including the check for illegal addresses and the MC STOP SET.

In a write cycle, the only response from a memory is the "address acknowledge", which becomes MAI CMC ADR ACK, and then is fed to a PA to reshape it into a standard pulse called MC ADR ACK. This pulse clears the MC RQ and PB, as in the read cycle, and, then because MC WR (1) and MC RD (0), goes on to MC WR RS. This pulse generates MC MEM BUS FM AR, which actually strobes the AR flip-flops onto the memory bus and into the PB (parity buffer), since it has permanently enabled input gates from the memory bus. After a delay to allow the parity network to settle, the MC BUS WR RS signals the memory to go on with its cycle, because it now has the data from the processor. Simultaneously, if the parity of the 36-bit word was even, the MC PAR PULSE is sent, causing the 37th bit to be set. In the meantime, if MC STOP has not been set, MCRST0 is triggered by MC WR RS. MCRST0 generates MCRST1, which restarts the calling routine.

For a write cycle which addresses fast memory, the situation is identical to a similar read cycle, except that FMAT1 does not load the AR from the fast memory but, instead, hits the FMA FM FM AR(J) (fast memory address fast memory from AR jammed) pulse.

4.6.4 Read/Write Cycles

The read/write cycle is used where the processor ex-. pects to read a word and send back a new word to the same address within a microsecond or so. Since this ties up the memory until the new word is sent back, precautions are taken to avoid tying up the memory for extended periods. The MC SPLIT CYC SYNC flip-flop determines whether read/write cycles are to be split into separate read and write cycles. This flip-flop is cleared at the beginning of each instruction by MR CLR, and is set at IT1 for any of the following reasons:

a. KEY ADR STOP is true, allowing the possibility of stopping the machine in the middle of an instruction

b. KEY SING CYCLE is true, for the same reason

c. IOB DR SPLIT is true, indicating that an I/O device cannot even tolerate the extra microsecond that may be squeezed into the memory cycle, because it has its own memory port, which needs rapid access

d. MC FM EN is false, indicating that core memory has to be used for the first 20_8 locations. Because the KA10 fetches accumulators after it fetches memory operands, it is possible that the memory module containing the AC needed will be hung up due to a previous read/write cycle.

The MC SPLIT CYC SYNC may also be set at FT1, if MA 18-31=0 because read/write cycles are suppressed in this instance, and this flip-flop carries the information through to the store cycle in which a read/write cycle was changed into separate read and write cycles.

When a read/write cycle is successfully initiated by FT1, the MC RD/WR RQ PULSE sets both the MC RD and MC WR flip-flops, and clears the AR. The cycle proceeds as a normal read cycle, returning through MCRST1, which clears MC RD. During the subsequent part of the instruction, the memory is proceeding to that part of its cycle where it must have the data to write into the cores. If that point is reached, the memory stops and waits for write restart.

At some later time, the KA10 indicates that the data to be written is available by causing the MC RD/WR PULSE. If the MC SPLIT CYC SYNC is on, the read part of the intended read/write cycle was turned into a simple read cycle, and the write part must be done as a simple write cycle; therefore, control goes to the MC WR RQ PULSE. If MC SPLIT CYC SYNC is off, the MC WR RS pulse is given, and control returns to the main sequence through MCRST1, as usual.

4.6.5 Memory Indicator Register

The memory indicator register (MI) usually follows the contents of the register addressed in the address switches. This is accomplished by a comparator, which dynamically compares the address selected by the memory address interface (MAI) with the AS (address switches), and the FMA with the AS. On memory references going through the memory control subroutine, MCRST0 samples the AS COND, which is true if the MAI comparator is true and the FMA select is false, or if the FMA comparator is true and the FMA select is true. On a read reference, MIT0 is squeezed in to allow a delay for the AR to set up; otherwise, MIT1 occurs immediately.

The MI PROG flip-flop indicates that the MI has been loaded by the DATAO PI instruction, and that automatic MI loading should be suspended; therefore, the MI LOAD pulse occurs after MIT1 only if MI PROG is a 0. The direct fast memory references are handled similarly, using only the AS=FMA comparator.

The MI PROG DIS switch on the maintenance panel allows the program-loading feature of the MI to be disabled by holding the MI PROG flip-flop off and disabling the MI PROG EN signal.

4.6.6 Address Stop or Break

The address stop feature allows the operator to cause the machine to stop in a selected mode whenever a given location is referenced. The address break feature is similar, except that a PI request is made rather than a stop. Three "address condition" switches are on the console: instruction, data fetch, data write. These translate to hardware terms as follows:

a. Instruction means that IF0(1)

b. Data fetch means that IF0(0) and MC RD(1)

c. Data write means that MC RD(0) and MC WR(1).

The OR of these conditions makes MC SW COND. AS=RLA (the archaic name for the AS=MAI comparator) ANDed with KEY ADR STOP and MC SW COND (ORed with KEY SING CYCLE) makes MC STOP EN, which sets MC STOP after a delay following the MC RQ PULSE.

In the case of address break, the AND of MC SW COND, AS=RLA, KEY ADR BREAK and MC RQ (1) generates a level transition which sets the CPA ADR BREAK flip-flop which will eventually cause an interrupt.

4.6.7 Input/Output System

The input/output system operates over the I/O bus, which is a cable system similar to the memory bus. A

complete description of the I/O bus is found in the PDP-10 Interface Manual (DEC-10-HIFB-D), knowledge of which is assumed below.

4.6.8 I/O Instructions

The flow of the eight I/O instructions is found on the "IOTF" flow diagram. BLKO and BLKI are special in that they go through the address, fetch, and execute cycles twice. Furthermore, the action of BLKI/O depends on whether the instruction was executed in normal program sequence or as a result of a priority interrupt (as the instruction in the PI location). BLKI/O first picks up the pointer word during the fetch cycle on a read/write request. At FT9, the adder is set to increment both halves of the pointer. At ET0, the new pointer is returned to the AR and, if the left half did not overflow (AD CRY0(0)) and if this instruction was not a PI instruction (PI CYC(0)), the PC is incremented, causing a skip. If overflow occurred, and PIOV EN was true (indicating PI CYC(1) or in read in mode), PI OV is set, causing the interrupt not to be dismissed. IR12 is set at ET0, changing this instruction into the corresponding DATAI/O instruction; i.e., BLKI becomes DATAI and BLKO becomes DATAO. ET0 also clears EX ILL OP, which fixes the following problem: a user program (running in user mode) executes a non-relocated UUO (op codes 0, 40-77); between the time the UUO is stored in 40 and the instruction in 41 can be executed, a PI occurs; the PI does a BLKI/O instruction, which does not overflow; if EX ILL OP remained set (from the UUO), the program would resume from the C(PC) unrelocated, because relocation is suspended by EX ILL OP. Lastly, ET0 sets IOT F1, a subroutine flip-flop, and calls MC RD/WR RS to return the pointer to memory. When the memory control returns, IOT T1 clears IOT F1, and returns to the end of the address cycle, AT3. The instruction is then treated as if it were always a DATAI/O instruction, with the right half of the incremented pointer as its effective address.

The remaining six I/O instructions (other than BLKI and BLKO) fetch operands as needed, and set the IOT GO flip-flop at ET0. As soon as the IOT RESET DELAY is false, IOT T0 is generated, starting the IOT special timing chain. The reset delay allows the bus to be reset to its quiescent state of -3V, without tying up the processor unless two I/O instructions come too close together. IOT T0 starts two one-shot delays: the initial setup delay and the IOT RESTART DELAY. The initial setup delay, lasting 1 μs, fires IOT T2 when completed. This pulse clears IOT GO and sends the first pulse associated with CONO and

DATAO, the CLR pulse. In another microsecond, the IOT RESTART DELAY times out, generating IOT T3, which causes the second pulse for the outward IOTs: CONO SET or DATAO SET. IOT T3B, a B-series pulse, occurs simultaneously with IOT T3, reads the I/O bus into the AR, and sets the adder BR+EN (reading the data switches into the AR, if the instruction were a DATAI APR). The IOB DATAI and IOB STATUS (called IOB CONI on the I/O bus) signals, which tell the I/O devices to place their data on the bus, are generated during the combined durations of the IOT RESTART DLY and the IOT DATA DLY. The IOT DATA DLY starts at IOT T2 and lasts 1.5 μs, giving a total of 2.5 μs, during which data appears on the bus. IOT T4 occurs at the end of the IOT DATA DLY. It causes the AR to be ANDed into the BR on CONSO and CONSZ instructions. Any other type of I/O instruction now proceeds directly to ST0. CONSX instructions take an extra time pulse, IOT T5, which increments the PC if the skip condition is met. The I/O bus is driven back toward -3V by the bus reset gates for the duration of the IOT RESET DLY, less the overlap time of the IOT DATA DLY of 0.5 μs, giving a reset time of 2 μs.

4.7 KA10 PRIORITY INTERRUPT SYSTEM

The priority interrupt (PI) system allows an I/O device to interrupt the normal sequence of instructions. There is no flow chart specifically for the PI system, because little sequential logic is involved.

Interrupt requests on the I/O bus may appear at any time and therefore must be synchronized in the processor to be sure that no new request disrupts the computation of which request to honor. The PI system is equipped with three registers: PIO (PI ON), PIR (PI Request) and PIH (PI Hold). There is also a master enable called PI ACT, and two auxiliary control flip-flops called PI CYC and PI OV.

The incoming interrupt requests, called IOB PI n, are sampled by the PIR STB (PI request strobe) pulse, which occurs at every MC RQ PULSE when not already in a PI cycle. The PIRn flip-flop is set when a request is true if the corresponding PIO flip-flop is on (indicating that the programmer wishes to honor requests from that channel) and if the corresponding PIH flip-flop is off (indicating that an interrupt is not presently in progress on that channel).

The processor determines whether it should take a PI cycle by examining the PI RQ signal, normally at IT1, which occurs just after retrieving an instruction or an

indirect address. PI RQ is also examined during the BLT instruction; however, if PI RQ is true, BLT will first terminate in an orderly fashion, and the request will be picked up by IT1. PI RQ is the OR of the seven PI REQ n signals, ANDed with PI CYC(0) (to insure one instruction is being completed for a given interrupt before a higher priority channel can interrupt) and KEY PI INH(0) (which suppresses interrupts during instructions executed by pushing the console EXECUTE key).

The determination of which of several waiting interrupt requests should be serviced is handled by a priority chain known as PIOK n. PIOK n will be true if neither the PIH nor PIR flip-flops of the channel of immediately higher priority are set, and the PIOK for that channel is also true. The highest-priority channel uses PI ACT(1) as its enabling condition. If PIOK is true for a given channel, its PIR flip-flop is set, and PIH flip-flop is clear (which is redundant because the PIR could not possibly be set unless the PIH was clear), the PI REQ n will be true.

The PI REQ n's feed the PI RQ gate and, also, a unary-to-binary converter (encoder) which comes up with a binary address from zero to seven. When the processor gets to IT1, if PI RQ is true, PI T0 comes next. PI T0 sets PI CYC and allows a delay for the MA to clear from IT1, and PI CYC to settle. Control returns to IT0, where the PI channel encoder address will be ORed into the MA, along with the 40_8 bit, and possibly the 100_8 bit, if the MA TRAP OFFSET switch is on.

If the instruction at the interrupt address is not an IOT (for instance, a JSR), matters are simple: PI HOLD will be true because PI CYC(1) and -IR IOT are true; therefore, at FT9, PIH FM PICH RQ will occur, and the PIH flip-flop associated with the channel whose PI REQ was honored will be set, clearing the PIR flip-flop. For a control-type I/O instruction, the machine will hang indefinitely, since neither PI HOLD nor PI OV ever become true. A DATAI or DATAO causes both PI HOLD and PI RESTORE to be true and, therefore, the PIH FM PICH RQ pulse will occur, clearing the PIR; then the PIOK CLRS PIH pulse will clear the PIH at ET0, thereby dismissing the interrupt. PI CYC is cleared by ST1.

The remaining cases involve BLKI and BLKO. If the count of a BLK IOT overflows, PI OV will be set at ET0. After the BLK IOT turns into a DATA IOT, ST9 will fire PI T0 again, instead of going back to IT0 first. PI T0 will clear the MA, and then IT0 will read the

PI channel encoder again; however, PI OV supplies a 1-bit in bit 35, thereby moving to the second interrupt location associated with the particular channel. The new instruction must not be an IOT for PI HOLD to be true. If this condition is true, the PI OV and PI CYC will be turned off at the end of the instruction, and the PIH flip-flop will be on, and PIR off. If the count of the BLKI or BLKO does not overflow, it is treated just as if it was a DATAI or DATAO; namely, the PI HOLD and PI RESTORE will both be true, allowing the processor to proceed from the state prior to interruption.

When an interrupt has been held, it may be dismissed by a JRST with IR9(1), causing PI RESTORE, which clears the PIH on the highest priority channel whose PIH is on.

4.7.1 User Mode Logic

User mode logic is available as either the KT10 or KT10A options. The KT10A option includes a dual protection and relocation system. Each system includes a user mode flip-flop (found on the EX print and called EX USER) which indicates when protection of memory and instructions is in effect. Since EX USER must be set between instructions, an EX MODE SYNC flip-flop is provided, which is set by JRST, either with bit 12 on, or when restoring the flags, with flag bit 5. If EX MODE SYNC is set, the next MR CLR will set EX USER.

When EX USER is set, the IR decoders turn 7XX (IOT) op-codes and JRSTs, which attempt to halt the machine or restore the PI system into UUOs, unless EX USER IOT is set. The user IOT flip-flop allows these otherwise illegal instructions to happen because it may be desirable in some circumstances to have only memory protection. This flip-flop can only be set by flag bit 6, when not already in user mode; however, it can be cleared by a 0 in bit 6 when restoring the flags in any mode.

The EX ILL OP flip-flop tells that an illegal operation has been performed (including all standard monitor call UUOs 040-077). When EX ILL OP is on, or EX PI SYNC is on (during a PI cycle), EX TRAP COND suppresses relocation, thereby forcing absolute addressing. If the monitor program does the expected subroutine call, the EX USER flip-flop will be cleared at ET0 of that instruction, after its old state has been preserved with the flags.

Relocation is controlled by the EX REL signal. This signal is generated twice to supply enough drive to all the gates using it, without taking the time to invert the signal twice. Relocation is in effect whenever EX USER (1) AND EX TRAP COND is false and the MA does not address an accumulator (since ACs are always in fixed lower memory) and MAI FMA SEL (0) (since, if that flip-flop is on, the address is to come from the FMA mixer) and KEY F1(0) (since that flip-flop indicates that the console examine or deposit function is operating).

The relocation adder(s) is always computing the sum of the MA and the relocation register(s).

In the simple KT10 option, the EX REL signal determines whether the MAI (memory address interface) bits come from the output of the relocation adder, or directly from the MA. With the KT10A option, if EX REL is true, and if the MA is less than or equal to the PR (the lower protection register), the RLA adder signals are selected; the RLB adder signals are selected if the MA is greater than the PR.

In the simple KT10, an address is valid if it is less than or equal to the PR. In the KT10A, the valid condition exists if the MA is either less than or equal to PR, or greater than or equal to 400000, and less than or equal to the PRB (upper protection register) and either the PR COR PROT (write protection) bit is off, or a write or read/write is not being made. Stated symbolically, this condition is:

$$\text{VALID} = (MA \leq PRB) + [(MA \geq 400000) \cdot (MA \leq PRB) \cdot (PR\ WR\ PROT\ (0) + MCWR\ RQ(0))]$$

CHAPTER 5
BASIC I/O DEVICES

5.1 PAPER TAPE READER

The paper tape reader is part of the Digital PC09.
The PC09 consists of a PC02 Perforated Tape Reader
and a PC03 Perforated Tape Punch, with a SCR punch
motor control added. The PC02 mechanism is de-
scribed in the PC02 Instruction Manual
(DEC-08-I2BA-D). The PC03 is a Model 500 Royal-
McBee (Roytron) tape punch. Its mechanism is de-
scribed in the Royal-McBee maintenance manual.
Some additional interface information for the PC02
and PC03 (which alone make up the PC01 Paper Tape
Reader/Punch) is found in the PDP-8/S Paper Tape
Reader/Punch Control combined manual
(DEC-00-IP1A-D, DEC-00-IP2A-D, DEC-00-IP3A-D).

The paper tape reader portion of the PC09 is arranged
to operate in two modes: binary, in which six char-
acters of tape are buffered to form a 36-bit word; and
alpha, in which each 8-bit character is sent to the
processor.

The control for the reader is found on three drawings:
PTR1, 2 and 3. The first drawing shows the fairly
standard I/O device control register, plus the PTR
TAPE flag logic. The PTR TAPE flip-flop indicates
to the program that tape is in the reader. If tape is
present, PTR TAPE SYNC will be pulled off at some
time shortly after each PTR MOTOR SHIFT. If
PTR TAPE SYNC is off by the next PTR MOTOR SHIFT,
PTR TAPE will be turned on. If the TAPE SYNC re-
mains on through a whole motor-shift cycle, then
PTR TAPE will be turned off. When a tape is loaded,
pushing the manual PTR FEED SWITCH will cause
PTR TAPE to be set, setting the PTR DONE flag. This
will request an interrupt, if a channel has been
assigned.

In operation, the PTR RUN signal is generated by
either the PTR FEED SWITCH or PTR BUSY (1). As-
suming the reader has been idle for at least 40 ms,
the PTR ENABLE flip-flop will be set. This enables
the clock circuit, which has an integrator controlling
the clock speed, since the stepping motor cannot start
at full speed. The first clock pulse (PTR CLK) will
sample the PTR RUN and, assuming it is true, will not
affect PTR ENABLE. Since the normal stop position of
the reader is between characters, the PTR DATA
PHASE will be false; therefore, the PTR STROBE

PULSES will not occur. The PTR MOTOR SHIFT ad-
vances the 2-bit counter PTR A and PTR B, and turns
on PTR POWER. PTR A and B count in a sequence
00 10 11 01 00, causing the stepping motor to ad-
vance one half of a character on each pulse.

At the next PTR CLK pulse, the motor will have ad-
vanced so that the character is over the read head.
PTR DATA PHASE will be true, enabling PTR STROBE,
if either the reader is not in binary mode or hole 8 is
seen. The PTR STROBE pulses will read the data into
the buffer and advance the PTR CNT counter. In
alpha mode, PTR LAST is always true, while in binary
mode, PTR LAST becomes true after five counts;
namely, prior to the sixth character. At the PTR
STROBE, when PTR LAST is true, PTR DONE is set
and PTR BUSY is cleared.

When the program does a DATAI, the PTR BUSY will
be turned on again. Each time BUSY comes on, the
PTR CLR pulse clears the buffer and counter. As long
as BUSY is again set within one-half character time
(i.e., before the next PTR CLK pulse), the reader
continues at full speed. If a PTR CLK pulse comes
when PTR RUN is false (hence PTR BUSY (0)), PTR
ENABLE will be cleared, inhibiting the delayed clock
pulse from causing PTR MOTOR SHIFT. Furthermore,
the PTR SHUTDOWN delay will be triggered, inhi-
biting PTR ENABLE from being set again for 40 ms.
At the end of the shutdown delay, if PTR RUN is still
false, PTR POWER will be cleared. If PTR RUN had
become true during the delay time, PTR ENABLE will
be set when PTR SHUTDOWN becomes false.

The PTR buffer is arranged to shift by groups of six.
The seventh and eighth bits from the right (bits 28
and 29) are also used for holding the two extra bits
when in the alpha mode.

5.2 PAPER TAPE PUNCH

The paper tape punch is part of the Digital PC09. It
is a Roytron punch, with SCR motor control. The
punch sends sync signals to the logic to time each
character. The control has both binary and alpha
modes; however, the only difference between the two
modes is that binary forces the eighth hole to 1 and
the seventh hole to 0. The control logic can be found
on drawings PTP1 and PTP2.

The PTP1 drawing shows the standard device control
register and the motor control logic. The object of
the motor control is to run the motor only when punch-
ing, but to keep it running for 5 seconds after the last

punching operation, in case more punching is to occur. If the motor has been shut down, either pushing TAPE FEED or setting the PTP BUSY flip-flop turns on both the 5s delay and the 1s delay. The 5s delay drives the PTP SCR DRIVER as long as the machine is not power-clearing. When the 1s delay times out, PTP SPEED will be true, indicating that the motor is up to speed. The 5s delay will remain on for 5s after the TAPE FEED button is released, or when PTP BUSY was last on.

The PTP2 drawing shows the data buffer and punch-magnet driving logic. When TAPE FEED is pushed, and BUSY is not on, a PTP DATAO CLR pulse clears the buffer. Each PTP SYNC pulse arriving while PTP SPEED is true generates a 10 ms PTP SYNC DEL, which enables the punch magnet drivers. During TAPE FEED, hole 8 is suppressed by PTP BUSY being off.

Under computer control, the action is similar to that described above; however, the data is loaded by DATAO SET. The PTP DONE PULSE is given at the end of the 10 ms PTP SYNC DEL, which clears BUSY and sets DONE.

5.3 TELEPRINTER CONTROL

The teleprinter control is designed to operate any teletypewriter using ASCII (USASCII) code conventions at 110 and 150 baud. The logic is found on drawings TTY1 and TTY2.

The TTY1 print shows the control register, which is straightforward. The only unusual thing is that the BUSY and DONE flags are set and cleared by different I/O bus bits.

The receiving logic, shown on the TTY2 print, has two primary inputs: TTY RCV EIA and TTY RCV LINE. Because of their different interface characteristics, these inputs are used for Teletype® Models 37 and 33/35, respectively. A third input comes from the transmitter logic and is gated with the TTY TEST flip-flop for purposes of checking the logic in a closed

loop. The definitions at the right of the drawing are useful in determining what conditions should exist on various signal lines. TTY INPUT remains false during the idle condition. When TTY INPUT becomes true, a START bit has arrived, and TTI ACTIVE is set. This gates the TTI CLK, which gives pulses at the desired bit rate. The clock output is actually a square wave, which switches true halfway through each bit, so that the bit can be sampled. The on-going transition of TTI ACTIVE fires TTI CLR, clearing the character assembly register. If, at the first TTI SHIFT pulse, the TTI INPUT has become false, then TTI ACTIVE is turned back off because the condition could only be caused by a noise pulse.

After the start bit and eight information bits have been read in, TTI STOP will be set, turning off TTI ACTIVE and setting the TTI DONE flag. The bits as read into the computer are the complement of the actual states at the TTI buffer, because the "start" bit is held therein as a 1, while it corresponds to a 0.

The output section has the option of putting out one or two units of "stop" following the eight code bits. The 10-unit option corresponds to 1 stop unit, and is selected by the TTY 10 UNIT SW. A character is transmitted by loading it into the TTO buffer by a DATAO instruction. The TTO ENB and TTO STOP bits are always set to 1 by the DATAO SET. The buffer is assumed to be clear from the last operation, or an I/O reset. TTO 1 is cleared by the DATAO CLR because it will be left on in the 10 unit mode. The output clock runs continuously and samples TTO ENB. At the first pulse following the setting of TTO ENB, TTO ACTIVE is set, turning on TTO LINE, giving a start bit. The next nine or ten clock pulses give TTY SHIFT pulses, sending the character out to the Teletype. Two true periods before it is desired to stop the shifting, TTO EMPTY becomes true and the next pulse clears TTO ACTIVE. The TTO FLAG is set when TTO ACTIVE goes off, although the last (or only) stop unit is still on true. If the computer returns with a DATAO before that time unit is over, characters will be sent at the maximum rate.

® Teletype is the registered trademark of Teletype Corporation.

This chapter describes maintenance and troubleshooting procedures for the KA10 Central Processor.

6.1 TROUBLESHOOTING

The first step toward fixing a reported fault is to locate it. Therefore, in a hardware-software system environment such as the PDP-10, it must be determined whether the problem lies in the hardware, software, or both. The only practical way of doing this is to establish cooperation between the programmer and maintenance personnel until it is established to the satisfaction of both where (and of what type) the error is.

Crucial to troubleshooting almost anything are the ability to reproduce the problem (ideally, at will) and the technique of systematically tracing the problem from its symptoms, step-by-step, back to its source.

A step-by-step procedure should be used to trace the problem back until a point is reached where all inputs (conditions) into an element (of the hardware or software) are proper, but the output is improper. The element thus located must be at fault and should be repaired. Where necessary, or desired, the element itself may be subjected to step-by-step fault location (from output to input) until the internal source of the problem is found.

Depending on circumstances, it may be desirable to use DDT, the diagnostic programs, or margining (as described later) to aid in the location of faults.

6.2 TEST EQUIPMENT

Special tools and test equipment required for maintenance are listed in Table 6-1. Except for DEC equipment, suggested commercial brands are given for purposes of specification only, and do not constitute exclusive endorsement.

Table 6-1
List of Maintenance Supplies

Name	Model
Field service kit FLIP CHIP module extender	DEC Type 142 DEC Type W980 or G998
Oscilloscope	Tektronix Type 453, or equivalent
Punch lubricants	Teletype KS7470 oil; Mobile-grease #2
33, 35, 37 KSR lubricants	Teletype KS7471 grease
Cleaning kit	DEC Type PN A425484D

6.3 PROCESSOR TEST PROGRAMS

6.3.1 Description

A series of test programs provide a complete check of the processor logic. They presume no major malfunctions in the core memory and operator's console. Table 6-2 lists the KA10 test programs.

Functionally, the programs fall into two categories, diagnostic and reliability. The diagnostics, test programs A through H, isolate genuine go/no-go type of hardware failures that are easily recognizable. The reliability programs isolate failures that are more difficult to detect because they are marginal in nature and/or occur infrequently or sporadically. The family of test programs are written so that, when run successively, they test the processor, beginning with small portions of the hardware and gradually expanding until they involve the entire machine. To accomplish this, they are built around instructions and portions of instructions whose demands upon processor capabilities progress from simple transfers and

Table 6-2
Processor Diagnostic Programs

Document Number *	Description
MAINDEC-10-D0AA	Test A Basic Instruction Diagnostic (MOVE and SKIP)
MAINDEC-10-D0BA	Test B Basic Instruction Diagnostic (MOVE, TEST, Half Word Instructions)
MAINDEC-10-D0CB	Test C Basic Instruction Diagnostic (Boole and PC Sensitive)
MAINDEC-10-D0DD	Test D PI System Instruction Diagnostic**
MAINDEC-10-D0EB	Test E Shift Rotate Diagnostic
MAINDEC-10-D0FC	Test F Fixed Point Mul/Div Diagnostic
MAINDEC-10-D0GB	Test G Floating Instruction Diagnostic (and BYTE Instructions)
MAINDEC-10-D0HC	Test H FMP, FDV, DFN
MAINDEC-10-D0IA	Test I Basic Instruction Reliability Test (ROTates)
MAINDEC-10-D0JA	Test J Basic Instruction Reliability Test
MAINDEC-10-D0KB	Test K Basic Instruction Reliability Test (Add-Substract and JFFO)
MAINDEC-10-D0LB	Test L Basic Instruction Reliability Test (Memory and Both Modes)
MAINDEC-10-D0MD	Test M Basic Instruction Reliability Test (PC Sensitive Instructions)
MAINDEC-10-D0NE	Test N Reliability Test for Fixed, Floating, and BYTE Instructions
MAINDEC-10-D0OC	Test O Automatic Block Transfer Test (BLT)**
MAINDEC-10-D0PA	Test P KT10A Protect and Relocate Diagnostic**
MAINDEC-10-D0QA	Test Q KT10A Reliability Test**
MAINDEC-10-D0RA	Test R Random Number Instruction Set**
MAINDEC-10-D0ZC	Test Z Processor Timing Test (SPEEDY)
MAINDEC-10-D1EA	Test 1E KT10 Relocation and Protection Reliability and Diagnostic Test **
MAINDEC-10-D1FA	Test 1F User Mode BLT Test
MAINDEC-10-D1GA	Test 1G Fast Memory Test
MAINDEC-10-D2AD	Test 2A Console Teletype Test **
MAINDEC-10-D2B0	Test 2B Paper Tape Reader/Punch Test

*Exact program number changes as programs are revised.
**Program can run only in executive mode.

skips to the most involved data manipulations and arithmetic computations. As portions of the system are proved operable, they become available to succeeding tests for use in checking out unproven portions of the machine.

The test programs are made up of numerous self-contained routines. In those programs that are diagnostic in nature, each routine is involved with a specific circuit or logic function. In the simplest form, for example, a separate routine is used to check each leg of an AND gate. When the diagnostics (A through H) have been run to completion, the processor has been exercised to the extent that it is proved capable of executing all instructions. However, such proof is conditional because it is based on the execution of instructions using pre-established constants as operands. Further tests are necessary to establish that the machine properly executes instructions using operands and various combinations of operands other than those used in the diagnostics. The reliability test programs (I through P) provide this additional testing. Primarily, each routine in the reliability test programs establish a loop whereby a specific instruction or group of instructions is repeated many times. Each repeat is executed using operands whose magnitudes are established by a pseudo-random number generator. This procedure insures machine capabilities being checked under a maximum number of unique conditions.

When an error is detected in diagnostics A through D, the program halts at the end of the unsatisfied routine. The reason for the halt may then be determined by using the console controls and indicators, maintenance switches, and the program listings included in the software package.

For diagnostics E through H and the reliability programs, various indications of errors can be selected with the 36 data switches on the console. The choice of indications include: halt-on-error, proceed-on-error, print-on-line printer (or on teleprinter if the line printer is not available), and ring teleprinter bell. The use of the data switches for selecting error indications is defined in the MAINDEC write-ups provided with the software package.

If halt-on-error is selected, the disposition of the data switches determines whether or not a printout and/or ringing of the bell occur at the error, and the program halts at the end of the unsatisfied routine. The same things hold true for proceed-on-error except that the program continues on to the next routine. If neither halt nor proceed-on-error is selected, the processor enters a loop whereby it continually repeats the

unsatisfied routine. In this case, various timing pulses and levels are available for oscilloscope display as an aid to troubleshooting.

In such a loop, the bell may or may not ring each time the error is encountered, but the printout always occurs on only the first pass through the routine. Again, console controls and indicators, maintenance switches, and program listing may be used to isolate the malfunction.

The printouts accompanying failures become longer and less specific (in terms of isolating circuits) as the complexities of the programs increase. For example, a printout associated with a malfunction in floating-point arithmetic may show the proper contents of the various registers at each event time within the floating-point calculation. Such printouts are derived by simulating the malfunctioning instruction; this is accomplished by performing the functions required using previously proved instructions.

6.3.2 Recognizing an Error

When an error is detected by diagnostics A through D, the machine comes to a halt. The halt condition is easily recognizable in that the RUN lamp extinguishes, the PROGRAM STOP lamp illuminates, and the displays in the console PC, IR, and MA indicators remain in a fixed configuration.

When an error is detected by the remaining test programs, the processor may enter an error loop. In this case, the RUN lamp remains illuminated, the PROGRAM STOP lamp remains extinguished, and the contents of the PC, IR, and MA indicators still change continually as during the normal running of the program. If a printout or other indication of error has not been selected from the DATA switches, the status of the machine is not always obvious. To overcome this ambiguity, all test programs contain a subroutine that is called whenever the processor enters a loop and does not leave it after either a predetermined time or predetermined number of loops. The loop is then referred to as an error loop. This subroutine establishes a memory location as an error counter and fills it with all 0s. Each time a pass is made through the error loop, the counter is incremented and its contents are displayed in the MI register. Consequently, the display of an up-count in the MI register is an indication that the processor is hung up in an error loop. The rate of the display is an indication of how consistently the failure occurs.

With the processor in an error loop, a memory location other than the error counter can be observed by activating the MI PROG DIS switch. When active, this switch removes control of the MI display from the program and again makes it a function of the MA switches.

Viewing the contents of the iteration counter (MI register) affords further evidence of when the program enters an error loop. One of the initial instructions in each program assigns a memory location as a iteration counter. The counter is set initially to all 1s, and is decremented by 1 each time a pass is made through the program. With each pass, the contents of the counter are automatically displayed in the MI register. The decreasing count is an indication that the program is repeating continuously. If an error occurs, the display changes from a decrementing count to an incrementing count of the error counter.

6.3.3 Typical Diagnostic Check

Figure 6-1 is a copy of a page from the diagnostic A program listing. The routines shown are a portion of several routines that check the adder. Although relatively simple, the adder check characterizes the type of tests undertaken by the diagnostic programs.

The routines successively check a portion of each bit of the AD by MOVEing the series of numbers 1, 2, 4, 8, etc. (which contain exactly one bit) to AC 0, and then adding zero to these single-bit numbers. (The series of numbers was "cooked up" when the program was assembled, by starting at 1 and adding the old number to itself to get the new number in a MACRO, not shown.)

The SKIPN instruction in each routine checks the result of adding the 1-bit number to 0. If, due to a malfunction, the adder loses the 1 bit in the constant, the sum will be equal to 0 and the processor will come to a halt. If the addition is performed properly, the adder generates a number other than 0. In this case, the processor skips to the JUMP .+1 instruction, which jumps to the next routine.

6.3.4 Troubleshooting with Test Programs A through C

The portion of test program A shown in Figure 6-1 is used here as an example of how to interpret and troubleshoot an error detected in one of the diagnostic programs A through C.

Assume that while running program A, the processor came to a halt with 007254 in the PC indicators. This indicates that the routine beginning at address 007251 has failed to run satisfactorily.

The prognosis in the program listing calls for a check of the AD AR + EN gates on the adder. In order to check these gates with an oscilloscope, it is best to place the processor in a loop whereby it continually repeats the failing routine. Proceed as follows:

a. Press the STOP key.

b. Using the appropriate console switches, change the HALT and JUMP .+1 instructions in locations 007254 and 007255 to JRST (254000) to location 007251 (the first location in the failing routine).

c. Select location 007251 with the MA switches.

d. Press the START key.

At this point, you may question the need for changing both the HALT and JUMP .+1 instructions to JRSTs. There is a possibility that the routine may not fail each time it is run. If the JUMP .+1 instruction was not altered, the machine would jump out of the maintenance loop the first time the routine ran successfully. Substituting the JRST instruction for JUMP .+1 insures the processor remaining in the loop whether or not the malfunction is intermittent.

With the processor continuously repeating the malfunctioning loop, apply a scope to the adder circuits. The MIT0 pulse at 1S37D makes an ideal synchronizing trigger. Setting the MA switches to the address of the initial instruction in the routine causes MIT0 to be generated when the memory reference to that location is made. Select an initial time base that produces an oscilloscope sweep at least as long as the period of the routine. When the time at which the malfunction occurs has been narrowed to a specific period within the period of the routine, use other pulses, perhaps the MIT0 of another instruction, as a synchronizing trigger. Generally, the most effective method of using the oscilloscope is to decrease the sweep length continually, in known time segments, until the time base is the shortest possible upon which the error can still be seen.

Making further use of the program listing, it can be determined that the MOVE instruction in the failing routine of this example fetched the constant from location 010525. Consulting the listing for the value of ZZ or the contents of 010525, it can be found that

```
       007232   200000   010522        MOVE    [ZZ]        ;CK THE AD AR+EN
       007233   270000   010466        ADD     [0]         ;GATES ON THE ADDER
       007234   336000   000000        SKIPN
                                       STOP †
       007235   254200   007235        HALT  .             ;INST FAIL, TO SCOPE REPLACE
       007236   320000   007237        JUMP .+1            ;CHG TO JRST BACK
                                     †
                000004   000000        ZZ =ZZ +ZZ

       007237   200000   010523        MOVE    [ZZ]        ;CK THE AD AR+EN
       007240   270000   010466        ADD     [0]         ;GATES ON THE ADDER
       007241   336000   000000        SKIPN
                                       STOP †
       007242   254200   007242        HALT  .             ;INST FAIL, TO SCOPE REPLACE
       007243   320000   007244        JMP   .+1           ;CHG TO JRST BACK
                                     †
                000010   000000        ZZ =ZZ +ZZ

       007244   200000   010524        MOVE    [ZZ]        ;CK THE AD AR+EN
       007245   270000   010466        ADD     [0]         ;GATES ON THE ADDER
       007246   336000   000000        SKIPN
                                       STOP †
       007247   254200   007247        HALT .              ;INST FAIL, TO SCOPE REPLACE
       007250   320000   007251        JUMP . +1           ;CHG TO JRST BACK
                                     †
                000020   000000        ZZ =ZZ +ZZ

       007251   200000   010525        MOVE    [ZZ]        ;CK THE AD AR+EN
       007252   270000   010466        ADD     [0]         ;GATES ON THE ADDER
       007253   336000   000000        SKIPN
                                       STOP †
       007254   254200   007254        HALT .              ;INST FAIL, TO SCOPE REPLACE
       007255   320000   007256        JUMP . +1           ;CHG TO JRST BACK
                                     †
                000040   000000        ZZ =ZZ +ZZ

       007256   200000   010526        MOVE    [ZZ]        ;CK THE AD AR+EN
       007257   270000   010466        ADD     [0]         ;GATES ON THE ADDER
       007260   336000   000000        SKIPN
                                       STOP †
       007261   254200   007261        HALT .              ;INST FAIL, TO SCOPE REPLACE
       007262   320000   007263        JUMP .+1            ;CHG TO JRST BACK
                                     †
                000100   000000        ZZ =ZZ +ZZ

       007263   200000   010527        MOVE    [ZZ]        ;CK THE AD AR+EN
       007264   270000   010466        ADD     [0]         ;GATES ON THE ADDER
       007265   336000   000000        SKIPN
                                       STOP †
       007266   254200   007266        HALT .              ;INST FAIL, TO SCOPE REPLACE
       007267   320000   007270        JUMP .+1            ;CHG TO JRST BACK
                                     †
                000200   000000        ZZ =ZZ +ZZ
```

Figure 6-1 Diagnostic A, Typical Page of Program Listing

the constant contains a 1 in bit 13. Since this constant is being added to 0, the result of the addition should also contain a 1 in bit 13. Therefore, focus the search for the malfunction on the circuits associated with that bit. Use the oscilloscope to determine if:

a. AD AR + EN failed to AND with AR bit 13, or

b. The adder itself failed, or

c. The AR from adder JAM failed.

As can be seen from this example, the program listings contain a great deal of troubleshooting data that may not be as obvious as the prognosis in the comments column. Make certain that full advantage is taken of all such data.

When the malfunction has been found and corrected, change the last two instructions in the maintenance routine back to the original HALT and JUMP .+1 instructions or reload the program. Then rerun the entire program.

To summarize, the technique for troubleshooting an error detected by the diagnostics might be:

a. Consult the program listing.

b. Place the processor in a loop of the failing routine.

c. Scope pertinent circuit areas, selecting various sweep times as necessary.

Using the built-in maintenance features, focus the oscilloscope display on a segment of time that most nearly coincides with the occurrence of the failure. Particularly useful for this purpose are the various "stop" keys: SING INST, SING CYCLE, ADR STOP, ADDRESS CONDITION INST FETCH/DATA FETCH/WRITE, AND SHIFT CNTR MAINT.

6.3.5 Troubleshooting with Diagnostic D

Diagnostic D is similar to diagnostics A through C in that many of its routines force the machine to halt when an error is detected. However, some routines in diagnostic D check the user mode in which HALT instructions are illegal. These routines hang up in a JUMPA . instruction when an error is detected. The processor continues to run, but it repeatedly jumps to

the address of the JUMPA . instruction. The RUN lamp remains illuminated; the PROGRAM STOP lamp remains extinguished; the PC becomes static; and the MA contains the address of the JUMPA . instruction.

To troubleshoot the unsatisfied routine, place the processor in a loop by changing the JUMPA . and JUMP .+1 instruction to JRSTs back to the first instruction in the routine. Then, proceed with the error detection techniques outlined for diagnostics A through C.

6.3.6 Troubleshooting with Test Programs E through M and P

Unless halt-on-error or proceed-on-error is selected, programs E through M cause the processor to continually repeat routines in which errors are detected. Once in a failing loop, the processor repeats the routine even though the failure may be intermittent. In the case of an intermittent failure, the routine may, upon occasion, run successfully. It can be released from the loop only by selecting proceed-from-error at the data switches or by restarting the program.

As with all diagnostics, E through H each checks a unique portion of the machine. When one of these programs detects an error and enters a failing loop, the troubleshooting practices already outlined for tests A through D can continue. However, should one of the reliability programs (I through M and P) fail, it is not always best to begin troubleshooting with the processor hung up in the unsatisfied routine. Instead, identify the failing instruction and return to the diagnostic that checks that instruction under fixed conditions. If the diagnostic does not fail immediately, it might be induced to fail by running under margining conditions. (Techniques for running margins are detailed in a later paragraph.) This procedure of backtracking to a diagnostic is recommended because, generally, the diagnostic programs are easier to troubleshoot than the reliability programs.

If the foregoing procedure is followed and the diagnostic does not fail, return to the failing reliability program. When the failing loop has been reestablished, troubleshoot, following the same basic practices used with the other tests. As the programs become more complex, effective use of the various stop switches becomes increasingly essential for isolating malfunctions.

CAUTION

If a simulation routine should fail when running reliability programs, the result of the simulation will disagree with the result derived by the instruction under test, and an error will be indicated. As a result, a great deal of time can be wasted troubleshooting a perfectly good instruction when, in fact, the problem lies with the simulation. This situation should not arise if the test programs are run consecutively because the instructions used for simulation in one program are always proved operable in a preceding program.

6.3.7 Troubleshooting with Test Program N

Reliability program N performs its checks according to a number of variables, one of which is the magnitude of the operands provided by the random number generator. Consequently, the program does not run in a precise order of consecutive instructions as the other programs do. For this reason, and because the instructions under test are executed indirect through AC14, it is not always obvious from the program listing to determine how the program progressed to the point of error when it hangs up in a loop. Because of its complexities, and because it does not include a prognosis of failures, the program listing is not particularly recommended as a troubleshooting tool. But this situation is more than compensated for by the printout.

The printouts (Figure 6-2) accompanying Test N errors are far more definitive than those for the other programs. In particular, they list the proper contents of the registers at various event times in the instruction. Assuming, of course, that the simulation is correct, the printout occurs as a result of one of two situations:

a. The right answer is obtained but the wrong flags are set.

b. The wrong answer is obtained and flags may or may not be set incorrectly.

In the first situation, focus troubleshooting on the logic associated with the incorrect flags; in the second, eliminate the cause of the wrong answer and any unwanted flags will probably be eliminated. In either case, the practice of backtracking to the diagnostic that checks the failing instruction is a good initial

step. If the diagnostic fails, troubleshoot it as previously described; if not, return to reliability program N.

When an instance of setting the wrong flags is involved, the first line of the printout resembles the following:

FLAGS FROM MACH AND SIMULATE 440100 000000

The first group of numbers defines the flags that are set; the second group, the flags that should be set. Table 6-3 matches the flags to their bit positions in the output word.

Table 6-3
CP Flags Versus Output Word Bits

Bit	Flag
0	AROV
1	AR CRY0
2	AR CRY1
3	AR FOV
4	BYF6
5	EX USER
6	EX IOT USER
11	AR FXU
12	AR DCK

With the program in a failing loop, a malfunction might be isolated as follows:

a. Study the printout.

b. Using the various stop switches, compare the contents of the registers at specific times in the instruction cycle with the contents listed on the printout. As an example, Figure 6-2 indicates possible stop times and the switches that produce the stops.

c. Using step b., determine the two stops between which the error occurs.

d. Synchronize the oscilloscope with the decoded IR level of the failing instruction. For example, the printout in Figure 6-2 states that the failing instruction is an FDVL. In this case, the oscilloscope on the IR FDVL level would be synchronized at 1M16N.

e. Determine where the two stops, isolated in step c., fall on the oscilloscope sweep.

```
THE MACH RESULTS IN AC,AC+1,E    000000000000 740752457505 752620767173

*004663 IDIV    01 ,000003        000752457505 344104140305 752620767173
                                  000000000000 000752457505

TIME   PC     MA     SC  FE  AR           MQ           RR
I T0   004663 004663 000 000 230040000003 00000000000  752620767173
I T1   004663 000000 000 000 230040000003 00000000000  752620767173
A T3   004663 000000 000 000 000000000003 000000000000 230040000003
A T6   004663 000003 000 000 000000000003 000000000000 230040000003
F T0   004663 000003 000 000 752620767173 000000000000 230040000003
F T1A  004663 000003 000 000 752620767173 000000000000 752620767173
F T2   004663 000003 000 000 000000000000 000000000000 752620767173
F T2RQ 004663 000003 000 000 000752457505 000000000000 752620767173
F T3   004663 000003 000 000 000752457505 000000000000 752620767173
F T9   004664 000003 000 000 000752457505 000000000000 752620767173
E T0   004664 000003 735 000 000752457505 000000000000 752620767173
E T1   004664 000003 735 000 000000000000 000752457505 752620767173
E T2   004664 000003 735 000 000000000000 001725137212 752620767173
DST1   004664 000003 735 000 000000000000 001725137212 752620767173
DST2   004664 000003 735 000 000000000000 001725137212 752620767173
S CT0  004664 000003 735 000 000000000000 001725137212 752620767173
S CT2  004664 000003 736 000 000000000000 001725137212 752620767173
S CT3  004664 000003 736 000 725441756366 003365227642 4 752620767173
S CT2  004664 000003 737 000 725441756366 003365227642 4 752620767173
S CT3  004664 000003 737 000 725441756366 007524575050 752620767173
S CT2  004664 000003 740 000 725441756366 007524575050 752620767173
S CT3  004664 000003 740 000 725441756366 017251372120 752620767173
S CT2  004664 000003 741 000 725441756366 017251372120 752620767173
S CT3  004664 000003 741 000 725441756366 036522764240 752620767173
S CT2  004664 000003 742 000 725441756366 036522764240 752620767173
S CT3  004664 000003 742 000 725441756366 075245750500 752620767173
S CT2  004664 000003 743 000 725441756366 075245750500 752620767173
S CT3  004664 000003 743 000 725441756366 172513721200 752620767173
S CT2  004664 000003 744 000 725441756366 172513721200 752620767173
S CT3  004664 000003 744 000 725441756366 365227642400 752620767173
S CT2  004664 000003 745 000 725441756366 365227642400 752620767173
S CT3  004664 000003 745 000 725441756366 752457505000 752620767173
S CT2  004664 000003 746 000 725441756366 752457505000 752620767173
S CT3  004664 000003 746 000 725441756367 725137212000 752620767173
S CT2  004664 000003 747 000 725441756367 725137212000 752620767173
CT3    004664 000003 747 000 725441756371 652276424000 752620767173
S CT2  004664 000003 750 000 725441756371 652276424000 752620767173
S CT3  004664 000003 750 000 725441756375 524575050000 752620767173
S CT2  004664 000003 751 000 725441756375 524575050000 752620767173
S CT3  004664 000003 751 000 725441756405 251372120000 752620767173
S CT2  004664 000003 752 000 725441756405 251372120000 752620767173
S CT3  004664 000003 752 000 725441756424 522764240000 752620767173
S CT2  004664 000003 753 000 725441756424 522764240000 752620767173
S CT3  004664 000003 753 000 725441756463 245750500000 752620767173
S CT2  004664 000003 754 000 725441756463 245750500000 752620767173
S CT3  004664 000003 754 000 725441756560 513721200000 752620767173
S CT2  004664 000003 755 000 725441756560 513721200000 752620767173
S CT3  004664 000003 755 000 725441756753 227642400000 752620767173
S CT2  004664 000003 756 000 725441756753 227642400000 752620767173
S CT3  004664 000003 756 000 725441757340 457505000000 752620767173
S CT2  004664 000003 757 000 725441757340 457505000000 752620767173
S CT3  004664 000003 757 000 725441760313 137212000000 752620767173
S CT2  004664 000003 760 000 725441760313 137212000000 752620767173
S CT3  004664 000003 760 000 725441762240 276424000000 752620767173
S CT2  004664 000003 761 000 725441762240 276424000000 752620767173
S CT3  004664 000003 761 000 725441766112 575050000000 752620767173
S CT2  004664 000003 762 000 725441766112 575050000000 752620767173
S CT3  004664 000003 762 000 725441775637 372120000000 752620767173
S CT2  004664 000003 763 000 725441775637 372120000000 752620767173
S CT3  004664 000003 763 000 725442015110 764240000000 752620767173
S CT2  004664 000003 764 000 725442015110 764240000000 752620767173
S CT3  004664 000003 764 000 725442053633 750500000000 752620767173
S CT2  004664 000003 765 000 725442053633 750500000000 752620767173
S CT3  004664 000003 765 000 725442151101 721200000000 752620767173
S CT2  004664 000003 766 000 725442151101 721200000000 752620767173
S CT3  004664 000003 766 000 725442343615 642400000000 752620767173
S CT2  004664 000003 767 000 725442343615 642400000000 752620767173
S CT3  004664 000003 767 000 725442731045 505000000000 752620767173
S CT2  004664 000003 770 000 725442731045 505000000000 752620767173
S CT3  004664 000003 770 000 725443703525 212000000000 752620767173
S CT2  004664 000003 771 000 725443703525 212000000000 752620767173
S CT3  004664 000003 771 000 725445630664 424000000000 752620767173
S CT2  004664 000003 772 000 725445630664 424000000000 752620767173
S CT3  004664 000003 772 000 725451503163 050000000000 752620767173
S CT2  004664 000003 773 000 725451503163 050000000000 752620767173
S CT3  004664 000003 773 000 725461227760 120000000000 752620767173
S CT2  004664 000003 774 000 725461227760 120000000000 752620767173
S CT3  004664 000003 774 000 725500591352 240000000000 752620767173
S CT2  004664 000003 775 020 725500591352 240000000000 752620767173
S CT3  004664 000003 775 020 725537224336 500000000000 752620767173
S CT2  004664 000003 776 000 725537224336 500000000000 752620767173
S CT3  004664 000003 776 000 725634472307 200000000000 752620767173
S CT2  004664 000003 777 000 725634472307 200000000000 752620767173
S CT3  004664 000003 777 000 726027206230 400000000000 752620767173
S CT2  004664 000003 000 000 726027206230 400000000000 752620767173
S CT3  004664 000003 000 000 726414436073 000000000000 752620767173
S CT4  004664 000003 000 000 726414436073 000000000000 752620767173
D ST3  004664 000003 000 000 753573446700 000000000000 752620767173
D ST4  004664 000003 000 000 000752457505 000000000000 752620767173
D ST5  004664 000003 000 000 000000000000 000752457505 752620767173
D IVT5 004664 000003 000 000 000000000000 000752457505 752620767173
S T0   004664 000003 000 000 000000000000 000752457505 752620767173
S T1   004664 000003 000 000 000000000000 000752457505 752620767173
S T2   004664 000003 000 000 000000000000 000752457505 752620767173
S T7   004664 000003 000 000 000752457505 000752457505 752620767173
S T8   004664 000003 000 000 000752457505 000752457505 752620767173
S T9   004664 000003 000 000 000752457505 000752457505 752620767173
```

Figure 6-2   Test Program N, Error Printout

f. Find a timing pulse, between the two stops, that nearly coincides with the failure.

g. Using the timing pulse from step f. as a reference, examine the logic to determine where the bit was lost or altered, which enabling level was not generated, which transfer gate failed to respond to a strobe, and so forth.

6.4 MARGIN CHECK SYSTEM

The margin check system provides a means of substituting a Type 702 Variable Power Supply for the fixed +10 and -15V system operating voltages. The 5-position MARGIN CHECK switch on the maintenance panel selects the polarity of the margin voltage and makes it available to the left or right side of the system (see Figure 6-3). The processor and those cabinets to the left of it are considered the left side of the system; cabinets to the right side of the processor are considered the right side of the system.

Margins are run on a rack-by-rack basis as a function of the local margin switches. Each rack usually has two such switches; one selects the +10V margin voltage and the other selects the -15V margin voltage. There may be an extra switch for circuits which need to be margined separately. The MARGIN CHECK switch overrides the local switches in the sense that it will apply the proper fixed voltage to a rack whose local switches are inadvertently selecting a margin condition other than the one selected by the MARGIN CHECK switch.

The margin check system is used in conjunction with the MAINDEC test programs for both corrective and preventive maintenance. In performing corrective maintenance, the MAINDEC programs are used essentially as described in Paragraph 6.3, Processor Test Programs. While the programs are being run, however, the margin check system is used to aggravate intermittent or borderline failures into consistent failures that are easier to recognize and troubleshoot. The margin check meter is arranged to accurately read the voltage which is applied to the panel of logic, if margin check switches in only one area of the machine are on.

As part of preventive maintenance, MAINDEC test programs N and O should be run every 1000 hours, with the machine operating under margin conditions. Because of the various modules involved, margin specifications differ from rack to rack, as specified in Table 6-4. The thoroughness with which MAINDEC

programs N and O exercise the machine make it unnecessary to include the other MAINDEC programs in the preventive maintenance schedule.

6.4.1 Considerations for Running Margins

Running margins with the MAINDEC programs is an effective troubleshooting tool. However, certain precautions must be taken if it is to be used effectively. When a program fails under margin, it is not always obvious where it is failing; assuming that the point of failure is determined, it is usually much more difficult to determine the cause of failure than when the program is running without margins. The reason for this difficulty is that if margins are not being used, and if the test programs have been run in order, encountering an error usually indicates that the machine is not properly executing the instruction under test. This is not true when margins are involved. In this case, the error may, in fact, occur because the machine cannot execute the instruction under test. However, it is just as likely that the error occurs because the margins have:

a. Forced a failure in the simulation routines

b. Forced an illegal printout

c. Altered the instruction code

d. Altered the address of the operand

e. Forced a failure in the memory control logic.

An infinite number of situations can be added to the foregoing list. Consider as many of these situations as possible. Otherwise a great deal of time can be wasted troubleshooting a perfectly good instruction. Unfortunately, there is no foolproof method of avoiding this mistake, but the chance of making it can be minimized, being constantly aware that it can happen. Never forget margins are being run; when an error occurs, question the validity of the printout. If it appears legitimate, there is no recourse but to proceed with the normal methods for troubleshooting an error loop. If the printout does not appear valid, it is probable that an instruction, other than the instruction under test, is failing. Try to determine which instruction is causing the error from the printout and/or circuits being margined. Then, follow the standard procedure of returning to the most basic diagnostic test for that instruction. Try to make the diagnostic fail at approximately the same margins at which the original failure occurred. If it does fail, the malfunction probably can be easily detected. If it does not fail,

go back to the original program and try to determine where it is failing, using the various stop switches, program listing, flow charts, and all other means available.

6.4.2 Using the Margin Switches

Repositioning either the MARGIN CHECK or local margin switches while margins are being applied can produce transients that adversely affect machine operation. For example, transients can alter the contents of active registers. They can cause even more serious problems by starting a memory reference cycle, thereby altering the stored test program. To minimize such occurrences, proceed as follows when repositioning the margin switches.

a. Press the STOP switch.

b. Examine location 00. (This sets the MA to 00. If location 00 is altered inadvertently, no serious consequences result since that location is reinitiated when the program is restarted.)

c. Position the margin switches as desired.

d. Press the RESET switch.

e. Set the ADDRESS switches to 4000.

f. Press the START switch.

6.4.3 Altered Programs

Margining in areas of the memory control, instruction register, or program counter logic can alter the stored program. Error indications resulting from this situation

10-0421
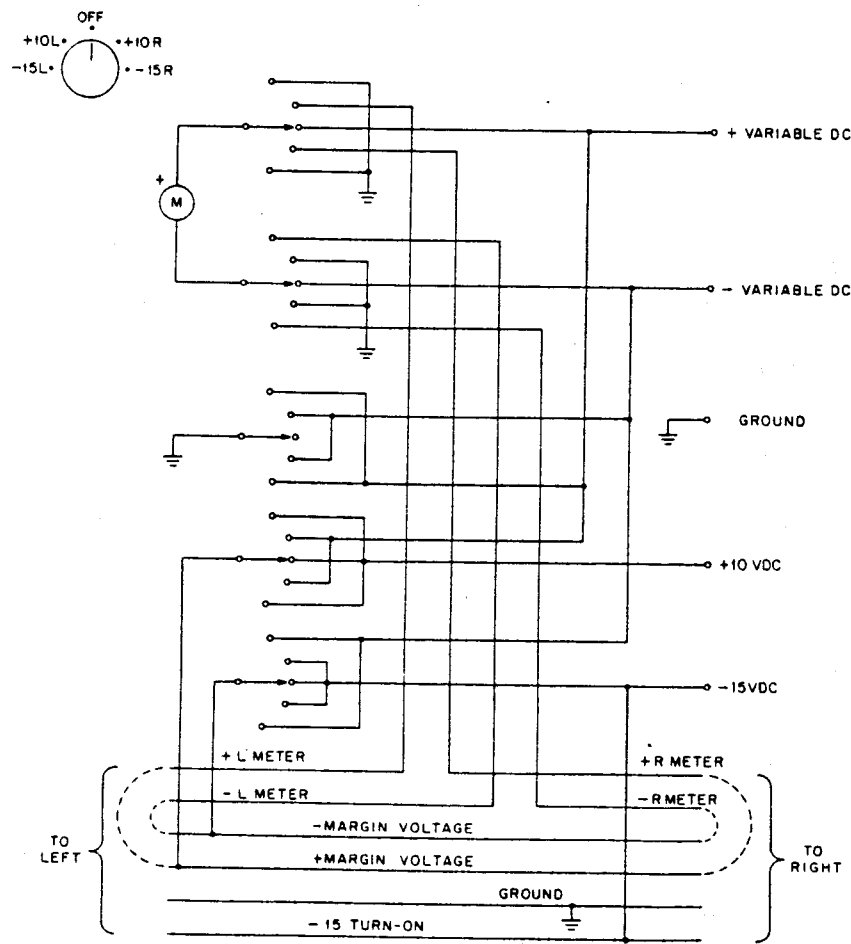
Figure 6-3 Margin Check System, Simplified Diagram

6-10

are invalid if the purpose of the test is to check out an instruction. When an error of this type is suspected, remove the margin voltage and restart the program at location 4000. If the program runs successfully with no margin voltage, the original error indication is probably valid. Reproduce the failure and trouble-shoot it. If the program does not run successfully with the margin voltage removed, it probably has been altered. In this case, ignore the error indications, reload the program, and resume the margining procedure at a point preceding the point of the original failure.

Table 6-4A
Margin Check Specifications
(for all tests)

| | +10V | | -15V | |
Panels	Max.	Min.	Max.	Min.
1A	+12.0V	+8.0V	-17.0V	-13.0V
1B	+17.5V	+3.0V	-18.0V	-12.0V
1C	+17.5V	+3.0V	-18.0V	-12.0V
1D	+17.5V	+3.0V	-18.0V	-12.0V
1E	+17.5V	+3.5V	-18.0V	-12.0V
1F	+17.5V	+3.5V	-18.0V	-12.0V
1H	+17.5V	+3.5V	-18.0V	-12.0V
1J	+17.5V	+3.5V	-18.0V	-12.0V
1K	+17.5V	+3.0V	-18.0V	-12.0V
1L	+17.5V	+3.0V	-18.0V	-12.0V
1M	+17.5V	+3.5V	-18.0V	-12.0V
1N	+17.5V	+3.0V	-18.0V	-12.0V
1P	+17.5V	+5.0V	-18.0V	-12.0V
1R	+17.5V	+3.0V	-18.0V	-12.0V
1S	+15.0V	+5.0 for Test D, L, and M*	-18.0V	-12.0V
	+17.5V	+3.0V for all other tests		
1T	+17.5V	+3.0V	-18.0V	-12.0V
2A	+12.0V	+8.0V	-17.0V	-13.0V
2B	+17.5V	+3.5V	-18.0V	-12.0V
2C	+17.5V	+3.0V	-18.0V	-12.0V
2D	+17.5V	+3.0V	-18.0V	-12.0V
2E	+17.5V	+3.0V	-18.0V	-12.0V
2F	+17.5V	+2.5V	-18.0V	-12.0V
2H	+17.5V	+3.0V	-18.0V	-12.0V
2J	+17.5V	+3.0V	-18.0V	-12.0V
2K	+17.5V	+2.5V	-18.0V	-12.0V
2L	+17.5V	+2.5V	-18.0V	-12.0V
2M	+12.0V	+8.0V	-17.0V	-13.0V
2N	+17.5V	+5.5V	-18.0V	-12.0V
2P	+17.5V	+3.5V	-18.0V	-12.0V
2R	+17.5V	+3.5V	-18.0V	-12.0V
2S	+17.5V	+3.0V	-18.0V	-12.0V
2T	+17.5V	+3.0V	-18.0V	-12.0V

*New Test Titles

Table 6-4B
Margin Check Specifications
(for all tests)

Programs Used for Checking Margins

Old Name	New Name	Passes or Duration
D	0D	2 min (ALL DATA SW0)
H	0I	2 min (Fast Mode)
I	0J	2 min (Fast Mode)
J	0K	2 min (Fast Mode)
K	0L	6 min (ALL DATA SW0)
L	0M	4 min (Fast Mode)
M	0N	4 min (Fast Mode)
BLT	0O	4 min (ALL DATA SW0)

Table 6-4C
Margin Check Specifications for Bay 3
(while running Tests D, TTY, and Reader Punch)

Panel	+10V		-15V	
	Max.	Min.	Max.	Min.
3A	+17.5V	+2.5V	-18.0V	-12.0V
3A(Reader Lamp)	+11.0V	+9.0V		
3B	+17.5V	+2.5V	-18.0V	-12.0V
3C	+17.5V	+2.5V	-18.0V	-12.0V
3D	+17.5V	+2.5V	-18.0V	-12.0V
3E	+17.5V	+5.5V	-18.0V	-12.0V
3F	+17.5V	+2.5V	-18.0V	-12.0V

NOTE: The Random Binary Test Mode of the Reader Punch Test should be run while taking margins.

6.5 TROUBLESHOOTING READ IN (RDI)

When an attempted read in operation does not initiate a data transfer or appears to transfer the wrong data, the malfunction could be in the processor and/or read in logic, or in the I/O device. To investigate the latter possibility, attempt a read in operation from another I/O device. If this attempt is successful, the first device and/or its control logic is probably malfunctioning. The failure can be isolated by using the appropriate I/O diagnostic program. If the read in from the second I/O device also fails, consider the character of the failure. If no data is being read in, the fault could be that the RDI logic is malfunction-

ing or that the processor has lost the ability to execute the DATAI and/or BLKI instructions required in an RDI operation. If the wrong data is being read in, the problem could lie with the RDI logic.

The following sections detail suggested techniques for isolating read in failures, assuming the I/O devices are operating properly.

6.5.1 No Data Read In

If pressing the READ IN key fails to initiate an input transfer, the first corrective step is to determine whether or not the processor can execute DATAIs and

BLKIs. This can be done by trying to read data from the paper tape reader, using the hardware read in (HRI) simulator program below. Proceed as follows: insert tape in the reader; deposit the HRI simulator program into memory from the console; press the START key.

HRI Simulator Program

Address	Left Half – Right Half		
60/	710600	000060	CONO PTR,60
61/	710740	000010	CONSO PTR,10
62/	254000	000061	JRST .-1
63/	710440	000000	DATAI PTR, 0
64/	710740	000010	CONSO PTR,10
65/	254000	000064	JRST .-1
66/	710400	000000	BLKI PTR,0
67/	256020	000000	XCT @ 0
70/	254000	000064	JRST .-4

If the HRI program fails to read the tape, the malfunction probably is not in the RDI logic but, rather, in the processor's inability to execute I/O instructions. This being the case, troubleshoot the program, using the SING INST and/or SING CYCLE keys. However, if the HRI program does read the tape, the malfunction is probably in the RDI logic. To isolate the failure, determine the point at which the RDI cycle is hanging up, as follows:

a. Select the paper tape reader (device code 104_8) with the READ IN DEVICE switches.

b. Remove any tape from the reader.

c. Set the MA switches to all 0s.

d. Deposit all 0s into location 00.

e. Press the SING CYCLE key.

f. Press the READ IN key.

If the processor were operating properly, 71044 would appear for an instant (an instant, in this case, is a period of time far too short to obtain a reaction from either the human eye or an incandescant filament; therefore, this transition is not seen) in the IR indicators; then, change to 71040; and the MI would contain all 1s. Since the processor is malfunctioning, however, one of the following will probably appear instead:

a. An instruction code other than 71044 or 71040 appears in the IR. This indicates that pressing the READ IN key does not generate the proper instruction(s) (DATAI, BLKI). Isolate the malfunction by checking the IR RDI SETUP.

b. 71044 shows in the IR, and the MI contains all 0s. This indicates that although the first DATAI is generated, it does not read in the pointer. Isolate the malfunction by checking the generation of pulses from IOB RDI DATA to IT1.

c. 71044 shows in the IR and the MI contains all 1s. This indicates that the first DATAI is not being changed to a BLKI after the pointer word is read in. Isolate the malfunction by checking for IR12 CLEAR at ST1 time.

6.5.2 Wrong Data Read In

If executing a READ IN appears to load data incorrectly into memory, it is possible that DATAI instructions in part 2 of the RDI cycle are not being changed to BLKI instructions. Again, a good initial step is to determine whether or not the processor properly executes BLKI instructions by running the HRI program with the paper tape reader as directed in the preceding chapter. If the HRI program reads the tape successfully, proceed as follows:

a. Select the paper tape reader (device code 104_8) with the READ IN DEVICE switches.

b. Press RESET

c. Remove any tape from the reader.

d. Set the MA switches to all 0s.

e. Deposit all 0s into location 00.

f. Cover all but the "8" hole (outermost hole) in the reader's photosensing mechanism.

g. With SING CYCLE and SING INST disable, depress READ IN.

Normally, the console indicators show that the contents of the MA and the pointer word in the MI are incremented continually as the tape is read. But if, as suspected, DATAOs in part 2 of the RDI cycle are not being changed to BLKIs, the contents of the MI and MA remain static and the IR contains 71044.

Such operation causes all data words to be loaded on top of one another in the initial memory location. The malfunction can probably be found by determining why IR RDI part 2 is not being set.

6.6 DELAY CHECKS AND ADJUSTMENTS

Use the following procedures to check adjustable delays. The delay times are screwdriver-adjustable.

6.6.1 SCT3 Delay

Use diagnostic test Z, Processor Timing Test (SPEEDY), to time the SCT3 delay. SPEEDY times this delay when it measures shifting time. The printout should include the following statement:

12-SHIFTING TAKES 150 NSEC

If necessary, adjust SCR3 delay as follows:

a. Set up a ROT 100 instruction on the console data switches (241000 000100).

b. Connect the oscilloscope to measure the delay between 1C21L and 1C21N.

c. Press XCT and REPT keys and adjust 1C21DE1 with a screwdriver for a 150-nsec delay.

6.6.2 SCT2 Delay

NOTE

Set the SCT3 delay before checking or adjusting the SCT2 delay.

a. Deposit the following program:

MA Instruction

10 IMUL 17   (220000 000017)
11 JRST 10   (254000 000010)
17 252525252525

b. Connect the oscilloscope at 1C16N (sync negative).

c. Start program execution at 10. The delay between SCT2 pulses should be 280 ±5 nsec. Screwdriver-adjust 1C16DE1 as required.

6.6.3 FDT1 Delay

a. Deposit the following program:

MA Instruction

10 MOVSI 201400   (205000 201400)
11 FDVRI 201400   (175000 201400)
12 JRST 10   (254000 000010)

b. Sync negative on channel A at 1E01L

c. Time output on channel B at 1E01N

d. Start program execution at 10. The delay between pulses should be 280 nsec. Screwdriver-adjust 1E01DE1 as required.

6.6.4 DST1 Delay

a. Deposit the following program:

MA Instruction

10 IDIVI 1   (231000 000001)
11 JRST 10   (254000 000010)

b. Start program execution at 10 and measure DST1 delay between 1F05L (sync negative) and 1F05N. The delay between pulses should be 280 nsec. Screwdriver-adjust 1F05DE1 as required.

6.6.5 MC RQ SET Delay

a. Deposit the following program:

MA Instruction

10 JRST 10   (254000 000010)

b. Start program execution at 10 and measure MC RQ SET delay from 1T15L to 1T15N. The delay should be 45 nsec. Screwdriver-adjust 1T15DE1 as required.

6.6.6 MR PWR CLR ENB

Two delays are to be checked: R303 at IP01, and R302 at IP02.

a. Check the R303 delay by applying a momentary ground to IP01T.

b. The MR PWR CLR ENB delay time should be about five sec. Screwdriver-adjust R303 as required.

c. To check the overall effect of the delay, turn off power and turn it on again. The delay time for the POWER ON indicator lamp to illuminate should be about five sec.

d. Check the R302 delay by applying a momentary ground to IP02E. The CPA PWR DLY measured at IP02M, should be about 5 ms. Screwdriver-adjust R302 as required.

6.6.7   MC NON EX MEM Delay

a. Set up a MOVE 777777 instruction on the console data switches (200000 777777).

b. Set the COARSE speed control to 6 and the FINE to 12 o'clock.

c. Connect the oscilloscope to measure the delay between a positive pulse at 1S02V and 1S02E going positive.

d. Press XCT and REPT keys. The delay should be 100 μsec. Adjust R303 at 1S02 as required.

6.6.8   IOT T2 and IOT RESTART Delays

a. Set up a DATAO P1, 0 instruction on the console data switches (700540 000000)

b. Connect the oscilloscope to 3F32E and sync positive.

c. Press XCT and REPT keys and measure the delay to 3F32M. The IOT T2 delay should be 1 μs.

d. Measure the delay from 3F32E to 3F32V. The IOT RESTART delay should be 2 μs. Adjust R302 at 3F32 as required.

6.6.9   IOT T4 and IOT RESET Delays

Maintain the same test conditions described in Paragraph 6.6.8.

a. Sync positive and scope 3F33N to measure IOT RESET DLY to 3F33V. The IOT RESET DLY should be 2.5 μs.

b. Sync positive and scope 3F33E to measure IOT T4 delay to 3F33M. The IOT T4 delay should be 1.5 μs.

c. Adjust R302 at 3F33 as required.

6.6.10   KEY REPT DLY

a. Set up an AOJA 0 instruction on the console data switches (344000 000000)

b. Press XCT and REPT keys and vary both COARSE and FINE speed controls to confirm that both control the speed of counting in location 0.

6.6.11   PTP SCR DRIVER

a. Punch some tape. There should be about a 5-sec delay from the time the feed button is released until the tape punch turns off.

b. Adjust R303 at 3A26 as required.

6.7   INDICATOR CHECKS

6.7.1   MEMORY DATA, MEMORY ADDRESS, and MEMORY INDICATOR Lamps

a. Turn off FM ENB.

b. Set COARSE speed control to 6 and FINE speed control to 3 o'clock.

c. Set all memory address and data switches to 0.

d. Press DEPOSIT THIS and REPT keys.

e. Set each data switch and memory address switch to a 1 and observe that every corresponding indicator lamp lights.

6.7.2   PC, IR, AC, I and INDEX Indicators

a. Leave all data and memory address switches set to 1.

b. Press the following keys in the order listed: STOP, RESET, SING CYCLE (leave on), START, STOP, RESET, DEPOSIT THIS, and XCT.

c. All of the following indicator lamps should light:

1. PROGRAM COUNTER indicators

2. INSTRUCTION indicators

3. AC indicators

4. I indicator

5. INDEX indicators

6. MEMORY DATA indicators

6.7.3 AR, BR, ADDER and PB Indicators

a. Press STOP and RESET keys.

b. Set up a MOVEI 0,0 instruction on the data switches (201000 000000).

c. Press XCT switch. The ARITHMETIC REGISTER, BUFFER REGISTER, ADDER, and PARITY BUFFER indicator lamps should all be off.

d. Set up an EXCH 0,0 instruction on the data switches (250000 000000)

e. Press XCT key. All ARITHMETIC REGISTER, BUFFER REGISTER, ADDER, and PARITY BUFFER indicators should light.

6.7.4 MQ, SC, and SCAD Indicators

a. Deposit all 0s in memory location 0 and deposit all 1s in memory location 1.

b. Switch SHIFT CNTR MAINT on.

c. Set up a ROTC 110 instruction on the data switches (245000 000110)

d. Press XCT key. All ARITHMETIC REGISTER indicators should be off. The MULTIPLIER QUOTIENT indicators should display 000----01. The SC REGISTER indication should be 670 and the SC ADDER indication should be 671.

e. Press CONT key 72 times and observe the indicator panel display. The least significant bit in the MEMORY QUOTIENT (MQ) should step one position to the left each time the CONT key is pressed, and pass into the ARITHMETIC REGISTER

(AR) display. When the bit enters the AR, the ADDER display should look like the AR. The bit displayed in the AR and ADDER should continue to step to the left.

f. Meanwhile, the SC REGISTER and SC ADDER indications should be counting down. When the SC REGISTER indicates 000, the SC ADDER should indicate 001 and the MQ and AR should equal 0----01.

6.8 SWITCH CHECKS

6.8.1 SHIFT CNTR MAINT Switch

The SHIFT CNTR MAINT function is tested in Paragraph 6.7.4.

6.8.2 FM ENB Switch

a. Examine memory locations 1 and 2 with the FM ENB switch on. No MEMORY/PROGRAM DATA indicators should change.

b. Set FM ENB switch off. The MEMORY/PROGRAM DATA indications should change.

6.8.3 REPT BYP Switch

a. Examine a nonexistent memory location such as 400000.

b. Press STOP and RESET keys. Then press NXM STOP and REPT keys.

c. Set COARSE speed control to 2 and FINE speed control to 2 o'clock.

d. Turn on REPT BYP and press EXAMINE NEXT key. The MA should count.

e. Set REPT BYP off. The MA should stop counting.

6.8.4 MI PROG DIS Switch

a. Turn MI PROG DIS switch on and press STOP and RESET keys.

b. Set up a DATAO P1 instruction on the data switches (700540 000000).

c. Press XCT key. The MEMORY DATA indicator should remain on.

d. Turn off MI PROG DIS switch and execute the same instruction. The MEMORY DATA indicator lamp should go out and the PROGRAM DATA indicator should light.

6.8.5 READ IN DEVICE Switches

a. Turn on all READ IN DEVICE switches.

b. Press the READ IN key and observe that corresponding indicators light in the IR display.

c. Turn off READ IN DEVICE switches, one by one. After each switch is turned off, press the READ IN key and observe that the corresponding indicator lamp goes out in the IR display.

6.8.6 Address Condition Switches

a. Deposit the following program:

MA	Instruction	
100	JRST 101	(254000 000101)
101	MOVE 100	(200000 000100)
102	MOVEM 100	(202000 000100)
103	JRST 100	(254000 000100)

b. Start program execution at 100.

c. Set memory address switches to 100.

d. Turn on INST FETCH and ADR STOP switches. The machine should stop with 100 in the PROGRAM COUNTER (PC).

e. Turn off ADR STOP and INST FETCH switches and press the CONT key. The program should continue.

f. Turn on DATA FETCH and ADR STOP switches. The machine should stop with 101 in the PC.

g. Turn off DATA FETCH and ADR STOP switches. Turn on WRITE, ADR STOP, and ADR BREAK switches.

h. Press CONT key. The machine should stop with 103 in the PC. The ADR BRK indicator (located in top row of Bay 1 indicator panel, 14th from left) should light.

APPENDIX A
FLOW DIAGRAM
AND
SCHEMATIC INTERPRETATION

The PDP-10 System, in particular the KA10 Central
Processor, can be generally described as an asynchro-
nous system. This has two meanings:

a. There is no central source of timing pulses.
(Pulses are generated by the logic as needed.)

b. There is no fixed sequence of timing pulses. The
system proceeds from task to task without allowing
time for more complicated cases which are not
required for the particular instruction.

The KA10 Central Processor incorporates pulse-
sampled level logic. Data residing in the various re-
gisters and control flip-flops of the processor are the
sources of the level logic. Generally, only a few
stages of level logic are necessary to reach a conclu-
sion. After a time adequate to cover the maximum
possible delay of the level logic stages, a pulse
is developed which samples the outputs of the level
logic. As a consequence, the registers and control
flip-flops may be changed, making up a new set of
conditions. The actual sequence of pulses which oc-
curs is also controlled by the level logic. The timing
is accomplished with a series of delay lines and pulse
amplifier-standardizers. These delay line-pulse am-
plifier combinations form a path with branches and
loops to accomplish repetitive tasks such as shifting.
The KA10 logic may itself be thought of as a program,
with branches and loops to accomplish its intended
functions.

A.1 SYMBOLOGY

Essential to understanding the flow chart and logic
diagrams, which illustrate the operations of the KA10,
is the ability of the reader to interpret the symbology
used in the preparation of these diagrams. For this
reason, a discussion of the symbol standards follows.
It is recommended that a reader unfamiliar with DEC
symbol standards read this section before studying the
theory of operation.

A.1.1 Lines

Lines are used to indicate flow. Where directional
arrows are not provided, the flow is presumed to be
down or to the right. Examples of flow lines are
shown below:
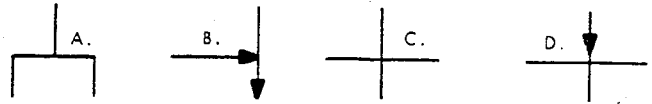
Example A.   Shows a path diverging.
Example B.   Shows two paths recombining.
Example C.   Shows two independent paths crossing;
Example D.   Shows a path branching three ways.

A.1.2 Pulses

Names enclosed in ovals as shown below represent
pulse amplifiers. A line leaving the bottom of an oval
generally indicates the continuation of the flow of
control. A line leaving the right side of the oval
generally leads to the indication of the other actions
caused by this pulse.
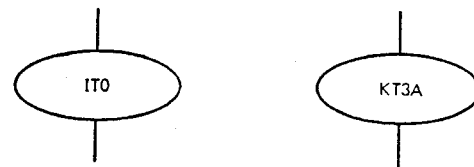
Pulses are named in such a manner that the location of
the flow diagram on which it appears is facilitated.
Only those pulse amplifiers which are part of the con-
trol flow appear as ovals and are named as described
below.

A pulse name is generally of the form XXTn. The XX
consists of from one to three characters which identify
the pulse-generating part of the logic. It usually cor-
responds directly to a drawing name, or to part of a
drawing name. The T indicates a time pulse. The
digit usually indicates which pulse, in a sequence of
pulses, it is. Due to the complexity of the logic,
however, pulses of a given group do not always occur
in numerical order. Sometimes an additional letter
appears following the digit. This usually indicates an
instance where additional drive was required, or some
other reason which required an extra pulse amplifier
to be inserted. These additional letters are, however,
logically separate from the pulse amplifier of the same
name without the suffix.

Time pulses are thought of not only as pulses which perform some action, but also as "times" which occur during some cycle of operation or hardware subroutine. Hence, in speaking, the names are often expanded; e.g., FT9 to "fetch time 9", DIVT3 to "divide time 3", IOT T2 to "iot time 2". (Iot is a short form of "input/output transfer" from earlier, and smaller, DIGITAL machines.)

A.1.3 Delays

Horizontal lines in the flow diagram, separated by a time specification, indicate delays. Times with unspecified units are in microseconds. Time delays under 250 ns are usually implemented by delay lines; longer time delays by monostable multivibrators. The time given is measured through the delay only and does not count delays through associated gating and pulse amplifier circuits. Hence, the time delay of a given operation cannot be determined by simply adding the delays specified.

.210

Delay is 0.210 µs
(or 210 ns)

5 ms

Delay is 5 milliseconds
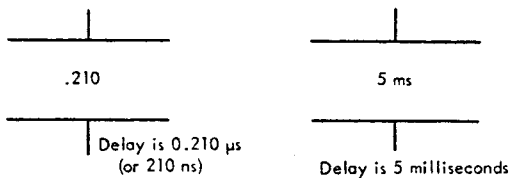
Horizontal lines in the flow (usually doubled) separated by names, as shown below, indicate that the flow has temporarily left this flow diagram to make use of a hardware subroutine. The upper name is the name of the subroutine, the lower name is the last pulse in the subroutine. Hardware subroutines are used where substantially similar events are desired from several places in the overall logic. Examples of these are: read and write from memory, shift a specified number of times, multiply two numbers. These subroutines are sometimes nested (e.g., "floating multiply" calls "multiply" which in turn calls "shift").

WRRQ
MCRST1

to SCT0
SCT4

A.1.4 Conditions

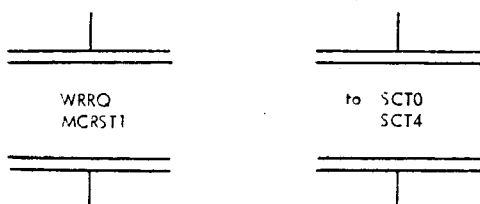Flow lines interrupted by some logical expressions, as shown below, indicate that flow only proceeds down paths where the given condition is true at the time the pulse appears at that point.

ADO (0)   ADO (1)   - (E LONG or ST INH)   E LONG
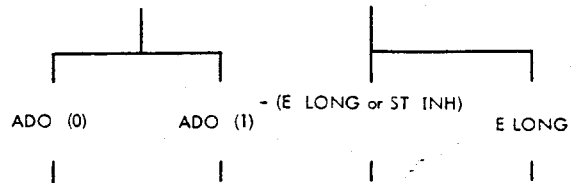
A.1.5 Operations

Boxes connected to ovals containing specifications of operations are best explained by considering the example shown below. The MQ FM AD(J) means that the MQ will have the contents of the adder jammed into it. The FM stands for from; this example is read as: "MQ from Adder, jammed". Other similar possibilities include MQ FM AD(1), meaning that only the 1s are transferred, causing an Inclusive OR; or MQ FM AD(0), transferring the 0s, causing an AND.

DST5

MQ FM AD(J);   CORRECT REMAINDER
AR FM MQ(J);   QUOTIENT
IR1(1): AD AR +EN CLR;
         AD CRY 36 SET;   NEGATE REMAINDER
         AD AR -EN SET;
IR1(0): AD CRY 36 SET;   ROUNDING
         AD AR +EN SET
         AD AR -EN CLR
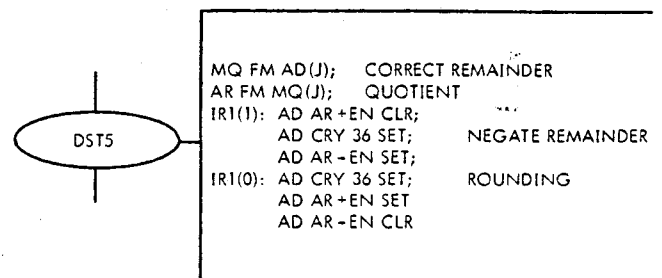
This backward notation is used in order to lead the reader directly to the block schematic diagram which has the logic in question. In this example, the gating and pulse amplifiers which load the MQ are found on the MQ control print. One would expect to find DST5 as an input on that PA. The semicolon delimits comments. The "correct remainder" is supplied as additional explanatory information. The AR FM MQ (J) is similar to that described above. Note that, in combination with the above item, the MQ is both changed and read out. Since these two items are in the same

box, they occur simultaneously, and the order in which they appear in the box is irrelevant. In all such cases, the old contents are what are read out.

The following three items, all indented past the IR1(1), are all contingent on IR bit 1 being in the 1 state. Hence, those actions only occur if IR1 is a 1. Specifications of the form XXXX SET or CLR refer to individual control flip-flops. As usual, the first few characters of the name indicate the logic diagram where the flip-flop can be found. A flip-flop which has been SET will be in the 1 state upon application of a time pulse; one which has been CLRed will be in the 0 state.

A.1.6 Tabular Format

The explanation above covers one of the two forms of flow charts used for the KA10. This form is used where it is most important to convey the actual flow through the logic. However, in the basic in-structions, most of the action occurs at a few standard time pulses. Hence, a different form of flow chart is used to detail which actions take place for various instructions, either individually or in groups, at the standard execution time pulses. This other form is a rather standardized tabular format. The example below, shown as Figure A-1, is taken from the basic instruction flow.

The use of X in the instruction conditions allows for variations of the basic instructions to be covered. Here XXX-: FCE means that the instructions ADD and SUB (without suffixes) cause the FCE function. The ADDM, ADDB, SUBM, SUBB cause FCE PSE, by similar reasoning. At FT9, any ADD instruction (ADD, ADDI, ADDM, ADDB) causes AD BR+EN SET.

A.2 LOGIC SYMBOLOGY

The system of logic symbology used in the PDP-10 drawings has as its primary objective the clarification of what logical function is being performed by a given

INSTRUCTION	ADD SUBTRACT	Which instructions are covered by this column.
INITIAL SWITCHES	EF0 LONG XXX-: FCE XXX_M : FCE XXX_B : FCE PSE	The settings of a number of gating conditions used in the fetch and execution cycles.
INITIAL REGISTERS	AR; C(AC) BR; (0,E) or (ACE) MA: E	The initial contents of the registers used in this instruction by FT9.
FT9	AD AR+EN SET ADDX: AD BR+EN SET SUBX: AD BR-EN SET AD CRY 36 SET	The events which take place at FT9 for instructions in this group. Events at FT9 cannot be data-dependent.
ET0	AR FM AD(J) ARF CRY STB	ET0 is the first execution time for data-dependent operations.
ET1		ET1 is an optional pulse.
ET2		ET2 is also optional.
FINAL SWITCHES	XXXM: SAC INH	Settings of conditions used in the store cycle.

Figure A-1 Tabular Format

circuit. Hence, a system of non-polar logic has been adopted. Non-polar implies that any given signal has two names: a mnemonic name to indicate one polarity, and a negation of that name at the other polarity. For example, the instruction decoders provide outputs that are true for the ground (or high) level; this is symbolized by an open diamond ——◇ at the end of signal line. One decoder output is named IR ILDB ——◇. Note that the assertion polarity symbol must be included with the name because the same line is also called -IR ILDB ——◆ (at -3V, the low level). Thus, this line carries potentially two useful signals: any gate that requires an input when the instruction register holds the instruction ILDB has the input available at the high level, and any gate that must know when ILDB is not present has the input available at the low level. Because some logic functions are required when either polarity is available, many of the signals in the processor are provided with both polarities by means of inverters. The inverters usually appear at the source end of the signal as named by its source. For example, an instruction decoder similar to the example above provides IR ASHC ——◇ and, by the same arguments, -IR ASHC ——◆. However, the machine requires one or both of the signals IR ASHC ——◆ or -IR ASHC ——◇; therefore, an inverter appears at the output of this decoder as shown in Figure A-2. For this case, all combinations of the signal IR ASHC are available.

10-0444
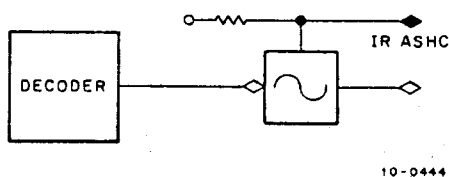
Figure A-2   Inverter

A similar situation exists for flip-flops and devices having register-like properties such as adders. Here, as an aid in identifying which signals are from flip-flops and which are produced by gates, the suffixes (1) and (0) appear at all flip-flop-originated signals. These suffixes are read aloud as: on a 1 and on a 0, respectively. Because flip-flops have two complementary output terminals, a situation exists that is analogous to the gate with an inverter. Each of the two outputs of the flip-flop has two logical names; for example, the signal IR 15(1) ——◆ is the same as the signal -IR 15 (0) ——◇; however, because a suffix is used on flip-flops, it can be seen that the signals -X(1) and X(0) are equivalent. Hence, the signal -IR 15(1) ——◇ always appears as IR 15(0) ——◇.

Other modules are indicated by an abbreviation enclosed in a rectangular box. These include the following:

BD   Bus driver used for driving cables

+   Adder used to perform arithmetic sum operations

ADR   Alternate designation for adder

ITD   Initial transient detector used in console logic

CLK   Clock

SS   Schmitt trigger

In some instances, it may be necessary to refer to a module catalog or the PDP-10 System Module Reference Manual to obtain the characteristics of a module.

A similar argument holds for the other terminal of a flip-flop. This is symbolized on drawings by showing four outputs from a flip-flop; these are grouped for (0) and (1), as shown in Figure A-3. The P and N letters outside the symbol are output pins of the flip-flop. Pin P can be both (0) ——◆ and (1) ——◇. Pin N can be both (0) ——◇ and (1) ——◆. Therefore, when a gate calls for one of the four outputs from the flip-flop, it becomes a simple matter to identify the signal source.

10-0445
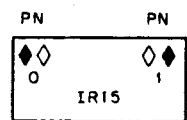
Figure A-3   Flip-Flop

Gates are shown as rectangular boxes with a symbol enclosed to tell what logical function is performed by the gate, given the polarity of the inputs shown. The dual nature of logic symbology shows itself again because a given physical gate can be both a NAND and a NOR type, depending on the polarity of its inputs as shown in Figure A-4. A NAND gate is symbolized by ~∧, and a NOR gate by ~V. The NOT part of

A-4

the gate inverts the polarity of the output; and there-
fore, the gates are used as AND and OR gates for log-
ical purposes. In the drawings, the representation of
a given gate is chosen, which makes the logical func-
tion clearest.

true at ground). Most ———▶ pulses are produced by
pulse amplifiers as shown in Figure A-5. Pulse ampli-
fiers (usually referred to as PAs) normally are trig-
gered from a ground-going input pulse, but because
PAs reshape the output pulse, they can operate from
any ground-going level change.

Figure A-5   Pulse Amplifier

Figure A-4   NAND and NOR Gates

Some logical functions are performed without the
necessity for gates by paralleling the outputs of exist-
ing gates. Because almost all the gates in the KA10
are of the DTL type (diode-transistor-logic), meaning
that they have diode inputs and a single ended tran-
sistor output, they can be considered as controlled
switches, which are either open circuits or shorts to
ground. By connecting two or more such gates in
parallel, the output is grounded when either gate is
on. Hence, paralleled gates give an OR at ground.
By the argument of duality, an AND function is pro-
duced by such a gate at -3V.

Gates can handle both pulses and levels. Pulses are
indicated by the symbol ———▶ (denoting a pulse which
is true at -3V) and ———▷ (denoting a pulse which is

Delay lines are shown as elongated rectangles, with a
delay time specification enclosed. Most delay lines
in the KA10 have discrete taps in 25-ns increments
(e.g., B311), while some are continuously variable
(e.g., B312).

Delay lines are normally driven by a PA output. Their
output pulse configuration, for a given input pulse, is
shown in Figure A-6. The output of the delay line is
of a different shape, wider, and of less power than the
PA pulse.
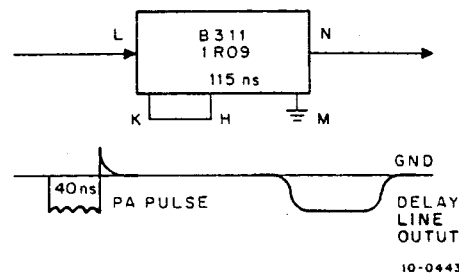
Figure A-6   Delay Line and Delay Line Output

APPENDIX B
INSTRUCTION CODE

	--0	--1	--2	--3	--4	--5	--6	--7
00—	(ILLEGAL)							
01—								
02—			USER DEFINED UUOs					
03—			(UNIMPLEMENTED USER OPERATIONS)					
04—	CALL	INIT		LEFT FOR SPECIAL MONITORS				CALLI
05—	OPEN	TTCALL	RESERVED FOR DEC			RENAME	IN	OUT
06—	SETSTS	STATO	GETSTS	STATZ	INBUF	OUTBUF	INPUT	OUTPUT
07—	CLOSE	RELEAS	MTAPE	UGETF	USETI	USETO	LOOKUP	ENTER
10—	UJEN							
11—								
12—								
13—	UFA	DFN	FSC	IBP	ILDB	LDB	IDPB	DPB
14—	FAD	-L	-M	-B	FADR	-I	-M	-B
15—	FSB	-L	-M	-B	FSBR	-I	-M	-B
16—	FMP	-L	-M	-B	FMPR	-I	-M	-B
17—	FDV	-L	-M	-B	FDVR	-I	-M	-B
20—	MOVE	-I	-M	-S	MOVS	-I	-M	-S
21—	MOVN	-I	-M	-S	MOVM	-I	-M	-S
22—	IMUL	-I	-M	-B	MUL	-I	-M	-B
23—	IDIV	-I	-M	-B	DIV	-I	-M	-B
24—	ASH	ROT	LSH	JFFO	ASHC	ROTC	LSHC	
25—	EXCH	BLT	AOBJP	AOBJN	JRST	JFCL	XCT	
26—	PUSHJ	PUSH	POP	POPJ	JSR	JSP	JSA	JRA
27—	ADD	-I	-M	-B	SUB	-I	-M	-B
30—	CAI	-L	-E	-LE	-A	-GE	-N	-G
31—	CAM	-L	-E	-LE	-A	-GE	-N	-G
32—	JUMP	-L	-E	-LE	-A	-GE	-N	-G
33—	SKIP	-L	-E	-LE	-A	-GE	-N	-G
34—	AOJ	-L	-E	-LE	-A	-GE	-N	-G
35—	AOS	-L	-E	-LE	-A	-GE	-N	-G
36—	SOJ	-L	-E	-LE	-A	-GE	-N	-G
37—	SOS	-L	-E	-LE	-A	-GE	-N	-G
40—	SETZ	-I	-M	-B	AND	-I	-M	-B
41—	ANDCA	-I	-M	-B	SETM	-I	-M	-B
42—	ANDCM	-I	-M	-B	SETA	-I	-M	-B
43—	XOR	-I	-M	-B	IOR	-I	-M	-B
44—	ANDCB	-I	-M	-B	EQV	-I	-M	-B
45—	SETCA	-I	-M	-B	ORCA	-I	-M	-B
46—	SETCM	-I	-M	-B	ORCM	-I	-M	-B
47—	ORCB	-I	-M	-B	SETO	-I	-M	-B
50—	HLL	-I	-M	-S	HRL	-I	-M	-S
51—	HLLZ	-I	-M	-S	HRLZ	-I	-M	-S
52—	HLLO	-I	-M	-S	HRLO	-I	-M	-S
53—	HLLE	-I	-M	-S	HRLE	-I	-M	-S
54—	HRR	-I	-M	-S	HLR	-I	-M	-S
55—	HRRZ	-I	-M	-S	HLRZ	-I	-M	-S
56—	HRRO	-I	-M	-S	HLRO	-I	-M	-S
57—	HRRE	-I	-M	-S	HLRE	-I	-M	-S
60—	TRN	TLN	TRNE	TLNE	TRNA	TLNA	TRNN	TLNN
61—	TDN	TSN	TDNE	TSNE	TDNA	TSNA	TDNN	TSNN
62—	TRZ	TLZ	TRZE	TLZE	TRZA	TLZA	TRZN	TLZN
63—	TDZ	TSZ	TDZE	TSZE	TDZA	TSZA	TDZN	TSZN
64—	TRC	TLC	TRCE	TLCE	TRCA	TLCA	TRCN	TLCN
65—	TDC	TSC	TDCE	TSCE	TDCA	TSCA	TDCN	TSCN
66—	TRO	TLO	TROE	TLOE	TROA	TLOA	TRON	TLON
67—	TDO	TSO	TDOE	TSOE	TDOA	TSOA	TDON	TSON

7—   INPUT - OUTPUT INSTRUCTIONS

7— —00—BLKI   the device number is inserted in bits 3 to 9
7— —04—DATAI   of each I/O instruction.
7— —10—BLKO
7— —14—DATAO
7— —20—CONO
7— —24—CONI
7— —30—CONSZ
7— —34—CONSO

APPENDIX C
INSTRUCTION WORD FORMATS

BASIC INSTRUCTIONS

| INSTRUCTION CODE (Operation Mode) | | I | X | Y |

IN OUT INSTRUCTIONS

| DEVICE CODE | INSTRUCTION CODE | I | X | Y |

PC WORD

| FLAGS | | 0 | 0 | 0 | 0 | 0 | PC |

| OVERFLOW | CARRY 0 | CARRY 1 | FLOATING OVERFLOW | BYTE INTERRUPT | USER | USER IN OUT | | FLOATING UNDERFLOW | NO DIVIDE |

BLT POINTER XWD

| SOURCE ADDRESS | DESTINATION ADDRESS |

BLKI/BLKO POINTER, PUSHDOWN POINTER, DATA CHANNEL CONTROL WORD IOWD

| WORD COUNT | ADDRESS 1 |

BYTE POINTER

| POSITION P | SIZE S | I | X | Y |

BYTE STORAGE

| BYTE | NEXT BYTE |

FIXED POINT OPERANDS

| BINARY NUMBER (2's COMPLEMENT) |

FLOATING POINT OPERANDS

| SIGN | EXCESS 128 EXPONENT (One's Complement) | FRACTION (2's COMPLEMENT) |

LOW ORDER WORD IN DOUBLE LENGTH FLOATING POINT OPERANDS

| SIGN | EXCESS 128 EXPONENT 27 IN POSITIVE FORM | LOW ORDER HALF OF FRACTION (2's COMPLEMENT) |

WORD FORMATS

10-0446

C-1

	00	04	10	14	20	24	30	34	40	44	50	54	60	64	70	74
0	APR CPA CENTRAL PROCESSOR	PI PRIORITY INTERRUPT	DRUM PROCESSOR	CCI PDP 8,9 INTERFACE	CCI2 PDP 8,9 INTERFACE	ADC ANALOG DIGITAL CONVERTER	ADC2 ANALOG DIGITAL CONVERTER								CLK REAL TIME CLOCK	CLK2 REAL TIME CLOCK
1	PTP PAPER TAPE PUNCH	PTR PAPER TAPE PUNCH	CDP CARD PUNCH	CDR CARD READER	TTY CONSOLE TELETYPE	LPT LINE PRINTER	DIS DISPLAY	DIS2 DISPLAY	PLT PLOTTER	PLT2 PLOTTER	CR CARD READER	CR2 CARD READER	PDP 7,8 INTERFACE		DSK SMALL DISK	DSK2 SMALL DISK
2	DC DATA CONTROL	DC2 DATA CONTROL	UTC DECTAPE	UTS DECTAPE	MTC	MTS MAGNETIC TAPE	MTM	LPT2 LINE PRINTER	DLS DATA LINE SCANNER	DLS2 DATA LINE SCANNER	DPC DISK PACK SYSTEM	DPC2 DISK PACK SYSTEM	MDF MASS DISK FILE	MDF2 MASS DISK FILE	DF DISK FILE	SMALL DISK
3	DCSA DATA COMMUNICATION	DCSB			DTC DECTAPE	DTS DECTAPE	DTC2 DECTAPE	DTS2 DECTAPE	TMC MAGNETIC TAPE	TMS MAGNETIC TAPE	TMC2 MAGNETIC TAPE	TMS2 MAGNETIC TAPE				
4																
5																
6																
7																

DEC STANDARD DEVICES

USER SPECIAL DEVICES

DEVICE MNEMONICS

DEVICE CODE

IN OUT INSTRUCTION WORD		FIRST OCTAL DIGIT			SECOND OCTAL DIGIT			THIRD OCTAL DIGIT (MSB bit)			INSTRUCTION CODE		
0	1	2	3	4	5	6	7	8	9	10	11	12

Used with PDP 6
Used with PDP 10
Device whose code is 124

646 LP10 LPT LINE PRINTER

Octium number for PDP 6
Octium number for PDP 10
The number indicates device is part of central processor
Mnemonic for device code 124

IO-0447

Digital Equipment Corporation
Maynard, Massachusetts

digital