

# decsystem10

## SYSTEM REFERENCE MANUAL

digital





**dec**system10

**SYSTEM REFERENCE MANUAL**

DEC-10-XSRMA-A-D

**digital equipment corporation • maynard, massachusetts**

1st Edition, May 1968  
2nd Edition, December 1971  
3rd Edition, August 1974

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The software described in this document is furnished to the purchaser under a license for use on a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Copyright © 1971, 1972, 1973, 1974, 1975 by Digital Equipment CORPORATION

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's evaluation to assist us in preparing future documentation.

Changes are indicated by a triangle (△) in the outside margin.

The following are standard trademarks of Digital Equipment Corporation:

CDP	DIGITAL	KA10	QUICKPOINT
COMPUTER LAB	DNC	K110	RAD-8
COMSYST	EDGRIN	LAB-8	RSTS
COMTEX	EDUSYSTEM	LAB-8/e	RSX
DDT	FLIP CHIP	LAB-K	RTM
DEC	FOCAL	OMNIBUS	RT-11
DECCOMM	GLC-8	OS/8	SABR
DECSYSTEM-10	IDAC	PDP	TYPESET-8
DECTAPE	IDACS	PHA	TYPESET-10
DIBOL	INDAC	PS/8	UNIBUS

# Preface

This manual explains the machine language programming and operation of the DECsystem-10, for both instructional and reference purposes. Basically the manual defines in detail how the central processor and the peripherals function, exactly what their instructions do, how they handle data, what their control and status information means, and what programming techniques and procedures must be employed to utilize them effectively. The programming is given in machine language, in that it uses only the basic instruction and device mnemonics and symbolic addressing defined by the assembler. The treatment relies on neither any other Digital software nor any of the more sophisticated features of the assembler; moreover the manual is completely self-contained – no prior knowledge of the assembler is required.

The text of the manual is devoted almost entirely to functional description and programming. Chapter 1 discusses the general characteristics of the system, defines the formats of the words used for numbers and instructions, and also explains the conventions needed to program the system and understand the examples given in the text. Chapter 2 covers all phases of the central processor, including the general principles of in-out programming and handling the interrupt system. The remaining chapters are devoted to the various categories of peripheral equipment. Chapters 3 and 4 cover the simple character-oriented devices that use form paper, paper tape and cards. Chapter 5 treats the data interfaces that are employed in the tape, disk and data communication systems covered in the three chapters following. Finally Chapter 9 describes the various terminals that can be used either at the console or in communication systems; this chapter includes both programming and operating information.

The first three appendices contain the basic reference tables for the programmer – word formats, instruction and device mnemonics, IO codes, IO bit assignments showing conditions and status, and a shorthand presentation of instruction actions in symbolic form. The next two appendices provide additional programming information of less general use: Appendix D gives the instruction times and Appendix E documents the differences among the several central processor models. The final three appendices provide a complete guide to the operation of the central processors, memories and peripheral devices (except terminals). This treatment is entirely in hardware terms, describing all lights and switches, how to load the devices, and so forth, but not how to run the system in terms of interacting with any Digital software – that information is given in the DECsystem-10 Operator's Guide.



# Contents

1.	INTRODUCTION	1-1
1.1	Number System	1-7
	Floating point arithmetic	1-8
1.2	Instruction Format	1-10
	Effective address calculation	1-11
1.3	Memory	1-12
	KI10 memory allocation	1-14
	KA10 memory allocation	1-14
1.4	Programming Conventions	1-15
2.	CENTRAL PROCESSOR	2-1
2.1	Half Word Data Transmission	2-2
2.2	Full Word Data Transmission	2-9
	Move instructions	2-10
	Pushdown list	2-12
2.3	Byte Manipulation	2-15
2.4	Logic	2-17
	Shift and rotate	2-24
2.5	Fixed Point Arithmetic	2-26
	Arithmetic shifting	2-30
2.6	Floating Point Arithmetic	2-31
	Scaling	2-33
	Number conversion	2-34
	Single precision with rounding	2-36
	Single precision without rounding	2-38
	Double precision operations	2-42
2.7	Arithmetic Testing	2-45
2.8	Logical Testing and Modification	2-51
2.9	Program Control	2-58
	Overflow trapping	2-69
2.10	Unimplemented Operations	2-70
2.11	Programming Examples	2-72
	Processor identification	2-72

	Parity	2-72	
	Counting ones	2-75	
	Number conversion	2-77	
	Table searching	2-78	
	Double precision floating point	2-79	
2.12	Input-Output		2-81
	Readin mode	2-85	
	Console-program communication	2-86	
2.13	Priority Interrupt		2-87
	KI10 interrupt	2-88	
	KA10 interrupt	2-94	
2.14	Processor Conditions		2-98
	KI10 processor conditions	2-98	
	KA10 processor conditions	2-101	
2.15	KI10 Program and Memory Management		2-104
	Paging	2-105	
	Page failure	2-104	
	Monitor programming	2-111	
	Executive XCT	2-114	
2.16	KA10 Program and Memory Management		2-117
	User programming	2-119	
	Monitor programming	2-119	
2.17	Real Time Clock DK10		2-120
3.	CONSOLE IN-OUT EQUIPMENT		3-1
3.1	Paper Tape Reader		3-1
	Readin mode	3-3	
3.2	Paper Tape Punch		3-4
3.3	Console Terminal		3-6
4.	HARDCOPY EQUIPMENT		4-1
4.1	Line Printer LP10		4-1
4.2	Plotter XY10		4-8
4.3	Card Reader CR10		4-11
4.4	Card Punch CP10		4-15
5.	DATA INTERFACES		5-1
5.1	Data Channel DF10		5-1
5.2	Twelve- and Eighteen-Bit Computer Interface DA10		5-7
	PDP-10 instructions	5-7	
	Twelve-bit computer instructions	5-8	
	Eighteen-bit computer instructions	5-10	
	Programming considerations	5-11	

6	MAGNETIC TAPE	6-1
Part I	DECTape	6-1
6.1	Tape Format	6-2
	Standard format DECTape	6-3
	Compatibility	6-3
6.2	Tape Handling Characteristics	6-4
6.3	Instructions	6-5
6.4	Normal Programming	6-11
	Timing	6-12
	Readin mode	6-14
6.5	Formatting a Tape	6-14
Part II	Standard Magnetic Tape	6-16
6.6	Tape Format	6-16
6.7	Instructions	6-19
6.8	Tape Functions	6-27
	Interrupt when unit ready	6-27
	Write	6-27
	Mark end of file	6-28
	Erase	6-28
	Erase and write	6-28
	Read record	6-28
	Read multirecord	6-29
	Read-compare record	6-29
	Read-compare multirecord	6-30
	Space records forward	6-30
	Space file forward	6-30
	Space records reverse	6-30
	Space file reverse	6-31
	Rewind	6-31
	Rewind and unload	6-31
6.9	Programming Considerations	6-31
	Readin mode	6-32
6.10	Timing	6-33
	Tape transport TU10	6-33
	Tape transport TU20	6-34
	Tape transport TU30	6-35
	Tape transport TU40	6-35

7	DISKS AND DRUMS	7-1
Part I	RC10 Disk/Drum System	7-2
7.1	Data Format	7-3
7.2	Instructions	7-4
7.3	Programming Considerations Timing 7-11	7-10
7.4	Operation	7-14
Part II	RP10 Disk Pack System	7-18
7.5	Data Format	7-18
7.6	Instructions	7-20
7.7	Disk Pack Functions	7-27
7.8	Programming Considerations Timing 7-29	7-28
7.9	Operation	7-30
8	DATA COMMUNICATIONS	8-1
8.1	Communication Signals and Procedures Bell System data sets 8-6	8-3
8.2	Data Communication System DC68A Data multiplexing 8-11 Modem control DC08F 8-17 Call control DC08H 8-19 689AG: Part I, modem control 8-21 689AG: Part II, call control 8-24	8-7
8.3	Data Line Scanner DC10 Instructions 8-29 Data line programming 8-33 Modem control programming 8-35	8-26
8.4	Single Synchronous Line Unit DS10 Instructions 8-37 Programming considerations 8-40	8-36

## APPENDICES

A	INSTRUCTIONS AND MNEMONICS	A-1
	Word Formats A-2	
	Mnemonic Derivation A-4	
	Numeric Listing A-5	
	Alphabetic Listing A-8	
	Device Mnemonics A-12	
	Algebraic Representation A-13	



B	IN-OUT CODES	B-1
	ASCII Code	B-2
	Line Printer Codes	B-4
	Card Codes	B-8
C	IO BIT ASSIGNMENTS	C-1
	KI10 Processor	C-2
	KA10 Processor	C-6
	Console IO	C-8
	<i>Peripheral devices follow in alphabetical order</i>	
D	TIMING	D-1
	KI10 Instruction Times	D-3
	KA10 Instruction Times	D-9
E	PROCESSOR COMPATIBILITY	E-1
F	PROCESSOR OPERATION	F-1
F1	KI10 Operation	F1-1
	Indicators	F1-2
	Operating keys	F1-6
	Operating switches	F1-8
	Real time clock DK10	F1-13
F2	KA10 Operation	F2-1
	Indicators	F2-1
	Operating keys	F2-3
	Operating switches	F2-7
	Real time clock DK10	F2-9
G	MEMORY OPERATION	G-1
	Address Structure	G-3
	MA10 Core Memory	G-4
	MB10 Core Memory	G-5
	MD10 Core Memory	G-6
	ME10 Core Memory	G-8
	MF10 Core Memory	G-9
H	OPERATION OF PERIPHERAL EQUIPMENT	H-1
H1	Console Equipment	H1-1
	Paper tape reader	H1-1
	Paper tape punch	H1-1
	Console terminal	H1-2
H2	Hardcopy Equipment	H2-1
H2.1	Line Printer LP10	H2-1
	Models LP10F, H	H2-1

	Models LP10B, C, D, E	H2-4
	Model LP10A	H2-6
H2.2	Plotter XY10	H2-6
H2.3	Card Reader CR10	H2-7
	Models CR10D, E, F	H2-7
	Model CR10A/B	H2-9
H2.4	Card Punch CP10	H2-9
H3	Data Interfaces ( <i>to be added</i> )	H3-1
H4	Magnetic Tape	H4-1
H4.1	DECTape TD10	H4-1
H4.2	Standard Magnetic Tape TM10	H4-3
	Tape transport TU10	H4-4
	Tape transport TU20	H4-6
	Tape transport TU30	H4-7
	Tape transport TU40	H4-8
H5	Disks and Drums ( <i>to be added</i> )	H5-1
H6	Data Communications	H6-1
	Data line scanner DC10	H6-1
	Single synchronous line unit DS10	H6-2
H7	Cleaning Procedures	H7-1
H7.1	Tape Equipment	H7-1
	DECTape	H7-1
	Standard magnetic tape	H7-2
	Tapes	H7-3
H7.2	Disk Packs	H7-3
H7.3	Other Equipment	H7-4
	Paper tape reader and punch	H7-4
	Line printer	H7-4
	Card reader and punch	H7-4
INDEX		I-1

# 1

## Introduction

The DECsystem-10 is a general purpose, stored program computing system that includes at least one PDP-10 central processor, a memory, and a variety of peripheral equipment such as paper tape reader and punch, teletypewriter, card reader and punch, line printer, DECTape, magnetic tape, disk, drum, display and data communications equipment. Each central processor is the control unit for an entire large-scale subsystem, in which it is connected by an in-out bus to its own peripheral equipment and by a memory bus to one or more memory units in a main memory, some of whose units may be shared by several processors. Within the subsystem the central processor governs all peripheral equipment, sequences the program, and performs all arithmetic, logical and data handling operations. Besides central processors, there are also direct-access processors, which have much more limited program capability and serve to connect large, fast peripheral devices to memory bypassing the central processor. Every direct-access processor is connected to the in-out bus of some central processor, to which it appears as an in-out device; the direct-access processor is also connected to memory by its own memory bus, and to its peripheral equipment by a device bus. The DECsystem-10 may also contain peripheral subsystems, such as for data communications, which are themselves based on small computers; such a subsystem in toto is connected to a PDP-10 in-out bus and is treated by the PDP-10 as a peripheral device. Unless otherwise specified, the words "processor" and "central processor" refer to the large-scale PDP-10 central processor, and "in-out bus" refers to the bus from the central processor to its peripheral equipment. A direct-access processor and the bus to its peripheral equipment are all always referred to by their names, *eg* the DF10 data channel and its channel bus (often a direct-access processor and device control are a single unit).

At present there are two types of PDP-10 central processors, the KA10 and the KI10. The latter is faster and more powerful, having a somewhat larger instruction repertoire including double precision floating point. Both processors handle words of thirty-six bits, which are stored in a memory whose maximum capacity depends upon the addressing capability of the processor. Internally both processors use 18-bit addresses and can thus reference 262,144 word locations in memory. This is the total addressing capability of the KA10, but in the KI10 it is only the virtual address space available to a single program. Paging hardware supplies four additional address bits to map pages in the program virtual address space into pages anywhere in a physical memory that is sixteen times as large. Thus for a number of different programs, the processor actually has access to a

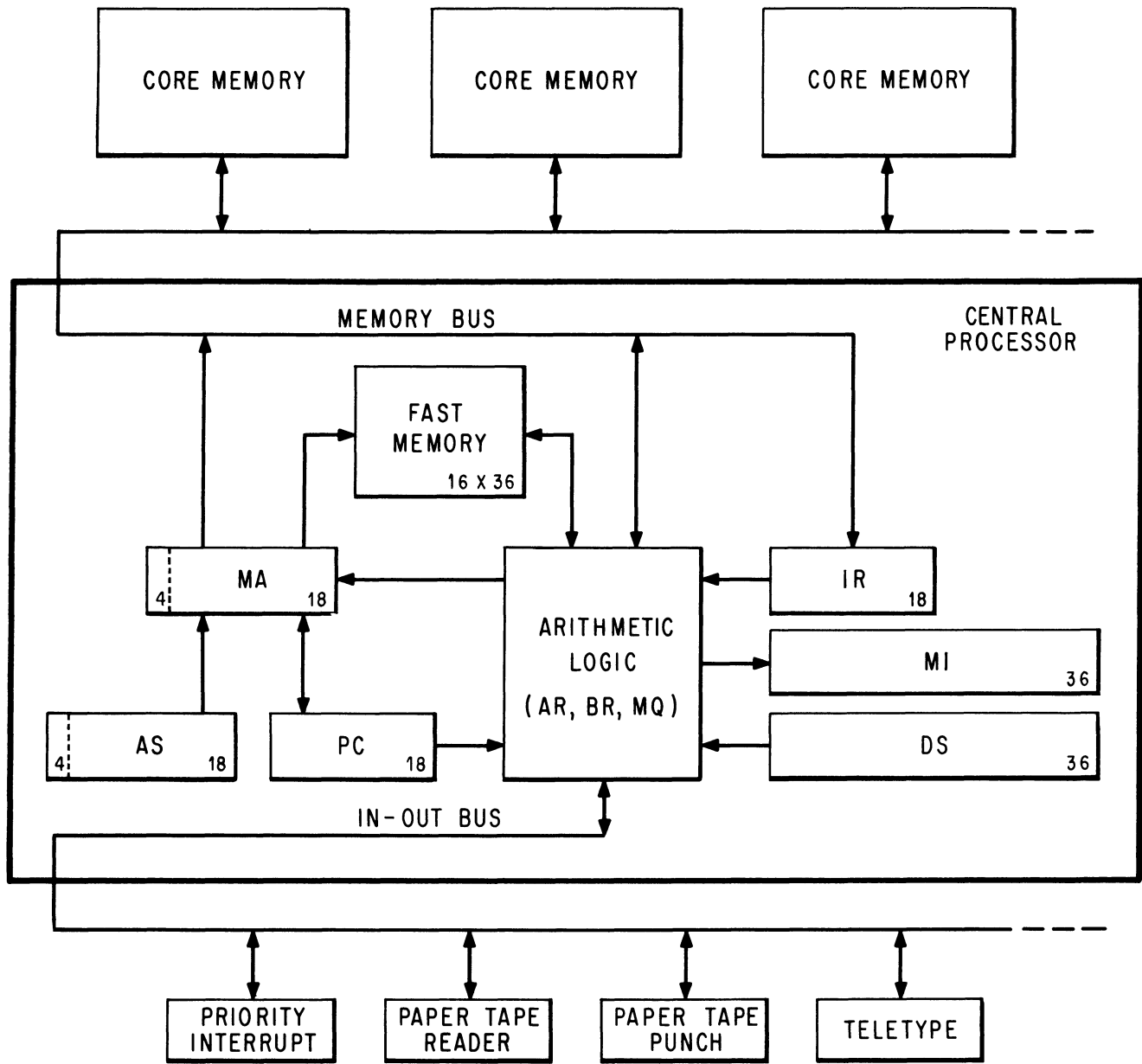
Confusion could result only in a chapter dealing with a small-computer subsystem. Here the small processor is usually referred to by its name (PDP-8, PDP-11) and the words "computer" and "memory" refer to the small computer. To differentiate, the PDP-10 is referred to by its name or as the "DECsystem-10 central processor", and the large scale memory connected to the PDP-10 memory bus is referred to as "DECsystem-10 main memory".

physical memory with a capacity of 4,194,304 words. Storage in memory is usually in the form of 37-bit words, the extra bit producing odd parity for the word. The bits of a word are numbered 0–35, left to right (most significant to least significant), as are the bits in the registers that handle the words. The processor can handle half words, wherein the left half comprises bits 0–17, the right half, bits 18–35. There is also hardware for byte manipulation – a byte is any contiguous set of bits within a word. KA10 registers that hold addresses have eighteen bits, numbered 18–35 according to the position of an address in a word. KI10 internal address registers have eighteen bits, but a register that must supply a complete address to physical memory has twenty-two bits (numbered 14–35). Words are used either as computer instructions in the program, as addresses, or as operands (data for the program).

Of the internal registers shown in the illustration on the next page, only PC, the 18-bit program counter, is directly relevant to the programmer. The processor performs a program by executing instructions retrieved from the locations addressed by PC. At the beginning of each instruction PC is incremented by one so that it normally contains an address one greater than the location of the current instruction. Sequential program flow is altered by changing the contents of PC, either by incrementing it an extra time in a skip instruction or by replacing its contents with the value specified by a jump instruction. Also of importance to the programmer are the sense switches and the 36-bit data switch register DS on the processor console: through these switches the program can read information supplied by the operator. The processor also contains flags that detect various types of errors, including several types of overflow in arithmetic and pushdown operations, and provide other information of interest to the programmer.

The processor has other registers but the programmer is not usually concerned with them except when manually stepping through a program to debug it. By means of the address switch register AS, the operator can examine the contents of, or deposit information into, any memory location; stop or interrupt the program whenever a particular location is referenced; and through AS the operator can supply a starting address for the program. Through the memory indicators MI the program can display data for the operator. The instruction register IR contains the left half of the current instruction word, *ie* all but the address part. The memory address register MA supplies the address for every memory access. The heart of the processor is the arithmetic logic, principally the 36-bit arithmetic register AR. This register takes part in all arithmetic, logical and data handling operations; all data transfers to and from memory, peripheral equipment and console are made via AR. Associated with AR are an extremely fast full adder, a buffer register BR that holds a second operand in many arithmetic and logical instructions, a multiplier-quotient register MQ that serves primarily as an extension of AR for handling double length operands, and smaller registers that handle floating point exponents and control shift operations and byte manipulation. In the KI10, AR and the adder each have a 28-bit left extension for handling double precision floating point numbers.

From the point of view of the programmer however the arithmetic logic can be regarded as a black box. It performs almost all of the operations



DECSYSTEM-10 SIMPLIFIED

necessary for the execution of a program, but it never retains any information from one instruction to the next. Computations performed in the black box either affect control elements such as PC and the flags, or produce results that are always sent to memory and must be retrieved by the processor if they are to be used as operands in other instructions.

An instruction word has only one 18-bit address field for addressing any location throughout all of the virtual address space. But most instructions have two 4-bit fields for addressing the first sixteen memory locations. Any instruction that requires a second operand has an accumulator address field,

which can address one of these sixteen locations as an accumulator; in other words as though it were a result held over in the processor from some previous instruction (the programmer usually has a choice of whether the result of the instruction will go to the location addressed as an accumulator or to that addressed by the 18-bit address field, or to both). Every instruction has a 4-bit index register address field, which can address fifteen of these locations for use as index registers in modifying the 18-bit memory address (a zero index register address specifies no indexing). Although all computations on both operands and addresses are performed in the single arithmetic register AR, the computer actually has sixteen accumulators, fifteen of which can double as index registers. The factor that determines whether one of the first sixteen locations in memory is an accumulator or an index register is not the information it contains nor how its contents are used, but rather how the location is addressed. These first sixteen memory locations are not actually in core memory, but are rather in a fast solid state memory contained in the processor. This allows much quicker access to these locations whether they are addressed as accumulators, index registers or ordinary memory locations. They can even be addressed from the program counter, gaining faster execution for a short but oft-repeated subroutine.

The KI10 actually has four fast memory blocks, but only one of these is available to a program at any given time.

Besides the registers that enter into the regular execution of the program and its instructions, the processor has a priority interrupt system and equipment to facilitate time sharing. The interrupt system facilitates processor control of the peripheral equipment by means of a number of priority-ordered channels over which external signals may interrupt the normal program flow. The processor acknowledges an interrupt request by executing the instruction contained in a particular location for the channel or doing some special operation specified by the device (such as incrementing the contents of a memory location). Assignment of channels to devices is entirely under program control. One of the devices to which the program can assign a channel is the processor itself, allowing internal conditions such as overflow or a parity error to signal the program.

**Time Sharing.** Inherent in the basic machine hardware are restrictions that apply universally: only certain instructions can be used to respond to a priority interrupt, and certain memory locations have predefined uses. But above this fundamental level, the time share hardware provides for different modes of processor operation and establishes certain instruction restrictions and memory restrictions so that the processor can handle a number of user programs (programs run in user mode) without their interfering with one another. The memory restrictions are dependent to a great extent on the processor, but the instruction restrictions are not, and these are relatively obvious: a program that is sharing the system with others cannot usually be allowed to halt the processor or to operate the in-out equipment arbitrarily. A program that runs in executive mode — the Monitor — is responsible for scheduling user programs, servicing interrupts, handling input-output needs, and taking action when control is returned to it from a user program. Any violation of an instruction or memory restriction by a user transfers control back to the Monitor. Dedication of the entire facility to a single purpose, in other words with only one user, is equivalent to

The KI10 allows unrestricted in-out with a limited number of devices for special real time applications.

operation in executive mode (specifically kernel mode in the KI10).

The KA10 has the two modes discussed above, user and executive. It also has protection and relocation hardware to confine the user virtual address space within a particular range, and to relocate user memory references to the appropriate area in physical core. A user ordinarily has access to two separate core areas, one of which may be write-protected, *ie* the user cannot alter its contents.

The KI10 has paging hardware for the mapping of pages from the limited virtual address space into pages anywhere in physical memory. A page map for each program specifies not only the correspondence from virtual address to physical address, but also whether an individual page is accessible or not, alterable or not, and public or concealed. Both user and executive modes are subdivided according to whether the program is running in a public area or a concealed area. Within user mode these are the public and concealed modes; within executive mode, the supervisor and kernel modes. A program in concealed mode can reference all of accessible user memory, but the public program cannot reference the concealed area except to transfer control into it at certain legitimate entry points.

In kernel mode the Monitor handles the in-out for the system, handles priority interrupts, constructs page maps, and performs those functions that affect all users. This mode has no instruction restrictions and the program can even address some of memory directly (*ie* unpagged); in the paged address space, individual pages may be restricted as inaccessible or write-protected, but it is the kernel mode program that establishes these restrictions. In supervisor mode the Monitor handles the general management of the system and those functions that affect only one user at a time. This mode has essentially the same instruction and memory restrictions as user mode, although the supervisor mode program can read, but not alter, the concealed areas; in this way the kernel mode Monitor supplies the supervisor program with information the latter cannot alter (even though the information is not write-protected from the kernel program). In either mode the Monitor automatically uses fast memory block 0 (the hardware requires this). The kernel program is responsible for assigning fast memory blocks to the various user programs: ordinarily blocks 2 and 3 are for special real time applications, and block 1 is assigned to all other users.

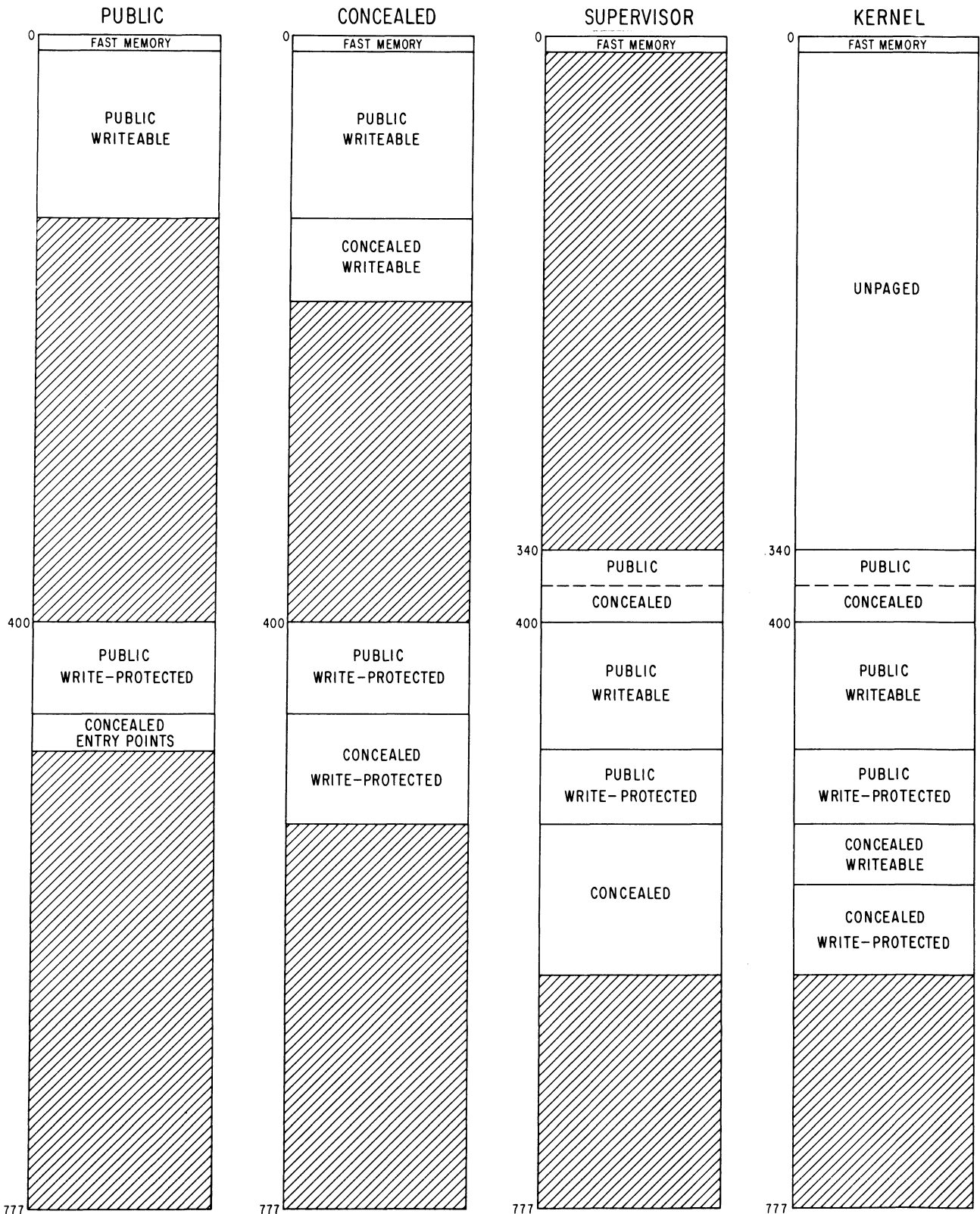
The illustration on the next page shows a typical layout of the virtual address space for the various modes. The space is 256K, made up of 512 pages numbered 0-777 octal. Any program can address locations 0-17 as these are in a fast memory block and are completely unrestricted (although the same addresses may be in different blocks for different programs). The public mode user program operates in the public area, part of which may be write-protected. The public program cannot access any locations in the concealed areas except to fetch instructions from prescribed entry points. The concealed mode user program has access to both public and concealed areas, but it cannot alter any write-protected location whether public or concealed, and fetching an instruction from the public area automatically returns the processor to public mode.

The supervisor mode program is confined within the paged area of the address space, pages 340 and above. Part of the public area in this space may

The concealed area would ordinarily be used for proprietary programs that the user can call but cannot read or alter.

USER MODE

EXECUTIVE MODE



SHADED AREAS ARE INACCESSIBLE

TYPICAL VIRTUAL ADDRESS SPACE CONFIGURATION



be write-protected, but the program can read information in the concealed areas — it cannot alter any location in a concealed area whether that area is write-protected or not. Pages 340–377 constitute the per-process area, which contains information specific to individual users and whose mapping accompanies the user page map. In other words the physical memory corresponding to these virtual pages can be changed simply by switching from one user to another, rather than the Monitor changing its own page map. The kernel mode program can access all of the unpagged area without restriction and can reference all of the accessible paged area, both public and concealed, with the usual restriction that it cannot alter a write-protected area. As in the case of concealed user mode, fetching an instruction from a public area returns control to supervisor mode.

### 1.1 NUMBER SYSTEM

The program can interpret a data word as a 36-digit, unsigned binary number, or the left and right halves of a word can be taken as separate 18-bit numbers. The PDP-10 repertoire includes instructions that effectively add or subtract one from both halves of a word, so the right half can be used for address modification when the word is addressed as an index register, while the left half is used to keep a control count.

The standard arithmetic instructions in the PDP-10 use two's complement, fixed point conventions to do binary arithmetic. In a word used as a number, bit 0 (the leftmost bit) represents the sign, 0 for positive, 1 for negative. In a positive number the remaining 35 bits are the magnitude in ordinary binary notation. The negative of a number is obtained by taking its two's complement. If  $x$  is an  $n$ -digit binary number, its two's complement is  $2^n - x$ , and its one's complement is  $(2^n - 1) - x$ , or equivalently  $(2^n - x) - 1$ . Subtracting a number from  $2^n - 1$  (ie, from all 1s) is equivalent to performing the logical complement, ie changing all 0s to 1s and all 1s to 0s. Therefore, to form the two's complement one takes the logical complement (usually referred to merely as the complement) of the entire word including the sign, and adds 1 to the result. In a negative number the sign bit is 1, and the remaining bits are the two's complement of the magnitude.

$$+153_{10} = +231_8 = \boxed{000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 010\ 011\ 001}$$

035

$$-153_{10} = -231_8 = \boxed{111\ 111\ 111\ 111\ 111\ 111\ 111\ 111\ 111\ 101\ 100\ 111}$$

035

Zero is represented by a word containing all 0s. Complementing this number produces all 1s, and adding 1 to that produces all 0s again. Hence there is only one zero representation and its sign is positive. Since the numbers are symmetrical in magnitude about a single zero representation, all even numbers both positive and negative end in 0, all odd numbers in 1 (a

The adder actually acts as though the words represented 36-bit unsigned numbers, ie the signs are treated just like magnitude bits. In the absence of a carry into the sign stage, adding two numbers with the same sign produces a plus sign in the result. The presence of a carry gives a positive answer when the summands have different signs. The result has a minus sign when there is a carry into the sign bit and the summands have the same sign, or the summands have different signs and there is no carry.

Thus the program can interpret the numbers processed in fixed point addition and subtraction as signed numbers with 35 magnitude bits or as unsigned 36-bit numbers. A computation on signed numbers produces a result that

is correct as an unsigned 36-bit number even if overflow occurs, but the hardware interprets the result as a signed number to detect overflow. Adding two positive numbers whose sum is greater than or equal to  $2^{35}$  gives a negative result, indicating overflow; but that result, which has a 1 in the sign bit, is the correct answer interpreted as a 36-bit unsigned number in positive form. Similarly adding two negatives gives a result which is always correct as an unsigned number in negative form.

Multiplication produces a double length product, and the programmer must remember that discarding the low order part of a double length negative leaves the high order part in correct twos complement form only if the low order part is null.

This convention for bit 0 of the low order word is inconsistent with that used for floating point arithmetic [see below]. This should cause no problem however, as fixed divide ignores bit 0 of the low order word in a double length dividend.

number all 1s represents  $-1$ ). But since there are the same number of positive and negative numbers and zero is positive, there is one more negative number than there are nonzero positive numbers. This is the most negative number and it cannot be produced by negating any positive number (its octal representation is  $400000\ 000000_8$  and its magnitude is one greater than the largest positive number).

If ones complements were used for negatives one could read a negative number by attaching significance to the 0s instead of the 1s. In twos complement notation each negative number is one greater than the complement of the positive number of the same magnitude, so one can read a negative number by attaching significance to the rightmost 1 and attaching significance to the 0s at the left of it (the negative number of largest magnitude has a 1 in only the sign position). In a negative integer, 1s may be discarded at the left, just as leading 0s may be dropped in a positive integer. In a negative fraction, 0s may be discarded at the right. So long as only 0s are discarded, the number remains in twos complement form because it still has a 1 that possesses significance; but if a portion including the rightmost 1 is discarded, the remaining part of the fraction is now a ones complement.

The computer does not keep track of a binary point — the programmer must adopt a point convention and shift the magnitude of the result to conform to the convention used. Two common conventions are to regard a number as an integer (binary point at the right) or as a proper fraction (binary point at the left); in these two cases the range of numbers represented by a single word is  $-2^{35}$  to  $2^{35} - 1$  or  $-1$  to  $1 - 2^{-35}$ . Since multiplication and division make use of double length numbers, there are special instructions for performing these operations with integral operands.

The format for double length fixed point numbers is just an extension of the single length format. The magnitude (or its twos complement) is the 70-bit string in bits 1–35 of the high and low order words. Bit 0 of the high order word is the sign, and bit 0 of the low order word is made equal to the sign. The range for double length integers and proper fractions is thus  $-2^{70}$  to  $2^{70} - 1$  and  $-1$  to  $1 - 2^{-70}$ .

**Floating Point Arithmetic.** The KI10 has hardware for processing single and double precision floating point numbers; the KA10 can generally process only single precision numbers, although the hardware does include features that facilitate double precision arithmetic by software routines. The same format is used for a single precision number and the high order word of a double precision number. A floating point instruction interprets bit 0 as the sign, but interprets the rest of the word as an 8-bit exponent and a 27-bit fraction. For a positive number the sign is 0, as before. But the contents of bits 9–35 are now interpreted only as a binary fraction, and the contents of bits 1–8 are interpreted as an integral exponent in excess 128 ( $200_8$ ) code. Exponents from  $-128$  to  $+127$  are therefore represented by the binary equivalents of 0 to 255 ( $0-377_8$ ). Floating point zero and negatives are represented in exactly the same way as in fixed point: zero by a word containing all 0s, a negative by the twos complement. A negative number has a 1 for its sign and the twos complement of the fraction, but since every fraction must ordinarily contain a 1 unless the entire number is zero (see

below), it has the ones complement of the exponent code in bits 1–8. Since the exponent is in excess 128 code, an actual exponent  $x$  is represented in a positive number by  $x + 128$ , in a negative number by  $127 - x$ . The programmer, however, need not be concerned with these representations as the hardware compensates automatically. *Eg.* for the instruction that scales the exponent, the hardware interprets the integral scale factor in standard twos complement form but produces the correct ones complement result for the exponent.

$$+153_{10} = +231_8 = +.462_8 \times 2^8 =$$

0	10 001 000	100 110 010 000 000 000 000 000 000	35
0 1	8 9		

$$-153_{10} = -231_8 = -.462_8 \times 2^8 =$$

1	01 110 111	011 001 110 000 000 000 000 000 000	35
0 1	8 9		

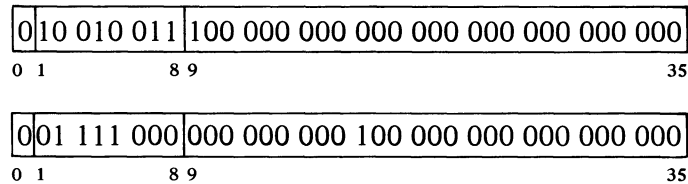
Except in special cases the floating point instructions assume that all nonzero operands are normalized, and they normalize a nonzero result. A floating point number is considered normalized if the magnitude of the fraction is greater than or equal to  $\frac{1}{2}$  and less than 1. The hardware may not give the correct result if the program supplies an operand that is not normalized or that has a zero fraction with a nonzero exponent.

Single precision floating point numbers have a fractional range in magnitude of  $\frac{1}{2}$  to  $1 - 2^{-27}$ . Increasing the length of a number to two words does not significantly change the range but rather increases the precision; in any format the magnitude range of the fraction is  $\frac{1}{2}$  to 1 decreased by the value of the least significant bit. In all formats the exponent range is  $-128$  to  $+127$ .

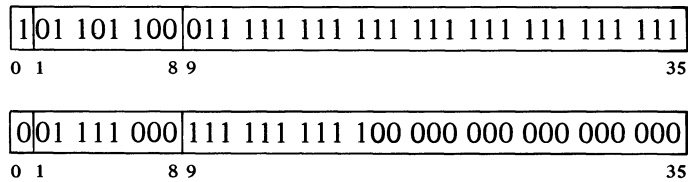
The precaution about truncation given for fixed point multiplication applies to most floating point operations as they produce extra length results; but here the programmer may request rounding, which automatically restores the high order part to twos complement form if it is negative. In single precision division the two words of the result are quotient and remainder, but in the other operations they form a double length number which is stored in two accumulators if the instruction is executed in “long” mode. (Long mode division uses a double length dividend.) A double length number used by the single precision instructions is in software double precision format. As such it contains a 54-bit fraction, half of which is in bits 9–35 of each word. The sign and exponent are in bits 0 and 1–8 respectively of the word containing the more significant half, and the standard twos complement is used to form the negative of the entire 63-bit string. In the remaining part of the less significant word, bit 0 is 0, and bits 1–8 contain a number 27 less than the exponent, but this is expressed in positive form even though bits 9–35 may be part of a negative fraction. *Eg.* the number  $2^{18} + 2^{-18}$  has this two-word representation in software

An instruction that generates a double length result sets the low word exponent part to zero whenever the low order fraction is zero, and sets the whole low order word to zero whenever the low order exponent overflows or underflows.

double precision format:



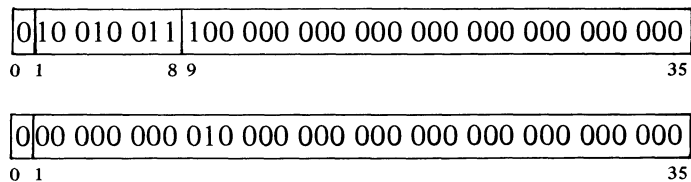
whereas its negative is



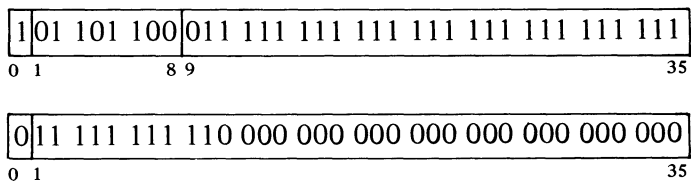
Essentially there are five number formats. Fixed point additive operations can be regarded as being performed on 36-bit unsigned numbers, which are equivalent to logical words. Otherwise fixed point arithmetic uses the fixed point format; numbers are single length with the exception that products and dividends can be double length, and there is provision for shifting a double length operand arithmetically. Double length format is an extension of single length format to two 36-bit words.

Single precision floating point instructions use two formats: single precision floating point format and *software* double precision floating point format. The latter appears only in the result of a long mode add, subtract or multiply, as the dividend in a long mode divide, and as the operand for an instruction that negates a number specifically in that format. Operands for double precision floating point instructions are exclusively in *hardware* double precision floating point format (and these instructions are not available on the KA10).

The double precision floating point instructions use a more straightforward double length format with greater precision than is allowed by the software format. For these instructions all operands and results are double length, and all instructions except division calculate a triple length answer, which is rounded to double length with the appropriate adjustment for a two's complement negative. In hardware double precision format the high order word is the same as a single precision number, and bits 1–35 of the low order word are simply an extension of the fraction, which is now sixty-two bits. Bit 0 is ignored. The number used above as an example of software double precision format has this representation in hardware format:



and its negative is



## 1.2 INSTRUCTION FORMAT

In all but the input-output instructions, the nine high order bits (0–8) specify the operation, and bits 9–12 usually address an accumulator but are sometimes used for special control purposes, such as addressing flags. The



out only for words indicated in the text as having the format shown. Do not assume that the procedure is used for any miscellaneous pointer simply because it happens to contain an address [see page C-2].



**PLEASE READ THIS**

*The calculation of  $E$  is the first step in the execution of every instruction. No other action taken by any instruction, no matter what it is, can possibly precede that calculation. There is absolutely nothing whatsoever that any instruction can do to any accumulator or memory location that can in any way affect its own effective address calculation.*

tion word is 0 and no memory reference is necessary, then  $Y$  is not an address. It may be a mask in some kind of test instruction, conditions to be sent to an in-out device, or part of it may be the number of places to shift in a shift or rotate instruction or the scale factor in a floating scale instruction. Even when modified by an index register, bits 18–35 do not contain an address when  $I$  is 0. But when  $I$  is 1, the number determined from bits 14–35 is an indirect address no matter what type of information the instruction requires, and the word retrieved in any step of the calculation contains an indirect address so long as  $I$  remains 1. When a location is found in which  $I$  is 0, bits 18–35 (perhaps modified by an index register) contain the desired effective mask, effective conditions, effective shift number, or effective scale factor. Many of the instructions that usually reference memory for an operand even have an “immediate” mode in which the result of the effective address calculation is itself used as a half word operand instead of a word taken from the memory location it addresses.

The important thing for the programmer to remember is that the same calculation is carried out for every instruction regardless of the type of information that must be specified for its execution, or even if the result is ignored. In the discussion of any instruction,  $E$  refers to the actual quantity derived from  $I$ ,  $X$  and  $Y$  and used in the execution of the instruction, be it the entire half word as in the case of an address, immediate operand, mask or conditions, or only part of it as in a shift number or scale factor.

### 1.3 MEMORY

The internal timing for each in-out device and each memory is entirely independent of the central processor. Because core memory readout is destructive, every word read must be written back in unless new information is to take its place. But the processor need never wait the entire cycle time. To read, it waits only until the information is available and then continues its operations while the memory performs the write portion of the cycle; to write, it waits only until the data is accepted, and the memory then performs an entire cycle to clear and write. To save time in an instruction that fetches an operand and then writes new data into the same location, the memory executes a read-modify-write cycle in which it performs only the read part initially and then completes the cycle when the processor supplies the new data. This procedure is not used however in a lengthy instruction (such as multiply or divide), which would tie up a memory that may be needed by some other processor. Such instructions instead request separate read and write access. The KI10 further increases the speed of memory operation by overlapping memory cycles. *Eg* it can start one memory to read a word before receiving a word previously requested from a different memory.

Access times for the accumulator-index register locations are decreased considerably by substitution of a fast memory (contained in the processor) for the first sixteen core locations. Readout is nondestructive, so the fast memory has no basic cycle: the processor reads or writes a word directly (note: to write, the KA10 must first clear the location and then load it).

The following table gives the characteristics of the various memories. Modify completion is the time to finish a read-modify-write cycle after the processor supplies the new data. Times are in microseconds and include the delay introduced by ten feet (three meters) of cable. Fast memory times are for referencing as a memory location (18-bit address); when a fast memory location is addressed as an accumulator or index register, the access time is considerably shorter.

	<i>Read Access</i>	<i>Write Access</i>	<i>Cycle</i>	<i>Modify Completion</i>	<i>Size</i>
161 Core Memory	2.5	.49	4.7	2.69	16K
163 Core Memory	.94	.49	1.8	1.33	16K
164 Core Memory } MB10 Core Memory }	.60*	.20*	1.65*	.97	16K
MA10 Core Memory	.61	.20	1.00	.57	16K
MD10 Core Memory	.83	.33	1.8	1.23	32-128K
ME10 Core Memory	.61	.20	1.00	.65	16K
MF10 Core Memory	.61	.20	1.00	.63	32K, 64K
KA10 Fast Memory	.21	.21			16
KI10 Fast Memory	.28	.0			16

\*Add .1 in a multiprocessor system.

MD10 can be increased in units of 32K up to 128K.

KI10 access to accumulators and index registers effectively takes no time — it is done in parallel with instruction operations that are required anyway. Retrieval of instructions or memory operands from fast memory is generally not worthwhile because of the extensive overlapping that speeds up core access. However, except in instructions that use two accumulators, storage of a memory operand in fast memory not only takes no time but actually decreases slightly the nonmemory time.

From the simple hardware addressing point of view, the entire memory is a set of contiguous locations whose addresses range from zero to a maximum dependent upon the capacity of the particular installation. In a system with the greatest possible capacity, the largest KA10 address is octal 777777, decimal 262,143; the largest KI10 address is 17777777, decimal 4,194,303. (Addresses are always in octal notation unless otherwise specified.) But the whole memory would usually be made up of a number of core memories of different capacities as listed above. Hence a given address actually selects a particular memory and a specific location within it. For a 16K memory with 18-bit addressing, the high order four address bits select the memory, the remaining fourteen bits address a single location in it; selecting a 32K memory takes three bits, leaving fifteen for the location. The times given above assume the addressed memory is idle when access is requested. To avoid waiting for a previously requested memory cycle to end, the program can make consecutive requests to different memories by taking instructions from one memory and data from another. All memories can be interleaved in pairs in such a way that consecutive addresses actually alternate between the two memories in the pair (thus increasing the probability that consecutive references are to different memories). Appropriate switch settings at the memories interchange the least significant address bits in the memory selection and location parts, so that in any two memories numbered  $n$  and  $n + 1$  where  $n$  is even, all even addresses are locations in the first memory, all odd addresses are locations in the second. Hence memories 0 and 1 can be interleaved as can 6 and 7, but not 3 and 4 or 5 and 7. Some memories can be interleaved in contiguous groups of four, where the number of the first memory in the

Information on memory setup is given in Appendix G.

group is divisible by four (eg memories 0–3 or 14–17). In this case all addresses ending in 0 or 4 reference the first memory in the group, all ending in 1 or 5 reference the second, and so forth.

In terms of the virtual address space (the addresses that can be specified within the limits of the instruction format) or the subset of it that is accessible to a user, the situation may be quite different. In the KA10 the user program has a continuous address space beginning at 0, or two continuous spaces beginning at 0 and 400000. In the KI10 the possible program address space is the set of all 18-bit addresses just as in the KA10, but which addresses a program can actually use depends entirely upon which of the 512 virtual pages (512 words per page) are accessible to it. For a so-called “small user”, the accessible space must lie within the ranges 0–37777 and 400000–437777. In any event all programs have access to fast memory, whether as accumulators, index registers or ordinary memory references (*ie* addresses 0–17 are never restricted or relocated).

**KI10 Memory Allocation.** The KI10 hardware defines the use of certain memory locations, but most are relative to pages whose physical location is specified by the Monitor. The auto restart uses location 70. The only other physical locations uniquely defined by the hardware are those in fast memory, whose addresses are the same for all programs: location 0 holds a pointer word during a bootstrap readin, 0–17 can be addressed as accumulators, and 1–17 can be addressed as index registers. The only addresses uniquely specified in the user virtual space are for user local UOs – locations 40 and 41.

All other addresses defined by the hardware, for use in page mapping, responding to priority interrupts, or other hardware-oriented situations, are to locations within a page specified by the Monitor for a particular user (including itself). For each user the Monitor keeps a process table, which must begin at location 0 of some page. The locations used by the hardware for the page map, traps, etc. of a given user are all in the first page of the table for that user. The parts of a user process table not used by the hardware may be used by the Monitor to keep accumulators (when the user is not running), a pushdown list that the Monitor uses for the job, and various user statistics such as running time, memory space, billing information, and job tables. The detailed configuration of the hardware-defined parts of the process tables (user and executive) is given in §2.15.

**KA10 Memory Allocation.** The use of certain memory locations is defined by the KA10 hardware.

0	Holds a pointer word during a bootstrap readin
0–17	Can be addressed as accumulators
1–17	Can be addressed as index registers
40–41	Trap for unimplemented user operations (UOs)
42–57	Priority interrupt locations
60–61	Trap for remaining unimplemented operations: these include the unassigned instruction codes that are reserved for future use, and also the byte manipulation and floating point instructions when the hardware for them is not installed

The kernel mode program can always address locations 0–337777 as these are un-paged. Virtual pages 340 and above are mapped.

The Monitor keeps a user process table for each user program and one executive process table for itself for each KI10 processor. In the text, the phrase “the user process table” refers to the process table currently specified by the Monitor as the one for the user, even if that user is not currently running. The Monitor must also specify the whereabouts of the executive process table for the processor under consideration.

The initial control word address for the DF10 Data Channel must be less than 1000.



140–161 Allocated to second processor if connected (same use as 40–61 for first processor)

In a user program the trap for a local UWO is relocated to locations 40 and 41 of the user area; a Monitor UWO uses unrelocated locations. All other addresses listed are for physical (unrelocated) locations.

All information given in this manual about memory locations 40–61 for a KA10 applies instead to locations 140–161 for programming a second KA10 connected to the same memory.

## 1.4 PROGRAMMING CONVENTIONS

The computer has five instruction classes: data transmission, logical, arithmetic, program control and in-out. The instructions in the in-out class control the peripheral equipment, and also control the priority interrupt and time sharing, control and read the processor flags, and communicate with the console. The next chapter describes all instructions mentioned above, presents a general description of input-output, and describes the effects of the in-out instructions on the processor, priority interrupt and time share hardware. Effects of in-out instructions on particular peripheral devices are discussed with the devices.

The MACRO-10 assembly program recognizes a number of mnemonics and other initial symbols that facilitate constructing complete instruction words and organizing them into a program. In particular there are mnemonics for the instruction codes (Appendix A), which are six bits in in-out instructions, otherwise nine or thirteen bits. *Eg* the mnemonic

MOVNS

assembles as 213000 000000, and

MOVNS 2570

assembles as 213000 002570. This latter word, when executed as an instruction, produces the two's complement negative of the word in memory location 2570.

The assembler translates every statement into a 36-bit word, placing 0s in all bits whose values are unspecified.

### NOTE

Throughout this manual all numbers representing instruction words, register contents, codes and addresses are always octal, and any numbers appearing in program examples are octal unless otherwise indicated. On the other hand, the ordinary use of numbers in the text to count steps in an operation or to specify word or byte lengths, bit positions, exponents, etc employs standard decimal notation.

The initial symbol @ preceding a memory address places a 1 in bit 13 to produce indirect addressing. The example given above uses direct addressing, but

MOVNS @2570

assembles as 213020 002570, and produces indirect addressing. Placing the

number of an index register (1–17) in parentheses following the memory address causes modification of the address by the contents of the specified register. Hence

```
MOVNS @2570(12)
```

which assembles as 213032 002570, produces indexing using index register 12, and the processor then uses the modified address to continue the effective address calculation.

An accumulator address (0–17) precedes the memory address part (if any) and is terminated by a comma. Thus

```
MOVNS 4,@2570(12)
```

assembles as 213232 002570, which negates the word in location *E* and stores the result in both *E* and in accumulator 4. The same procedure may be used to place 1s in bits 9–12 when these are used for something other than addressing an accumulator, but mnemonics are available for this purpose.

The device code in an in-out instruction is given in the same manner as an accumulator address (terminated by a comma and preceding the address part), but the number given must correspond to the octal digits in the word (000–774). Mnemonics are however available for all standard device codes. To control the priority interrupt system whose code is 004, one may give

```
CONO 4,1302
```

which assembles as 700600 001302, or equivalently

```
CONO PI,1302
```

The programming examples in this manual use the following addressing conventions:

- ◆ A colon following a symbol indicates that it is a symbolic location name.

```
A:      ADD    6,5704
```

indicates that the location that contains ADD 6,5704 may be addressed symbolically as A.

- ◆ The period represents the current address, *eg*

```
ADD    5,.+2
```

is equivalent to

```
A:      ADD    5,A+2
```

- ◆ Square brackets specify the contents of a location, leaving the address of the location implicit but unspecified. *Eg*

```
ADD    12,[7256004]
```

and

```
ADD    12,A
```





# 2

## Central Processor

This chapter describes all PDP-10 instructions but does not discuss the effects of those in-out instructions that address specific peripheral devices. In the description of each instruction, the mnemonic and name are at the top, the format is in a box below them. The mnemonic assembles to the word in the box, where bits in those parts of the word represented by letters assemble as 0s. The letters indicate portions that must be added to the mnemonic to produce a complete instruction word.

For many of the non-IO instructions, a description applies not to a unique instruction with a single code in bits 0-8, but rather to an instruction set defined as a basic instruction that can be executed in a number of modes. These modes define properties subsidiary to the basic operation; eg in data transmission the mode specifies which of the locations addressed by the instruction is the source and which the destination of the data, in test instructions it specifies the condition that must be satisfied for a jump or skip to take place. The mnemonic given at the top is for the basic mode; mnemonics for the other forms of the instruction are produced by appending letters directly to the basic mnemonic. Following the description is a table giving the mnemonics and octal codes (bits 0-8) for the various modes.

In a description  $E$  refers to the effective address, half word operand, mask, conditions, shift number or scale factor calculated from the  $I$ ,  $X$  and  $Y$  parts of the instruction word. In an instruction that ordinarily references memory, a reference to  $E$  as the source of information means that the instruction retrieves the word contained in location  $E$ ; as a destination it means the instruction stores a word in location  $E$ . In the immediate mode of these instructions, the effective half word operand is usually treated as a full word that contains  $E$  in one half and zero in the other, and is represented either as  $0, E$  or  $E, 0$  depending upon whether  $E$  is in the right or left half.

Most of the non-IO instructions can address an accumulator, and in the box showing the format this address is represented by  $A$ ; in the description, "AC" refers to the accumulator addressed by  $A$ . "AC left" and "AC right" refer to the two halves of AC. If an instruction uses two accumulators, these have addresses  $A$  and  $A+1$ , where the second address is 0 if  $A$  is 17. In some cases an instruction uses an accumulator only if  $A$  is nonzero: a zero address in bits 9-12 specifies no accumulator.

The instructions are described in terms of their effects as seen by the user in a normal program situation, and on the assumption that nothing is amiss — the program is not attempting to reference a memory that does not exist or to write in a protected area of core. In general, all descriptions apply equally

Letters representing modes are suffixes, which produce new mnemonics that are recognized as distinct symbols by the assembler.



### PLEASE READ THIS

*The calculation of  $E$  is the first step in the execution of every instruction. No other action taken by any instruction, no matter what it is, can possibly precede that calculation. There is absolutely nothing whatsoever that any instruction can do to any accumulator or memory location that can in any way affect its own effective address calculation.*

well to operation in executive mode. For completeness, the effects of restrictions on certain instructions are noted, as are the effects of executing instructions in special circumstances. But for the details of programming in such special situations the reader must look elsewhere. In particular, §2.9 discusses trapping, §2.13 describes the priority interrupt, and §§2.15 and 2.16 describe the special effects and restrictions associated with program and memory management in the KI10 and the KA10 respectively.

To minimize processor execution time the programmer should minimize the number of memory references and the number of shifts and other iterative operations. When there is a choice of actions to be taken on the basis of some test, the conditions tested should be set up so that the action that results most often takes the least time. There are also various subtleties that affect timing (such as the nature of the arithmetic algorithms), but these are generally not worth considering except in very special circumstances (to determine the effect often takes more than the time saved).

No execution times are given with the instruction descriptions as the time may vary greatly depending upon circumstances. At the outset the time depends upon which processor performs the instruction, the mode the processor is in, and the speeds of the memories used for fetching the instruction, fetching its operands, and storing its results. Beyond this the time depends in many cases on the configuration of the operands and the number of iterative steps specified by the programmer as in a shift. Lastly the processor is designed to save time wherever possible by inspecting the operands in order to skip unnecessary steps.

The text sometimes refers to an instruction as being “executed.” To “execute” an instruction means that the processor performs the instruction out of the normal sequence, *ie* the sequence defined by the program counter (which sequence may not be consecutive, as when a skip or jump or some special circumstance changes PC). The processor fetches an executed instruction from a location whose address is supplied not by PC, but rather by an execute instruction (whose operand is itself interpreted as an instruction) or by some feature of the hardware such as a priority interrupt, trap, etc. It is assumed that control will shortly be returned to PC, at the location it originally specified before the interruption unless the instruction executed or the hardware feature itself changes PC.

Some simple examples are included with the instruction descriptions, but more complex examples using a variety of instructions are given in §2.11.

## 2.1 HALF WORD DATA TRANSMISSION

These instructions move a half word and may modify the contents of the other half of the destination location. There are sixteen instructions determined by which half of the source word is moved to which half of the destination, and by which of four possible operations is performed on the other

half of the destination. The basic mnemonics are three letters that indicate the transfer

HLL	Left half of source to left half of destination
HRL	Right half of source to left half of destination
HRR	Right half of source to right half of destination
HLR	Left half of source to right half of destination

plus a fourth, if necessary, to indicate the operation.

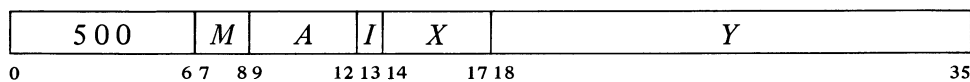
<i>Operation</i>	<i>Suffix</i>	<i>Effect on Other Half of Destination</i>
Do nothing		None
Zeros	Z	Places 0s in all bits of the other half
Ones	O	Places 1s in all bits of the other half
Extend	E	Places the sign (the leftmost bit) of the half word moved in all bits of the other half. This action extends a right half word number into a full word number but is valid arithmetically only for positive left half word numbers – the right extension of a number requires 0s regardless of sign (hence the Zeros operation should be used to extend a left half word number).

An additional letter may be appended to indicate the mode, which determines the source and destination of the half word moved.

<i>Mode</i>	<i>Suffix</i>	<i>Source</i>	<i>Destination</i>
Basic		<i>E</i>	AC
Immediate	I	The word 0, <i>E</i>	AC
Memory	M	AC	<i>E</i>
Self	S	<i>E</i>	<i>E</i> , but full word result also goes to AC if <i>A</i> is nonzero

Note that selecting the left half of the source in immediate mode merely clears the selected half of the destination.

### HLL Half Word Left to Left

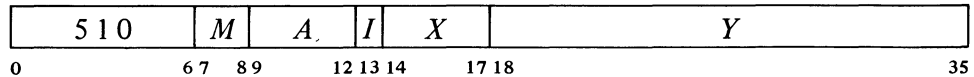


Move the left half of the source word specified by *M* to the left half of the specified destination. The source and the destination right half are unaffected; the original contents of the destination left half are lost.

HLLI merely clears AC left. If *A* is zero, HLLS is a no-op, otherwise it is equivalent to MOVE.

HLL	Half Left to Left	500
HLLI	Half Left to Left Immediate	501
HLLM	Half Left to Left Memory	502
HLLS	Half Left to Left Self	503

### HLLZ Half Word Left to Left, Zeros

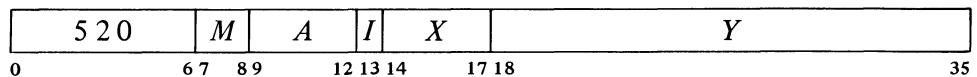


Move the left half of the source word specified by *M* to the left half of the specified destination, and clear the destination right half. The source is unaffected, the original contents of the destination are lost.

HLLZI merely clears AC. If *A* is zero, HLLZS merely clears the right half of location *E*.

HLLZ	Half Left to Left, Zeros	510
HLLZI	Half Left to Left, Zeros, Immediate	511
HLLZM	Half Left to Left, Zeros, Memory	512
HLLZS	Half Left to Left, Zeros, Self	513

### HLLO Half Word Left to Left, Ones

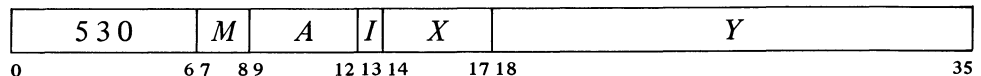


Move the left half of the source word specified by *M* to the left half of the specified destination, and set the destination right half to all 1s. The source is unaffected, the original contents of the destination are lost.

HLLOI sets AC to all 0s in the left half, all 1s in the right.

HLLO	Half Left to Left, Ones	520
HLLOI	Half Left to Left, Ones, Immediate	521
HLLLOM	Half Left to Left, Ones, Memory	522
HLLOS	Half Left to Left, Ones, Self	523

### HLLE Half Word Left to Left, Extend



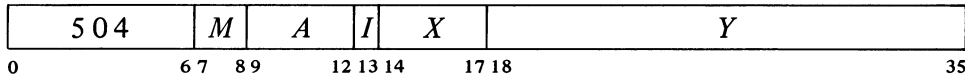
Move the left half of the source word specified by *M* to the left half of the specified destination, and make all bits in the destination right half equal to bit 0 of the source. The source is unaffected, the original contents of the destination are lost.



HLLE	Half Left to Left, Extend	530
HLLEI	Half Left to Left, Extend, Immediate	531
HLLEM	Half Left to Left, Extend, Memory	532
HLLES	Half Left to Left, Extend, Self	533

HLLEI is equivalent to HLLZI  
(it merely clears AC).

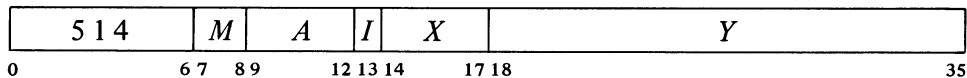
### HRL Half Word Right to Left



Move the right half of the source word specified by *M* to the left half of the specified destination. The source and the destination right half are unaffected; the original contents of the destination left half are lost.

HRL	Half Right to Left	504
HRLI	Half Right to Left Immediate	505
HRLM	Half Right to Left Memory	506
HRLS	Half Right to Left Self	507

### HRLZ Half Word Right to Left, Zeros

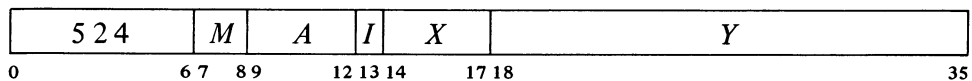


Move the right half of the source word specified by *M* to the left half of the specified destination, and clear the destination right half. The source is unaffected, the original contents of the destination are lost.

HRLZ	Half Right to Left, Zeros	514
HRLZI	Half Right to Left, Zeros, Immediate	515
HRLZM	Half Right to Left, Zeros, Memory	516
HRLZS	Half Right to Left, Zeros, Self	517

HRLZI loads the word *E,0*  
into AC.

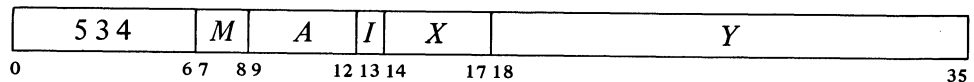
### HRLO Half Word Right to Left, Ones



Move the right half of the source word specified by *M* to the left half of the specified destination, and set the destination right half to all 1s. The source is unaffected, the original contents of the destination are lost.

HRLO	Half Right to Left, Ones	524
HRLOI	Half Right to Left, Ones, Immediate	525
HRLOM	Half Right to Left, Ones, Memory	526
HRLOS	Half Right to Left, Ones, Self	527

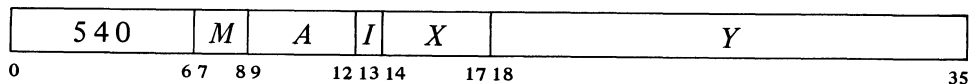
### HRLE Half Word Right to Left, Extend



Move the right half of the source word specified by *M* to the left half of the specified destination, and make all bits in the destination right half equal to bit 18 of the source. The source is unaffected, the original contents of the destination are lost.

HRLE	Half Right to Left, Extend	534
HRLEI	Half Right to Left, Extend, Immediate	535
HRLEM	Half Right to Left, Extend, Memory	536
HRLES	Half Right to Left, Extend, Self	537

### rRR Half Word Right to Right

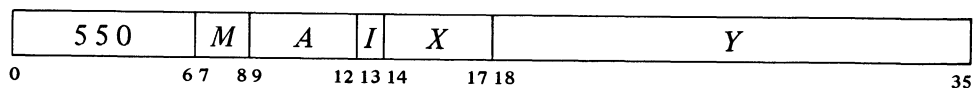


Move the right half of the source word specified by *M* to the right half of the specified destination. The source and the destination left half are unaffected; the original contents of the destination right half are lost.

HRR	Half Right to Right	540
HRRI	Half Right to Right Immediate	541
HRRM	Half Right to Right Memory	542
HRRS	Half Right to Right Self	543

If *A* is zero, HRRS is a no-op; otherwise it is equivalent to MOVE.

### HRRZ Half Word Right to Right, Zeros



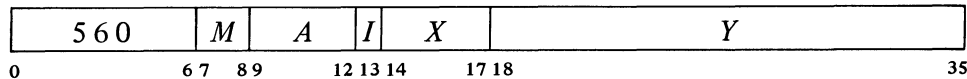
Move the right half of the source word specified by *M* to the right half of the

specified destination, and clear the destination left half. The source is unaffected, the original contents of the destination are lost.

HRRZ	Half Right to Right, Zeros	550
HRRZI	Half Right to Right, Zeros, Immediate	551
HRRZM	Half Right to Right, Zeros, Memory	552
HRRZS	Half Right to Right, Zeros, Self	553

HRRZI loads the word  $0, E$  into AC. If  $A$  is zero, HRRZS merely clears the left half of location  $E$ .

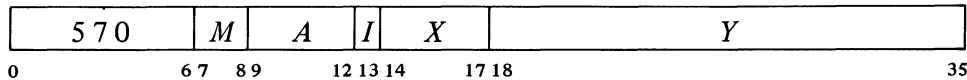
**HRRO Half Word Right to Right, Ones**



Move the right half of the source word specified by  $M$  to the right half of the specified destination, and set the destination left half to all 1s. The source is unaffected, the original contents of the destination are lost.

HRRO	Half Right to Right, Ones	560
HRROI	Half Right to Right, Ones, Immediate	561
HRROM	Half Right to Right, Ones, Memory	562
HRROS	Half Right to Right, Ones, Self	563

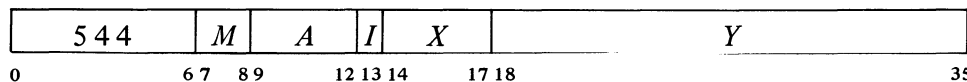
**HRRE Half Word Right to Right, Extend**



Move the right half of the source word specified by  $M$  to the right half of the specified destination, and make all bits in the destination left half equal to bit 18 of the source. The source is unaffected, the original contents of the destination are lost.

HRRE	Half Right to Right, Extend	570
HRREI	Half Right to Right, Extend, Immediate	571
HRREM	Half Right to Right, Extend, Memory	572
HRRES	Half Right to Right, Extend, Self	573

**HLR Half Word Left to Right**

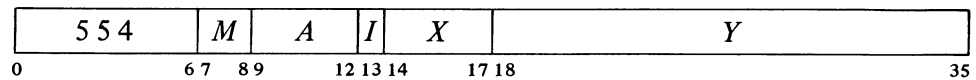


Move the left half of the source word specified by  $M$  to the right half of the

specified destination. The source and the destination left half are unaffected; the original contents of the destination right half are lost.

	HLR	Half Left to Right	544
HLRI merely clears AC right.	HLRI	Half Left to Right Immediate	545
	HLRM	Half Left to Right Memory	546
	HLRS	Half Left to Right Self	547

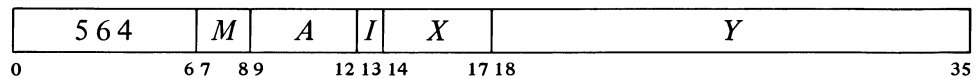
### HLRZ Half Word Left to Right, Zeros



Move the left half of the source word specified by *M* to the right half of the specified destination, and clear the destination left half. The source is unaffected, the original contents of the destination are lost.

	HLRZ	Half Left to Right, Zeros	554
HLRZI merely clears AC and is thus equivalent to HLLZI.	HLRZI	Half Left to Right, Zeros, Immediate	555
	HLRZM	Half Left to Right, Zeros, Memory	556
	HLRZS	Half Left to Right, Zeros, Self	557

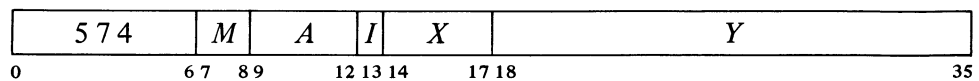
### HLRO Half Word Left to Right, Ones



Move the left half of the source word specified by *M* to the right half of the specified destination, and set the destination left half to all 1s. The source is unaffected, the original contents of the destination are lost.

	HLRO	Half Left to Right, Ones	564
HLROI sets AC to all 1s in the left half, all 0s in the right.	HLROI	Half Left to Right, Ones, Immediate	565
	HLROM	Half Left to Right, Ones, Memory	566
	HLROS	Half Left to Right, Ones, Self	567

### HLRE Half Word Left to Right, Extend



Move the left half of the source word specified by *M* to the right half of the

specified destination, and make all bits in the destination left half equal to bit 0 of the source. The source is unaffected, the original contents of the destination are lost.

HLRE	Half Left to Right, Extend	574
HLREI	Half Left to Right, Extend, Immediate	575
HLREM	Half Left to Right, Extend, Memory	576
HLRES	Half Left to Right, Extend, Self	577

HLREI is equivalent to HLRZI (it merely clears AC).

EXAMPLES. The half word transmission instructions are very useful for handling addresses, and they provide a convenient means of setting up an accumulator whose right half is to be used for indexing while a control count is kept in the left half. *Eg* this pair of instructions loads the 18-bit numbers *M* and *N* into the left and right halves respectively of an accumulator that is addressed symbolically as XR.

```
HRLZI XR,M
HRLRI XR,N
```

Of course the source program must somewhere define the value of the symbol XR as an octal number between 1 and 17.

Suppose that at some point we wish to use the two halves of XR independently as operands (taken as 18-bit positive numbers) for computations. We can begin by moving XR left to the right half of another accumulator AC and leaving the contents of XR right alone in XR.

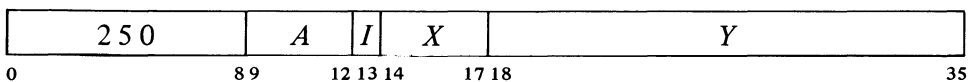
```
HLRZM XR,AC
HLLI XR, ;Clear XR left
```

It is not necessary to clear the other half of XR when loading the first half word. But any instruction that modifies the other half is faster than the corresponding instruction that does not, as the latter must fetch the destination word in order to save half of it. (The difference does not apply to self mode, for here the source and destination are the same.)

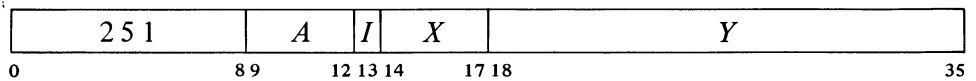
## 2.2 FULL WORD DATA TRANSMISSION

These are the instructions whose basic purpose is to move one or more full words of data from one place to another, usually from an accumulator to a memory location or vice versa. In a few cases instructions may perform minor arithmetic operations, such as forming the negative or the magnitude of the word being processed.

### EXCH Exchange



Move the contents of location *E* to AC and move AC to location *E*.

**BLT           Block Transfer**

For a reverse BLT procedure (highest addresses first), refer to the POP instruction on page 2-13.

Beginning at the location addressed by AC left, move words to another area of memory beginning at the location addressed by AC right. Continue until a word is moved to location *E*. The total number of words in the block is thus  $E - AC_R + 1$ . If  $AC_R \geq E$ , the BLT moves one word to location  $AC_L$ .

*CAUTION*

Priority interrupts are allowed during the execution of this instruction, following the processing of each word. If an interrupt occurs, the BLT stores the source and destination addresses for the next word in AC, so when the processor restarts upon the return to the interrupted program, it actually resumes at the correct point within the BLT. Therefore, unless the interrupt system is inactive, *A* and *X* must not address the same register as this would produce a different effective address calculation upon resumption should an interrupt occur; and the instruction must not attempt to load an accumulator addressed either by *A* or *X* unless it is the final location being loaded. Furthermore, the program cannot assume that AC is the same after the BLT as it was before.

**EXAMPLES.** This pair of instructions loads the accumulators from memory locations 2000–2017.

```
HRLZI 17,2000      ;Put 2000 000000 in AC 17
BLT   17,17
```

A convenient way to clear a block in memory is to clear the first location and then use a BLT to transfer the zero successively from one location to the next. Suppose the block starts at *A* and contains *B* words.

```
MOVE  AC,[A,,A+1]
CLEAR A
BLT   AC,A+B
```

Vis-a-vis the BLT, the source block runs from *A* to  $A+B-1$ , the destination block from  $A+1$  to  $A+B$ .

But to transfer the block in the opposite direction requires that one accumulator first be made available to the BLT:

```
MOVEM 17,2017      ;Move AC 17 to 2017 in memory
MOVEI 17,2000      ;Move the number 2000 to AC 17
BLT   17,2016
```

If at the time the accumulators were loaded the program had placed in location 2017 the control word necessary for storing them back in the same block (2000), the three instructions above could be replaced by

```
EXCH 17,2017
BLT  17,2016
```

**Move Instructions**

Besides the move instructions for single words there are also

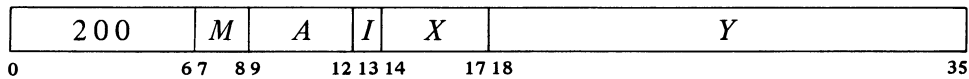
Each of these instructions moves a single word, which may be changed in the process (eg its two halves may be swapped). There are four instructions,

each with four modes that determine the source and destination of the word moved.

Mode	Suffix	Source	Destination
Basic		<i>E</i>	AC
Immediate	I	The word 0, <i>E</i>	AC
Memory	M	AC	<i>E</i>
Self	S	<i>E</i>	<i>E</i> , but also AC if <i>A</i> is nonzero

four transmission instructions that handle double length operands (operands of two adjacent words). These are available, however, only in the KI10; and since they are principally for use in hardware double precision floating point operations, they are described with the floating point instructions in §2.6

**MOVE Move**

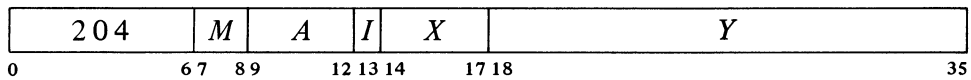


Move one word from the source to the destination specified by *M*. The source is unaffected, the original contents of the destination are lost.

MOVE	Move	200
MOVEI	Move Immediate	201
MOVEM	Move to Memory	202
MOVES	Move to Self	203

MOVEI loads the word 0, *E* into AC and is thus equivalent to HRRZI. If *A* is zero, MOVES is a no-op; otherwise it is equivalent to MOVE.

**MOVS Move Swapped**

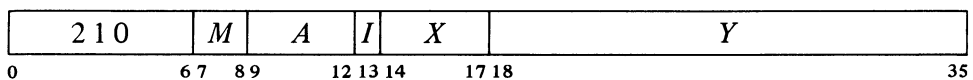


Interchange the left and right halves of the word from the source specified by *M* and move it to the specified destination. The source is unaffected, the original contents of the destination are lost.

MOVS	Move Swapped	204
MOVSI	Move Swapped Immediate	205
MOVSM	Move Swapped to Memory	206
MOVSS	Move Swapped to Self	207

Swapping halves in immediate mode loads the word *E*, 0 into AC. MOVSI is thus equivalent to HRLZI.

**MOVN Move Negative**



Negate the word from the source specified by *M* and move it to the specified destination. If the source word is fixed point  $-2^{35}$  (400000 000000) set the

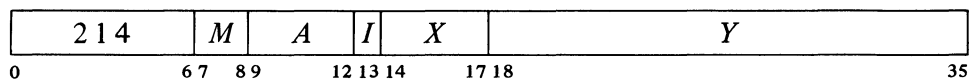
In the KI10 a move executed as an interrupt instruction can set no flags.

MOVNI loads AC with the negative of the word 0, E and can set no flags.

Overflow and Carry 1 flags. (Negating the equivalent floating point  $-1 \times 2^{127}$  sets the flags, but this is not a normalized number.) If the source word is zero, set Carry 0 and Carry 1. The source is unaffected, the original contents of the destination are lost. Setting Overflow also sets the Trap 1 flag in the KI10.

MOVN	Move Negative	210
MOVNI	Move Negative Immediate	211
MOVNM	Move Negative to Memory	212
MOVNS	Move Negative to Self	213

### MOVMM Move Magnitude



In the KI10 a move executed as an interrupt instruction can set no flags.

The word 0, E is equivalent to its magnitude, so MOVMI is equivalent to MOVEI.

Take the magnitude of the word contained in the source specified by *M* and move it to the specified destination. If the source word is fixed point  $-2^{35}$  (400000 000000) set the Overflow and Carry 1 flags. (Negating the equivalent floating point  $-1 \times 2^{127}$  sets the flags, but this is not a normalized number.) The source is unaffected, the original contents of the destination are lost. Setting Overflow also sets the Trap 1 flag in the KI10.

MOVMM	Move Magnitude	214
MOVMI	Move Magnitude Immediate	215
MOVMM	Move Magnitude to Memory	216
MOVMS	Move Magnitude to Self	217

An example at the end of the preceding section demonstrates the use of a pair of immediate-mode half word transfers to load an address and a control count into an accumulator. The same result can be attained by a single move instruction. This saves time but still requires two locations. *Eg* if the number 200 001400 is stored in location M, the instruction

MOVE AC, M

loads 200 into AC left and 1400 into AC right. If the same word, or its negative, or with its halves swapped, must be loaded on several occasions, then both time and space can be saved as each transfer requires only a single move instruction that references M.

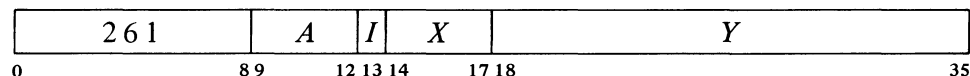
### Pushdown List

These two instructions insert and remove full words in a pushdown list. The address of the top item in the list is kept in the right half of a pointer in AC,



and the program can keep a control count in the left half. There are also two subroutine-calling instructions that utilize a pushdown list of jump addresses [ §2.9].

**PUSH      Push Down**

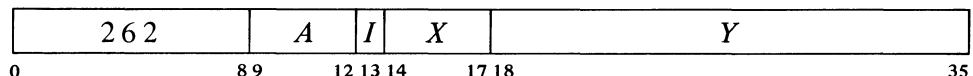


Add one to each half of AC, then move the contents of location *E* to the location now addressed by AC right. If the addition causes the count in AC left to reach zero, set the Pushdown Overflow flag in the KA10, set the Trap 2 flag in the KI10. The contents of *E* are unaffected, the original contents of the location added to the list are lost.

Note: The KA10 increments the two halves of AC by adding  $1\ 000001_8$  to the entire register. In the KI10 the two halves are handled independently.

In the KI10 a PUSH executed as an interrupt instruction cannot set Trap 2.

**POP      Pop Up**



Move the contents of the location addressed by AC right to location *E*, then subtract one from each half of AC. If the subtraction causes the count in AC left to reach  $-1$ , set the Pushdown Overflow flag in the KA10, set the Trap 2 flag in the KI10. The original contents of *E* are lost.

Because of the order in which the operands are stored, the instruction POP AC,AC would load the contents of the location addressed by AC right into AC on top of the pushdown count, destroying it.

Note: The KA10 decrements the two halves of AC by subtracting  $1\ 000001_8$  from the entire register. In the KI10 the two halves are handled independently.

In the KI10 a POP executed as an interrupt instruction cannot set Trap 2.

A POP can be used to implement a reverse BLT, *ie* to transfer a block of words from one area of memory to another, starting at the largest addresses and proceeding to the smallest. To move a block of *N* words from a source area to a destination area whose maximum addresses are *S* and *D* respectively, the program must first set up a pushdown pointer in accumulator T, where T left contains  $N - 1 + 400000$  and T right contains *S*. The transfer is then effected by this pair of instructions

```
POP      T, D-S(T)
JUMPL   T, -1
```

In the KA10, incrementing and decrementing both halves of AC together is effected by adding and subtracting  $1\ 000001_8$ . Hence a count of  $-2$  in AC left is increased to zero if  $2^{18} - 1$  is incremented in AC right, and conversely, 1 in AC left is decreased to  $-1$  if zero is decremented in AC right.

A pushdown list is simply a set of consecutive memory locations from which words are read in the order opposite that in which they are written. In more general terms, it is any list in which the only item that can be removed at any given time is the last item in the list. This is usually referred to as "first in, last out" or "last in, first out". Suppose locations *a, b, c, ...* are set aside for a pushdown list. We can deposit data in *a, b, c, d*, then read

where the jump returns to the POP until T left is less than 400000, *ie* until it looks positive. The 400000 added into T left prevents pushdown overflow, but also limits the block to  $2^{17}$  words.

*d*, then write in *d* and *e*, then read *e*, *d*, *c*, etc.

Note that by trapping or checking overflow and keeping a control count in AC left, the programmer can set a limit to the size of the list by starting the count negative, or he can prevent the program from extracting more words than there are in the list by starting the count at zero, but he cannot do both at once. The common practice is to limit the size of the list.

Pushdown storage is very convenient for a program that can use data stored in this manner as the pointer is initialized only once and only one accumulator is required for the most complex pushdown operations. To initialize a pointer P for a list to be kept in a block of memory beginning at BLIST and to contain at most *N* items, the following suffices.

```
MOVSI  P,-N
HRRI   P,BLIST-1
```

Of course the programmer must define BLIST elsewhere and set aside locations BLIST to BLIST + *N* - 1. Using MACRO to full advantage one could instead give

```
MOVE   P,[IOWD N,BLIST]
```

where the pseudoinstruction

```
IOWD J,K
```

is replaced by a word containing  $-J$  in the left half and  $K - 1$  in the right. Elsewhere there would appear

```
BLIST:  BLOCK  N
```

which defines BLIST as the current contents of the location counter and sets aside the *N* locations beginning at that point.

In the PDP-10 the pushdown list is kept in a random access core memory, so the restrictions on order of entry and removal of items actually apply only to the standard addressing by the pointer in pushdown instructions — other addressing methods can reference any item at any time. The most convenient way to do this is to use the right half of the pointer as an index register. To move the last entry to accumulator AC we need simply give

```
MOVE   AC,(P)
```

Of course this does not shorten the list — the word moved remains the last item in it.

One usually regards an index register as supplying an additive factor for a basic address contained in an instruction word, but the index register can supply the basic address and the instruction the additive factor. Thus we can retrieve the next to last item by giving

```
MOVE   AC,-1(P)
```

and so forth. Similarly

```
PUSH   P,-3(P)
```

adds the third to last item to the end of the list;

POP P, -2(P)

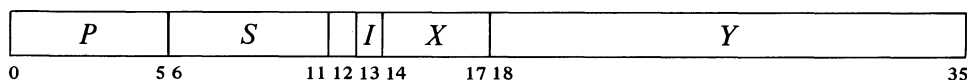
removes the last item and inserts it in place of the next to last item in the shortened list.

Note that  $E$  is calculated before the contents of  $P$  are changed.

### 2.3 BYTE MANIPULATION

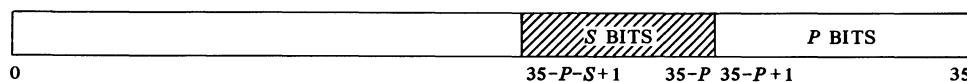
This set of five instructions allows the programmer to pack or unpack bytes of any length anywhere within a word. Movement of a byte is always between AC and a memory location: a deposit instruction takes a byte from the right end of AC and inserts it at any desired position in the memory location; a load instruction takes a byte from any position in the memory location and places it right-justified in AC.

The byte manipulation instructions have the standard memory reference format, but the effective address  $E$  is used to retrieve a pointer, which is used in turn to locate the byte or the place that will receive it. The pointer has the format



Bit 12 is reserved for future use and should be 0.

where  $S$  is the size of the byte as a number of bits, and  $P$  is its position as the number of bits remaining at the right of the byte in the word (eg if  $P$  is 3 the rightmost bit of the byte is bit 32 of the word). The rest of the pointer is interpreted in the same way as in an instruction:  $I$ ,  $X$  and  $Y$  are used to calculate the address of the location that is the source or destination of the byte. Thus the pointer aims at a word whose format is



where the shaded area is the byte.

To facilitate processing a series of bytes, several of the byte instructions increment the pointer, ie modify it so that it points to the next byte position in a set of memory locations. Bytes are processed from left to right in a word, so incrementing merely replaces the current value of  $P$  by  $P - S$ , unless there is insufficient space in the present location for another byte of the specified size ( $P - S < 0$ ). In this case  $Y$  is increased by one to point to the next consecutive location, and  $P$  is set to  $36 - S$  to point to the first byte at the left in the new location.

#### CAUTION (KA10 ONLY)

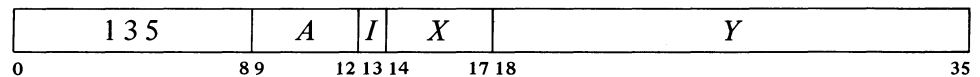
Do not allow  $Y$  to reach maximum value. The whole pointer is incre-

In the KI10, incrementing maximum  $Y$  produces a zero address without affecting  $X$ .

mented, so if  $Y$  is  $2^{18} - 1$  it becomes zero and  $X$  is also incremented. The address calculation for the pointer uses the original  $X$ , but if a priority interrupt should occur before the calculation is complete, the incremented  $X$  is used when the instruction is repeated.

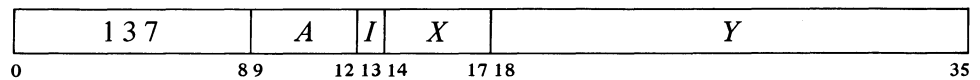
Among these five instructions one simply increments the pointer, the others load or deposit a byte with or without incrementing.

### LDB Load Byte



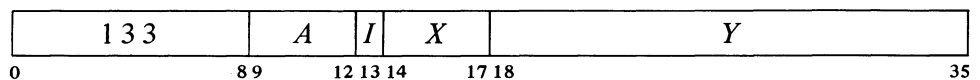
Retrieve a byte of  $S$  bits from the location and position specified by the pointer contained in location  $E$ , load it into the right end of AC, and clear the remaining AC bits. The location containing the byte is unaffected, the original contents of AC are lost.

### DPB Deposit Byte



Deposit the right  $S$  bits of AC into the location and position specified by the pointer contained in location  $E$ . The original contents of the bits that receive the byte are lost, AC and the remaining bits of the deposit location are unaffected.

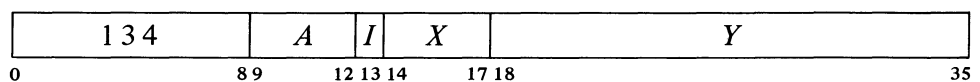
### IBP Increment Byte Pointer



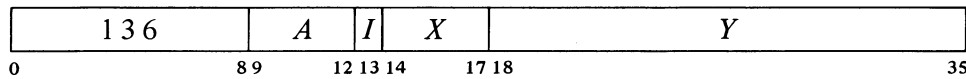
The  $A$  portion of this instruction is reserved for future use and should be zero (at present it is ignored).

Increment the byte pointer in location  $E$  as explained above.

### ILDB Increment Pointer and Load Byte



Increment the byte pointer in location  $E$  as explained above. Then retrieve a byte of  $S$  bits from the location and position specified by the newly incremented pointer, load it into the right end of AC, and clear the remaining AC bits. The location containing the byte is unaffected, the original contents of AC are lost.

**IDPB          Increment Pointer and Deposit Byte**

Increment the byte pointer in location  $E$  as explained above. Then deposit the right  $S$  bits of AC into the location and position specified by the newly incremented pointer. The original contents of the bits that receive the byte are lost, AC and the remaining bits of the deposit location are unaffected.

Note that in the pair of instructions that both increment the pointer and process a byte, it is the *modified* pointer that determines the byte location and position. Hence to unpack bytes from a block of memory, the program should set up the pointer to point to a byte just *before* the first desired, and then load them with a loop containing an ILDB. If the first byte is at the left end of a word, this is most easily done by initializing the pointer with a  $P$  of 36 ( $44_8$ ). Incrementing then replaces the 36 with  $36 - S$  to point to the first byte. At any time that the program might inspect the pointer during execution of a series of ILDBs or IDPBs, it points to the last byte processed (this may not be true when the pointer is tested from an interrupt routine [§2.13]).

**Special Considerations.** If  $S$  is greater than  $P$  and also greater than 36, incrementing produces a new  $P$  equal to  $100 - S$  rather than  $36 - S$ . For  $S > 36$  the byte is at most the entire word; for  $P \geq 36$  no byte is processed (loading merely clears AC). If both  $P$  and  $S$  are less than 36 but  $P + S > 36$ , a byte of size  $36 - P$  is loaded from position  $P$ , or the right  $36 - P$  bits of the byte are deposited in position  $P$ .

## 2.4 LOGIC

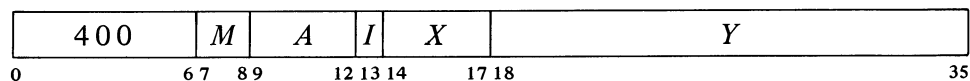
For logical operations the PDP-10 has instructions for shifting and rotating as well as for performing the complete set of sixteen Boolean functions of two variables (including those in which the result depends on only one or neither variable). The Boolean functions operate bitwise on full words, so each instruction actually performs thirty-six logical operations simultaneously. Thus in the AND function of two words, each bit of the result is the AND of the corresponding bits of the operands. The table on page 2-23 lists the bit configurations that result from the various operand configurations for all instructions.

Each Boolean instruction has four modes that determine the source of the non-AC operand, if any, and the destination of the result. For an instruction without an operand (one that merely clears a location or sets it to all 1s) the modes differ only in the destination of the result, so basic and immediate

modes are equivalent. The same is true also of an instruction that uses only an AC operand. When specified by the mode, the result goes to the accumulator addressed by *A*, even when there is no AC operand.

<i>Mode</i>	<i>Suffix</i>	<i>Source of non-AC operand</i>	<i>Destination of result</i>
Basic		<i>E</i>	AC
Immediate	I	The word 0, <i>E</i>	AC
Memory	M	<i>E</i>	<i>E</i>
Both	B	<i>E</i>	AC and <i>E</i>

### SETZ Set to Zeros



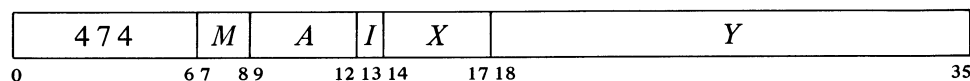
SETZ and SETZI are equivalent (both merely clear AC). In them, *I*, *X* and *Y* are reserved for future use and should be zero (at present *E* is ignored).

MACRO also recognizes CLEAR, CLEARI, CLEARM and CLEARB as equivalent to the set-to-zeros mnemonics.

Change the contents of the destination specified by *M* to all 0s.

SETZ	Set to Zeros	400
SETZI	Set to Zeros Immediate	401
SETZM	Set to Zeros Memory	402
SETZB	Set to Zeros Both	403

### SETO Set to Ones

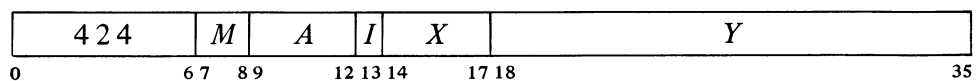


SETO and SETOI are equivalent. In them, *I*, *X* and *Y* are reserved for future use and should be zero (at present *E* is ignored).

Change the contents of the destination specified by *M* to all 1s.

SETO	Set to Ones	474
SETOI	Set to Ones Immediate	475
SETOM	Set to Ones Memory	476
SETOB	Set to Ones Both	477

### SETA Set to AC



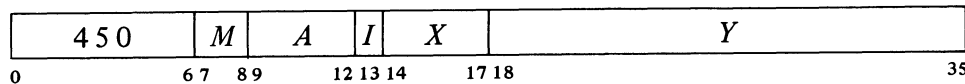
Make the contents of the destination specified by *M* equal to AC.

SETA	Set to AC	424
SETAI	Set to AC Immediate	425
SETAM	Set to AC Memory	426
SETAB	Set to AC Both	427

SETA and SETAI are no-ops. In them, *I*, *X* and *Y* are reserved for future use and should be zero (at present *E* is ignored).

SETAM and SETAB are both equivalent to MOVEM (all move AC to location *E*).

**SETCA Set to Complement of AC**

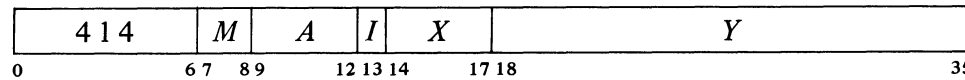


Change the contents of the destination specified by *M* to the complement of AC.

SETCA	Set to Complement of AC	450
SETCAI	Set to Complement of AC Immediate	451
SETCAM	Set to Complement of AC Memory	452
SETCAB	Set to Complement of AC Both	453

SETCA and SETCAI are equivalent (both complement AC). In them, *I*, *X* and *Y* are reserved for future use and should be zero (at present *E* is ignored).

**SETM Set to Memory**

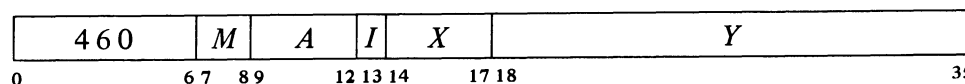


Make the contents of the destination specified by *M* equal to the specified operand.

SETM	Set to Memory	414
SETMI	Set to Memory Immediate	415
SETMM	Set to Memory Memory	416
SETMB	Set to Memory Both	417

SETM and SETMB are equivalent to MOVE. SETMI moves the word 0,*E* to AC and is thus equivalent to MOVEI. SETMM is a no-op that references memory.

**SETCM Set to Complement of Memory**

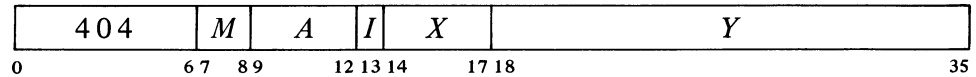


Change the contents of the destination specified by *M* to the complement of the specified operand.

SETCMI moves the complement of the word  $O, E$  to AC.  
SETCMM complements location  $E$ .

SETCM	Set to Complement of Memory	460
SETCMI	Set to Complement of Memory Immediate	461
SETCMM	Set to Complement of Memory Memory	462
SETCMB	Set to Complement of Memory Both	463

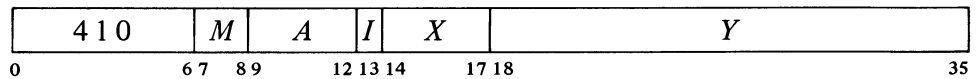
### AND And with AC



Change the contents of the destination specified by  $M$  to the AND function of the specified operand and AC.

AND	And	404
ANDI	And Immediate	405
ANDM	And to Memory	406
ANDB	And to Both	407

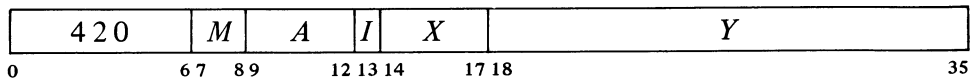
### ANDCA And with Complement of AC



Change the contents of the destination specified by  $M$  to the AND function of the specified operand and the complement of AC.

ANDCA	And with Complement of AC	410
ANDCAI	And with Complement of AC Immediate	411
ANDCAM	And with Complement of AC to Memory	412
ANDCAB	And with Complement of AC to Both	413

### ANDCM And Complement of Memory with AC



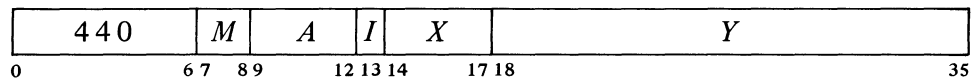
Change the contents of the destination specified by  $M$  to the AND function of the complement of the specified operand and AC.

ANDCM	And Complement of Memory	420
ANDCMI	And Complement of Memory Immediate	421



ANDCMM	And Complement of Memory to Memory	422
ANDCMB	And Complement of Memory to Both	423

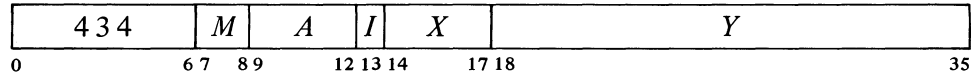
### ANDCB      And Complements of Both



Change the contents of the destination specified by *M* to the AND function of the complements of both the specified operand and AC. The result is the NOR function of the operands.

ANDCB	And Complements of Both	440
ANDCBI	And Complements of Both Immediate	441
ANDCBM	And Complements of Both to Memory	442
ANDCBB	And Complements of Both to Both	443

### IOR      Inclusive Or with AC

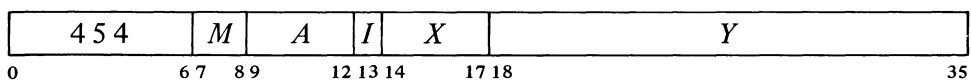


Change the contents of the destination specified by *M* to the inclusive OR function of the specified operand and AC.

IOR	Inclusive Or	434
IORI	Inclusive Or Immediate	435
IORM	Inclusive Or to Memory	436
IORB	Inclusive Or to Both	437

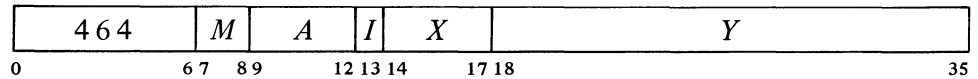
MACRO also recognizes OR, ORI, ORM and ORB as equivalent to the inclusive OR mnemonics.

### ORCA      Inclusive Or with Complement of AC



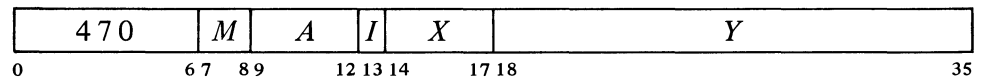
Change the contents of the destination specified by *M* to the inclusive OR function of the specified operand and the complement of AC.

ORCA	Or with Complement of AC	454
ORCAI	Or with Complement of AC Immediate	455
ORCAM	Or with Complement of AC to Memory	456
ORCAB	Or with Complement of AC to Both	457

**ORCM Inclusive Or Complement of Memory with AC**

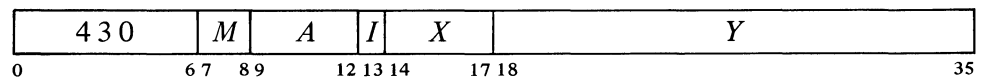
Change the contents of the destination specified by *M* to the inclusive or function of the complement of the specified operand and AC.

<b>ORCM</b>	Or Complement of Memory	464
<b>ORCMI</b>	Or Complement of Memory Immediate	465
<b>ORCMM</b>	Or Complement of Memory to Memory	466
<b>ORCMB</b>	Or Complement of Memory to Both	467

**ORCB Inclusive Or Complements of Both**

Change the contents of the destination specified by *M* to the inclusive or function of the complements of both the specified operand and AC. The result is the NAND function of the operands.

<b>ORCB</b>	Or Complements of Both	470
<b>ORCBI</b>	Or Complements of Both Immediate	471
<b>ORCBM</b>	Or Complements of Both to Memory	472
<b>ORCBB</b>	Or Complements of Both to Both	473

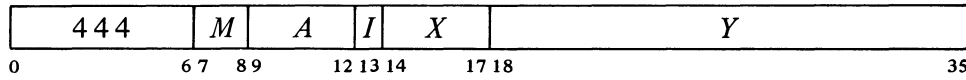
**XOR Exclusive Or with AC**

Change the contents of the destination specified by *M* to the exclusive or function of the specified operand and AC.

<b>XOR</b>	Exclusive Or	430
<b>XORI</b>	Exclusive Or Immediate	431
<b>XORM</b>	Exclusive Or to Memory	432
<b>XORB</b>	Exclusive Or to Both	433

The original contents of the destination can be recovered except in **XORB**, where both operands are replaced by the result. In the other three modes the replaced operand is restored by repeating the instruction in the same mode, *ie* by taking the exclusive or of the remaining operand and the result.

**EQV            Equivalence with AC**



Change the contents of the destination specified by *M* to the complement of the exclusive OR function of the specified operand and AC (the result has 1s wherever the corresponding bits of the operands are the same).

EQV	Equivalence	444
EQVI	Equivalence Immediate	445
EQVM	Equivalence to Memory	446
EQVB	Equivalence to Both	447

The original contents of the destination can be recovered except in EQVB, where both operands are replaced by the result. In the other three modes the replaced operand is restored by repeating the instruction in the same mode, *ie* by taking the equivalence function of the remaining operand and the result.

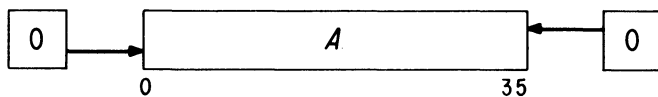
For the four possible bit configurations of the two operands, the above sixteen instructions produce the following results. In each case the result as listed is equal to bits 3–6 of the instruction word.

	<i>AC</i>	0	1	0	1
<i>Mode Specified Operand</i>	0	0	1	1	1
SETZ	0	0	0	0	0
AND	0	0	0	0	1
ANDCA	0	0	1	0	0
SETM	0	0	1	1	1
ANDCM	0	1	0	0	0
SETA	0	1	0	1	1
XOR	0	1	1	1	0
IOR	0	1	1	1	1
ANDCB	1	0	0	0	0
EQV	1	0	0	0	1
SETCA	1	0	1	1	0
ORCA	1	0	1	1	1
SETCM	1	1	0	0	0
ORCM	1	1	0	1	1
ORCB	1	1	1	1	0
SETO	1	1	1	1	1

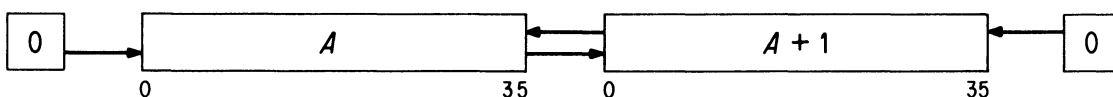
**Shift and Rotate**

The remaining logical instructions shift or rotate right or left the contents of AC or the contents of two accumulators,  $A$  and  $A+1$  (mod  $20_8$ ), concatenated into a 72-bit register with  $A$  on the left. The illustration below shows the movement of information these instructions produce in the accu-

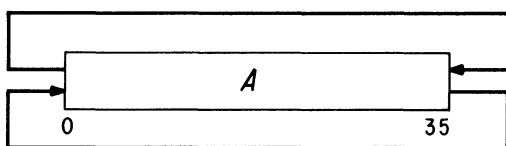
LSH



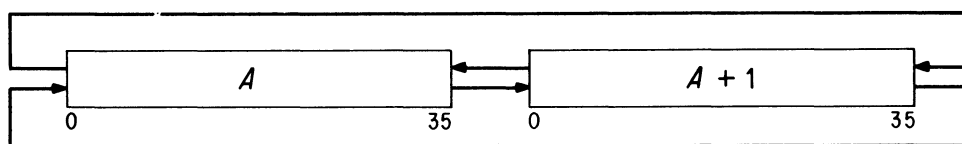
LSHC



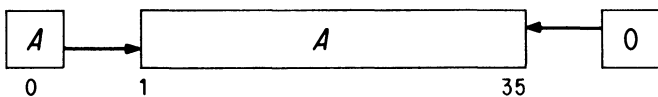
ROT



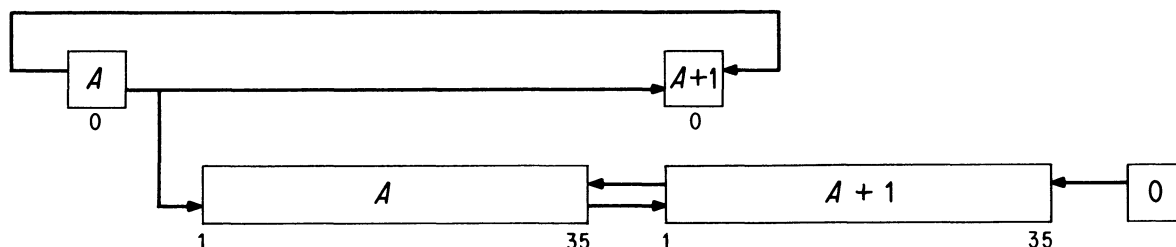
ROTC



ASH



ASHC

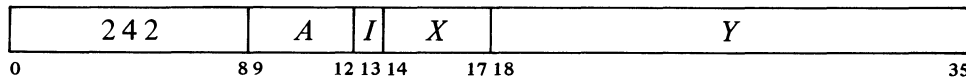


ACCUMULATOR BIT FLOW IN SHIFT AND ROTATE INSTRUCTIONS

mulators. In a (logical) shift the contents of a register are moved bit-to-bit with 0s brought in at the end being vacated; information shifted out at the other end is lost. [For a discussion of arithmetic shifting see §2.5.] In rotation the contents are moved cyclically such that information rotated out at one end is put in at the other.

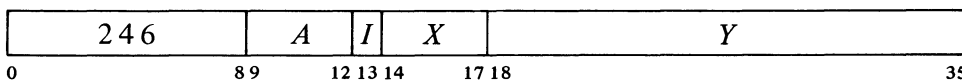
The number of places moved is specified by the result of the effective address calculation taken as a signed number (in twos complement notation) modulo  $2^8$  in magnitude. In other words the effective shift  $E$  is the number composed of bit 18 (which is the sign) and bits 28–35 of the calculation result. Hence the programmer may specify the shift directly in the instruction (perhaps indexed) or give an indirect address to be used in calculating the shift. A positive  $E$  produces motion to the left, a negative  $E$  to the right. In the KA10, maximum movement is 255 places. The KI10 eliminates redundant movement by logical shifting at most 72 places regardless of the value of  $E$ , and rotating  $E \bmod 72$  places (except 72 places if  $E$  is a nonzero multiple of 72).

**LSH Logical Shift**



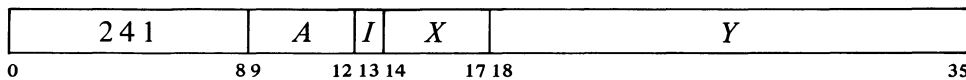
Shift AC the number of places specified by  $E$ . If  $E$  is positive, shift left bringing 0s into bit 35; data shifted out of bit 0 is lost. If  $E$  is negative, shift right bringing 0s into bit 0; data shifted out of bit 35 is lost.

**LSHC Logical Shift Combined**

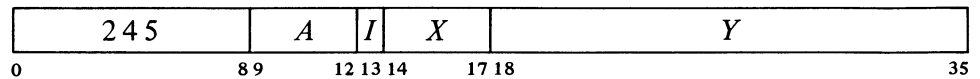


Concatenate accumulators  $A$  and  $A+1$  with  $A$  on the left, and shift the 72-bit combination the number of places specified by  $E$ . If  $E$  is positive, shift left bringing 0s into bit 71 (bit 35 of AC  $A+1$ ); bit 36 is shifted into bit 35; data shifted out of bit 0 is lost. If  $E$  is negative, shift right bringing 0s into bit 0; bit 35 is shifted into bit 36; data shifted out of bit 71 is lost.

**ROT Rotate**



Rotate AC the number of places specified by  $E$ . If  $E$  is positive, rotate left; bit 0 is rotated into bit 35. If  $E$  is negative, rotate right; bit 35 is rotated into bit 0.

**ROTC Rotate Combined**

Concatenate accumulators  $A$  and  $A+1$  with  $A$  on the left, and rotate the 72-bit combination the number of places specified by  $E$ . If  $E$  is positive, rotate left; bit 0 is rotated into bit 71 (bit 35 of AC  $A+1$ ) and bit 36 into bit 35. If  $E$  is negative, rotate right; bit 35 is rotated into bit 36 and bit 71 into bit 0.

## 2.5 FIXED POINT ARITHMETIC

For fixed point arithmetic the PDP-10 has instructions for arithmetic shifting (which is essentially multiplication by a power of 2) as well as for performing addition, subtraction, multiplication and division of numbers in fixed point format [§ 1.1]. In such numbers the position of the binary point is arbitrary (the programmer may adopt any point convention). The add and subtract instructions involve only single length numbers, whereas multiply supplies a double length product, and divide uses a double length dividend. The high and low order words respectively of a double length fixed point number are in accumulators  $A$  and  $A+1$  (mod  $20_8$ ), where the magnitude is the 70-bit string in bits 1-35 of the two words and the signs of the two are identical. There are also integer multiply and divide instructions that involve only single length numbers and are especially suited for handling smaller integers, particularly those of eighteen bits or less such as addresses (of course they can be used for small fractions as well provided the programmer keeps track of the binary point). For convenience in the following, all operands are assumed to be integers (binary point at the right).

The processor has four flags, Overflow, Carry 0, Carry 1 and No Divide, that indicate when the magnitude of a number is or would be larger than can be accommodated. Carry 0 and Carry 1 actually detect carries out of bits 0 and 1 in certain instructions that employ fixed point arithmetic operations: the add and subtract instructions treated here, the move instructions that produce the negative or magnitude of the word moved [§ 2.2], and the arithmetic test instructions that increment or decrement the test word [§ 2.7]. In these instructions an incorrect result is indicated — and the Overflow flag set — if the carries are different, *ie* if there is a carry into the sign but not out of it, or vice versa. The Overflow flag is also set by No Divide being set, which means the processor has failed to perform a division because the magnitude of the dividend is greater than or equal to that of the divisor, or in integer divide, simply that the divisor is zero. In other overflow cases only Overflow itself is set: these include too large a product in multiplication, too large a number to convert to fixed point [§ 2.6], and loss of significant bits in left arithmetic shifting. In the KI10 any condition that sets Overflow also sets the Trap 1 flag.

These flags can be read and controlled by certain program control instructions [§§ 2.9, 2.10]. KI10 overflow is handled by trapping through the

Overflow is determined directly from the carries, not from the carry flags, as their states may reflect events in previous instructions.

In the KI10 an arithmetic instruction executed as an interrupt instruction can set no flags.

setting of Trap 1 [§2.9], but in the KA10, the program must make direct use of the Overflow flag, which is available as a processor condition (via an in-out instruction) that can request a priority interrupt if enabled [§2.14]. The conditions detected can only set the arithmetic flags and the hardware does not clear them, so the program must clear them before an instruction if they are to give meaningful information about the instruction afterward. However, the program can check the flags following a series of instructions to determine whether the entire series was free of the types of error detected.

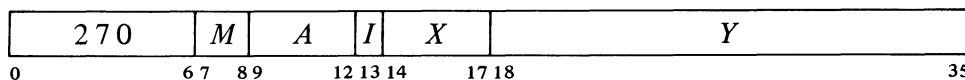
All but the shift instructions have four modes that determine the source of the non-AC operand and the destination of the result.

User overflow is handled by the Monitor according to instructions from the user. Refer to Chapter 3 of *DECsystem-10 Monitor Calls*.

Besides indicating error types, the carry flags facilitate performing multiple precision arithmetic.

<i>Mode</i>	<i>Suffix</i>	<i>Source of non-AC operand</i>	<i>Destination of result</i>
Basic		<i>E</i>	AC
Immediate	I	The word 0, <i>E</i>	AC
Memory	M	<i>E</i>	<i>E</i>
Both	B	<i>E</i>	AC and <i>E</i>

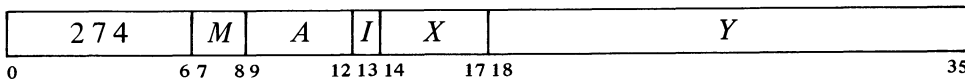
**ADD      Add**



Add the operand specified by *M* to AC and place the result in the specified destination. If the sum is  $\geq 2^{35}$  set Overflow and Carry 1; the result stored has a minus sign but a magnitude in positive form equal to the sum less  $2^{35}$ . If the sum is  $< -2^{35}$  set Overflow and Carry 0; the result stored has a plus sign but a magnitude in negative form equal to the sum plus  $2^{35}$ . Set both carry flags if both summands are negative, or their signs differ and their magnitudes are equal or the positive one is the greater in magnitude.

ADD	Add	270
ADDI	Add Immediate	271
ADDM	Add to Memory	272
ADDB	Add to Both	273

**SUB      Subtract**

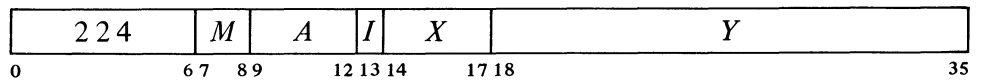


Subtract the operand specified by *M* from AC and place the result in the specified destination. If the difference is  $\geq 2^{35}$  set Overflow and Carry 1; the result stored has a minus sign but a magnitude in positive form equal to the difference less  $2^{35}$ . If the difference is  $< -2^{35}$  set Overflow and Carry 0; the result stored has a plus sign but a magnitude in negative form equal to

the difference plus  $2^{35}$ . Set both carry flags if the signs of the operands are the same and AC is the greater or the two are equal, or the signs of the operands differ and AC is negative.

SUB	Subtract	274
SUBI	Subtract Immediate	275
SUBM	Subtract to Memory	276
SUBB	Subtract to Both	277

**MUL Multiply**



Multiply AC by the operand specified by *M*, and place the high order word of the double length result in the specified destination. If *M* specifies AC as a destination, place the low order word in accumulator *A*+1. If both operands are  $-2^{35}$  set Overflow; the double length result stored is  $-2^{70}$ .

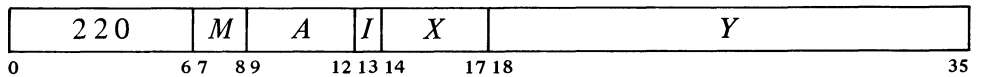
MUL	Multiply	224
MULI	Multiply Immediate	225
MULM	Multiply to Memory	226
MULB	Multiply to Both	227

▲ Remember that bit 0 of the low order word is equal to the sign of the product.

*CAUTION*

In the KA10, an AC operand of  $-2^{35}$  is treated as though it were  $+2^{35}$ , producing the incorrect sign in the product.

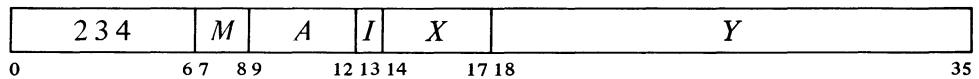
**IMUL Integer Multiply**



Multiply AC by the operand specified by *M*, and place the sign and the 35 low order magnitude bits of the product in the specified destination. Set Overflow if the product is  $\geq 2^{35}$  or  $< -2^{35}$  (*ie* if the high order word of the double length product is not null); the high order word is lost.

IMUL	Integer Multiply	220
IMULI	Integer Multiply Immediate	221
IMULM	Integer Multiply to Memory	222
IMULB	Integer Multiply to Both	223

**DIV Divide**



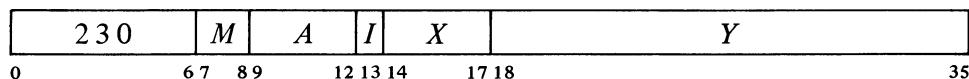
▲ If the high order word of the magnitude of the double length number in



accumulators  $A$  and  $A+1$  is greater than or equal to the magnitude of the operand specified by  $M$ , set Overflow and No Divide, and go immediately to the next instruction without affecting the original AC or memory operand in any way. Otherwise divide the double length number contained in accumulators  $A$  and  $A+1$  by the specified operand, calculating a quotient of 35 magnitude bits including leading zeros. Place the unrounded quotient in the specified destination. If  $M$  specifies AC as a destination, place the remainder, with the same sign as the dividend, in accumulator  $A+1$ .

<b>DIV</b>	Divide	234
<b>DIVI</b>	Divide Immediate	235
<b>DIVM</b>	Divide to Memory	236
<b>DIVB</b>	Divide to Both	237

**IDIV Integer Divide**



If the operand specified by  $M$  is zero, or AC contains  $-2^{35}$  and the operand specified by  $M$  is  $\pm 1$ , set Overflow and No Divide, and go immediately to the next instruction without affecting the original AC or memory operand in any way. Otherwise divide AC by the specified operand, calculating a quotient of 35 magnitude bits including leading zeros. Place the unrounded quotient in the specified destination. If  $M$  specifies AC as the destination, place the remainder, with the same sign as the dividend, in accumulator  $A+1$ .

<b>IDIV</b>	Integer Divide	230
<b>IDIVI</b>	Integer Divide Immediate	231
<b>IDIVM</b>	Integer Divide to Memory	232
<b>IDIVB</b>	Integer Divide to Both	233

**EXAMPLES.** The integer multiply and divide instructions are very useful for computations on addresses or character codes, or performing any integral operations in which the result is small enough to be accommodated in a single register.

Suppose we wish to reverse the order of the digits in the 6-bit character *abcdef*, where the letters represent the bits of the character. We first duplicate it three times to the left and shift the result left one place producing

$$a\ bcd\ efa\ bcd\ efa\ bcd\ efa\ bcd\ ef0$$

where the bits are grouped corresponding to the octal digits in the word. Anding this with

$$1\ 000\ 100\ 100\ 010\ 010\ 000\ 001\ 000$$

gives

$$a\ 000\ e00\ b00\ 0f0\ 0c0\ 000\ 00d\ 000$$

Now it just so happens this number is configured such that the residues of the values of its bits modulo  $2^8 - 1$  are in exactly the opposite order from the bits of the original character and have the binary orders of magnitude 0-5. In other words this number is equal to the sum of the numbers in the upper row below, and dividing each of these summands by 255 gives the remainder listed in the lower row.

<i>Dividend</i>	$f \times 2^{13}$	$e \times 2^{20}$	$d \times 2^3$	$c \times 2^{10}$	$b \times 2^{17}$	$a \times 2^{24}$
<i>Remainder</i>	$f \times 2^5$	$e \times 2^4$	$d \times 2^3$	$c \times 2^2$	$b \times 2^1$	$a \times 2^0$

The remainder in a division is equal to the sum, modulo the divisor, of the remainders left by dividing each of the dividend summands by the same divisor. And the sum of the terms in the lower row is obviously *fedcba*. The above procedure is implemented by this sequence (due to Schroepfel\*) where the character is right-justified in accumulator A, and its reverse appears right-justified in accumulator A+1.

```

IMUL  A,[2020202]      ;4 copies shifted left one
AND   A,[104422010]   ;Pick bits for reverse
IDIVI A,3777          ;Divide by  $2^8 - 1$ 

```

\*HAKMEM 140, page 78  
(*Artificial Intelligence Memorandum, No. 239*, February 29, 1972, MIT Artificial Intelligence Laboratory).

These examples require that the rest of A, outside the character, be clear.

To reverse eight bits we can use a similar procedure (also due to Schroepfel) where again the original character is right-justified in A and its reverse appears right-justified in A+1. But this time we cannot manage the manipulation within a single length word, so we must use multiply, divide, and a pair of ANDs.

```

MUL   A,[100200401002] ;5 copies in A and A+1
AND   A+1,[20420420020] ;Pick bits for reverse via
ANDI  A,41              ;residues mod  $2^{10} - 1$ 
DIVI  A,1777           ;Divide by  $2^{10} - 1$ 

```

### Arithmetic Shifting

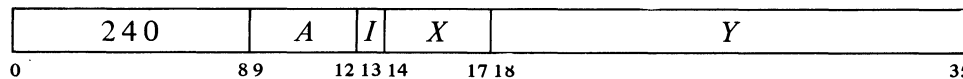
These two instructions produce an arithmetic shift right or left of the number in AC or the double length number in accumulators A and A+1. Shifting is the movement of the contents of a register bit-to-bit. The operation discussed here is similar to logical shifting [see § 2.4 and the illustration on page 2-24], but in an arithmetic shift only the magnitude part is shifted — the sign is unaffected. In a double length number the 70-bit string made up of the magnitude parts of the two words is shifted, but the sign of the low order word is made equal to the sign of the high order word.

Null bits are brought in at the end being vacated: a left shift brings in 0s at the right, whereas a right shift brings in the equivalent of the sign bit at the left. In either case, information shifted out at the other end is lost. A single

shift left is equivalent to multiplying the number by 2 (provided no bit of significance is shifted out); a shift right divides the number by 2.

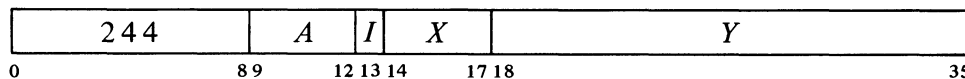
The number of places shifted is specified by the result of the effective address calculation taken as a signed number (in twos complement notation) modulo  $2^8$  in magnitude. In other words the effective shift  $E$  is the number composed of bit 18 (which is the sign) and bits 28–35 of the calculation result. Hence the programmer may specify the shift directly in the instruction (perhaps indexed) or give an indirect address to be used in calculating the shift. A positive  $E$  produces motion to the left, a negative  $E$  to the right;  $E$  is thus the power of 2 by which the number is multiplied. In the KA10, maximum movement is 255 places. The KI10 eliminates redundant movement by shifting at most 72 places regardless of the value of  $E$ . ▲

**ASH Arithmetic Shift**



Shift AC arithmetically the number of places specified by  $E$ . Do not shift bit 0. If  $E$  is positive, shift left bringing 0s into bit 35; data shifted out of bit 1 is lost; set Overflow if any bit of significance is lost (a 1 in a positive number, a 0 in a negative one). If  $E$  is negative, shift right bringing 0s into bit 1 if AC is positive, 1s if negative; data shifted out of bit 35 is lost.

**ASHC Arithmetic Shift Combined**



Concatenate the magnitude portions of accumulators  $A$  and  $A+1$  with  $A$  on the left, and shift the 70-bit combination in bits 1–35 and 37–71 the number of places specified by  $E$ . Do not shift AC bit 0, but make bit 0 of AC  $A+1$  equal to it if at least one shift occurs (*ie* if  $E$  is nonzero). If  $E$  is positive, shift left bringing 0s into bit 71 (bit 35 of AC  $A+1$ ); bit 37 (bit 1 of AC  $A+1$ ) is shifted into bit 35; data shifted out of bit 1 is lost; set Overflow if any bit of significance is lost (a 1 in a positive number, a 0 in a negative one). If  $E$  is negative, shift right bringing 0s into bit 1 if AC is positive, 1s if negative; bit 35 is shifted into bit 37; data shifted out of bit 71 is lost.

An arithmetic right shift truncates a negative result differently from IDIV if 1s are shifted out. The result of the shift is more negative by one than the quotient of IDIV.

To obtain the same quotient that IDIV would give with a dividend in  $A$  divided by  $N = 2^K$ , use

```
SKIPGE  A
ADDI    A,N-1
ASH     A,-K
```

For  $K < 20$  this is only slightly faster than IDIVI, except in the KA10 where it takes only 5–6  $\mu$ s as opposed to about 16  $\mu$ s for IDIVI.

Note that the effect of a shift on bit 0 of the low order word is consistent with the convention used for double length fixed point numbers. When there is no shift however, the result may be inconsistent with that convention. ▲

**2.6 FLOATING POINT ARITHMETIC**

For floating point arithmetic the PDP-10 has instructions for scaling the exponent (which is multiplication of the entire number by a power of 2)

In a KA10 without floating point hardware, all of the instructions presented in this section are trapped as unassigned codes [§2.10].

and negating double length numbers (software format) as well as performing addition, subtraction, multiplication and division of numbers in single precision floating point format. Moreover the KI10 has instructions for performing the four standard arithmetic operations on floating point numbers in hardware double precision format, for moving double precision numbers (with the option of taking the negative) between a pair of accumulators and a pair of memory locations, and for converting single precision numbers from fixed format to floating and vice versa. Except for the conversion instructions and the simple moves, all instructions treated here interpret all operands as floating point numbers in the formats given in §1.1, and generate results in those formats. The reader is strongly advised to reread §1.1 if he does not remember the formats in detail.

For the four standard arithmetic operations in single precision, the program can select whether or not the result shall be rounded. Rounding produces the greatest consistent precision using only single length operands. Instructions without rounding have a "long" mode, which supplies a two-word result for greater precision; the other modes save time in one-word operations where rounding is of no significance.

Actually the result is formed in a double length register in addition, subtraction and multiplication, wherein any bits of significance in the low order part supply information for normalization, and then for rounding if requested. Consider addition as an example. Before adding, the processor right shifts the fractional part of the operand with the smaller exponent until its bits correctly match the bits of the other operand in order of magnitude. Thus the smaller operand could disappear entirely, having no effect on the result ("result" shall always be taken to mean the information (one word or two) stored by the instruction, regardless of the number of significant bits it contains or even whether it is the correct answer). Long mode is likely to retain information that would otherwise be lost, but in any given mode the significance of the result depends on the relative values of the operands. Even when both operands contain twenty-seven significant bits, a long addition may store two words that together contain only one significant bit. In division the processor always calculates a one-word quotient that requires no normalization if the original operands are normalized. An extra quotient bit is calculated for rounding when requested; long mode retains the remainder.

Among the floating point instructions available only in the KI10, those that convert between number types operate only on single words. The instruction that converts to floating point assumes the operand is an integer and always normalizes and rounds the result. In the opposite direction, only the integral part of the result is saved, and rounding is an option of the program. The instructions for the four standard operations using double precision have no modes. In division the processor always calculates a two-word quotient that is normalized if the original operands are normalized, but rounding is not available. In addition, subtraction and multiplication, the result is formed in a triple length register, wherein bits of significance in the lowest order part supply information for limited normalization and then for rounding, which is automatic.

The processor has four flags, Overflow, Floating Overflow, Floating Underflow and No Divide, that indicate when the exponent is too large or

A subtraction involving two like-signed numbers whose exponents are equal and whose fractions differ only in the LSB gives a result containing only one bit of significance.

too small to be accommodated or a division cannot be performed because of the relative values of dividend and divisor. Except where the result would be in fixed point form, any of these circumstances sets Overflow and Floating Overflow. If only these two are set, the exponent of the answer is too large; if Floating Underflow is also set, the exponent is too small. No Divide being set means the processor failed to perform a division, an event that can be produced only by a zero divisor if all nonzero operands are normalized. Any condition that sets Overflow in the KI10 also sets the Trap 1 flag. These flags can be read and controlled by certain program control instructions [ §§2.9, 2.10]. KI10 overflow is handled by trapping through the setting of Trap 1 [ §2.9], but in the KA10, the program must make direct use of Overflow and Floating Overflow, which are available as processor conditions (via an in-out instruction) that can request a priority interrupt if enabled [ §2.14]. The conditions detected can only set the arithmetic flags and the hardware does not clear them, so the program must clear them before a floating point instruction if they are to give meaningful information about the instruction afterward. However, the program can check the flags following a series of instructions to determine whether the entire series was free of the types of error detected.

The floating point hardware functions at its best if given operands that are either normalized or zero, and except in special situations the hardware normalizes a nonzero result. An operand with a zero fraction and a nonzero exponent can give wild answers in additive operations because of extreme loss of significance; *eg* adding  $\frac{1}{2} \times 2^2$  and  $0 \times 2^{69}$  gives a zero result, as the first operand (having a smaller exponent) looks smaller to the processor and is shifted to oblivion. A number with a 1 in bit 0 and 0s in bits 9–35 is not simply an incorrect representation of zero, but rather an unnormalized “fraction” with value  $-1$ . This unnormalized number can produce an incorrect answer in any operation. Use of other unnormalized operands simply causes loss of significant bits, except in division where they can prevent its execution because they can satisfy a no-divide condition that is impossible for normalized numbers.

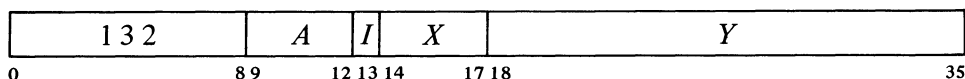
### Scaling

One floating point instruction is in a category by itself: it changes the exponent of a number without changing the significance of the fraction. In other words it multiplies the number by a power of 2, and is thus analogous to arithmetic shifting of fixed point numbers except that no information is lost, although the exponent can overflow or underflow. The amount added to the exponent is specified by the result of the effective address calculation taken as a signed number (in twos complement notation) modulo  $2^8$  in magnitude. In other words the effective scale factor  $E$  is the number composed of bit 18 (which is the sign) and bits 28–35 of the calculation result. Hence the programmer may specify the factor directly in the instruction (perhaps indexed) or give an indirect address to be used in calculating it. A positive  $E$  increases the exponent, a negative  $E$  decreases it;  $E$  is thus the power of 2 by which the number is multiplied. The scale factor lies in the range  $-256$  to  $+255$ .

In the KI10 an arithmetic instruction executed as an interrupt instruction can set no flags.

The processor normalizes the result by shifting the fraction and adjusting the exponent to compensate for the change in value. Each shift and accompanying exponent adjustment thus multiply the number both by 2 and by  $\frac{1}{2}$  simultaneously, leaving its value unchanged.

Note that with normalized operands, the processor uses at most two bits of information from the lowest order part to normalize the result. In multiplication this is obvious, since squaring the minimum fractional magnitude  $\frac{1}{2}$  gives a result of  $\frac{1}{4}$ . In an addition or subtraction of numbers that differ greatly in order of magnitude, the result is determined almost completely by the operand of greater order. A subtraction involving two like-signed numbers with equal exponents requires no shifting beforehand so there is no information in the lowest order part. Hence an addition or subtraction never requires shifting both before the operation and in the normalization; when there is no prior shifting, the normalization brings in 0s.

**FSC Floating Scale**

This instruction can be used to float a fixed number with 27 or fewer significant bits. To float an integer contained within AC bits 9–35,

**FSC AC,233**

inserts the correct exponent to move the binary point from the right end to the left of bit 9 and then normalizes ( $233_8 = 155_{10} = 128 + 27$ ).

If the fractional part of AC is zero, clear AC. Otherwise add the scale factor given by  $E$  to the exponent part of AC (thus multiplying AC by  $2^E$ ), normalize the resulting word bringing 0s into bit positions vacated at the right, and place the result back in AC.

**NOTE**

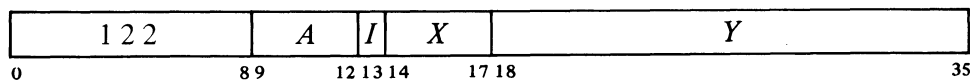
A negative  $E$  is represented in standard twos complement notation, but the hardware compensates for this when scaling the exponent.

If the exponent after normalization is  $> 127$ , set Overflow and Floating Overflow; the result stored has an exponent 256 less than the correct one. If  $< -128$ , set Overflow, Floating Overflow and Floating Underflow; the result stored has an exponent 256 greater than the correct one.

**Number Conversion**

In the KA10 these instructions are trapped as unassigned codes.

Although FSC can be used to float a fixed point number, the KI10 has three single precision instructions specifically for converting between integers and floating point numbers. In all cases the operand is taken from location  $E$ , and the converted result is placed in AC.

**FIX Fix**

This overflow test checks for a value  $\geq 2^{35}$  assuming the operand is normalized.

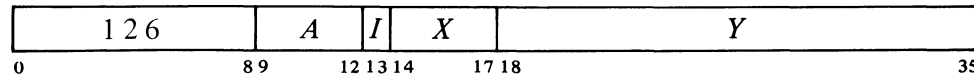
If the exponent of the floating point number in location  $E$  is  $> 35$ , set Overflow and Trap 1, and go immediately to the next instruction without affecting AC or the contents of  $E$  in any way.

Otherwise replace the exponent  $X$  in the word from location  $E$  with bits equal to the sign of the fraction, and shift the (now fixed) extended fraction  $N = X - 27$  places to the correct position for its order of magnitude with the binary point at the right of bit 35. For positive  $N$ , shift left bringing 0s into bit 35 and dropping null bits out of bit 1. For negative  $N$ , shift right bringing null bits (0s for positive, 1s for negative) into bit 1, and then truncate to an integer. Place the result in AC.

This is the standard Fortran truncation ("fixation"). For it, the processor drops the

Truncation produces the integer of largest magnitude less than or equal to the magnitude of the original number. Eg a number  $> +1$  but  $< +2$  becomes  $+1$ ; a number  $< -1$  but  $> -2$  becomes  $-1$ .

**FIXR      Fix and Round**



If the exponent of the floating point number in location *E* is  $> 35$ , set Overflow and Trap 1, and go immediately to the next instruction without affecting AC or the contents of *E* in any way.

Otherwise replace the exponent *X* in the word from location *E* with bits equal to the sign of the fraction, and shift the (now fixed) extended fraction  $N = X - 27$  places to the correct position for its order of magnitude with the binary point at the right of bit 35. For positive *N*, shift left bringing 0s into bit 35 and dropping null bits out of bit 1. For negative *N*, shift right bringing null bits (0s for positive, 1s for negative) into bit 1, and then round the integral part. Place the result in AC.

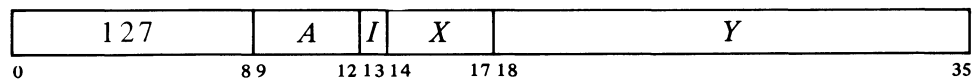
Rounding is in the positive direction: the magnitude of the integral part is increased by one if the fractional part is  $\geq \frac{1}{2}$  in a positive number but  $> \frac{1}{2}$  in a negative number. Eg +1.4 (decimal) is rounded to +1, whereas +1.5 and +1.6 become +2; but with negative numbers, -1.4 and -1.5 become -1, whereas -1.6 becomes -2.

fractional part in a positive number, but adds one to the integral part (as required by twos complement format) if any bits of significance are shifted out in a negative number.

This overflow test checks for a value  $\geq 2^{35}$  assuming the operand is normalized.

This is the Algol standard for real to integer conversion. For it the processor adds one to the integral part if the fractional part is  $\geq \frac{1}{2}$  in a positive number or (as required by twos complement format) is  $\leq \frac{1}{2}$  in a negative number.

**FLTR      Float and Round**



Shift the magnitude part of the fixed point integer from location *E* right eight places, insert the exponent 35 (in proper form) into bits 1-8 to move the shifted binary point to the left of bit 9 ( $35 = 27 + 8$ ), and normalize the fraction bringing first the bits originally shifted out and then 0s into bit positions vacated at the right. If fewer than eight bits (left shifts) are needed to normalize, use the next bit to round the single length fraction. Place the result in AC.

The rounding function is the same as that used by the standard floating point arithmetic instructions [see below].

Since the largest fixed point magnitude (without considering sign) is  $2^{35} - 1$ , a floating point number with exponent greater than 35 (and assumed normalized) cannot be converted to fixed point. There is no limit in the opposite direction, but precision can be lost as floating point format provides fewer significant bits. A fixed integer greater than  $2^{27} - 1$  cannot be represented exactly in floating point unless all its significant bits are clustered within a group of twenty-seven.

### Single Precision with Rounding

In the hardware the rounding operation is actually somewhat more complex than stated here. If the result is negative, the hardware combines rounding with placing the high order word in twos complement form by decreasing its magnitude if the low order part is  $< \frac{1}{2}\text{LSB}$ . Moreover an extra single-step re-normalization occurs if the rounded word is no longer normalized.

There are four instructions that use only one-word operands and store a single-length rounded result. Rounding is away from zero: if the part of the normalized answer being dropped (the low order part of the fraction) is greater than or equal in magnitude to one half the LSB of the part being retained, the magnitude of the latter part is increased by one LSB.

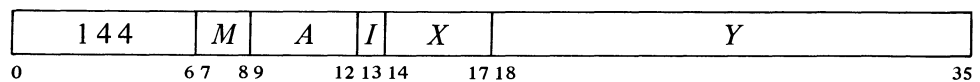
The rounding instructions have four modes that determine the source of the non-AC operand and the destination of the result. These modes are like those of logic and fixed point arithmetic, including an immediate mode that allows the instruction to carry an operand with it.

<i>Mode</i>	<i>Suffix</i>	<i>Source of non-AC operand</i>	<i>Destination of result</i>
Basic		<i>E</i>	AC
Immediate	I	The word <i>E,0</i>	AC
Memory	M	<i>E</i>	<i>E</i>
Both	B	<i>E</i>	AC and <i>E</i>

Note however that floating point immediate uses *E,0* as an operand, not *0,E*. In other words the half word *E* is interpreted as a sign, an 8-bit exponent, and a 9-bit fraction.

In each of these instructions, the exponent that results from normalization and rounding is tested for overflow or underflow. If the exponent is  $> 127$ , set Overflow and Floating Overflow; the result stored has an exponent 256 less than the correct one. If  $< -128$ , set Overflow, Floating Overflow and Floating Underflow; the result stored has an exponent 256 greater than the correct one.

### FADR Floating Add and Round

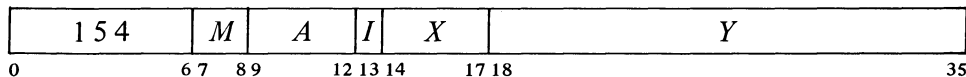


Floating add the operand specified by *M* to AC. If the double length fraction in the sum is zero, clear the specified destination. Otherwise normalize the double length sum bringing 0s into bit positions vacated at the right, round the high order part, test for exponent overflow or underflow as described above, and place the result in the specified destination.

FADR	Floating Add and Round	144
FADRI	Floating Add and Round Immediate	145
FADRM	Floating Add and Round to Memory	146
FADRB	Floating Add and Round to Both	147



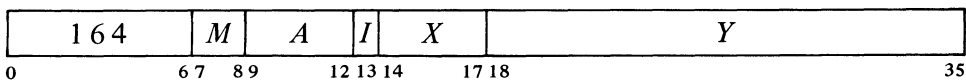
**FSBR Floating Subtract and Round**



Floating subtract the operand specified by *M* from AC. If the double length fraction in the difference is zero, clear the specified destination. Otherwise normalize the double length difference bringing 0s into bit positions vacated at the right, round the high order part, test for exponent overflow or underflow as described above, and place the result in the specified destination.

FSBR	Floating Subtract and Round	154
FSBRI	Floating Subtract and Round Immediate	155
FSBRM	Floating Subtract and Round to Memory	156
FSBRB	Floating Subtract and Round to Both	157

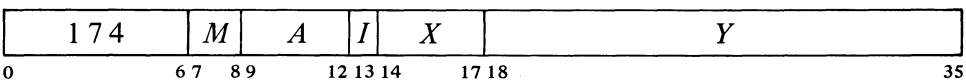
**FMPR Floating Multiply and Round**



Floating Multiply AC by the operand specified by *M*. If the double length fraction in the product is zero, clear the specified destination. Otherwise normalize the double length product bringing 0s into bit positions vacated at the right, round the high order part, test for exponent overflow or underflow as described above, and place the result in the specified destination.

FMPR	Floating Multiply and Round	164
FMPRI	Floating Multiply and Round Immediate	165
FMPRM	Floating Multiply and Round to Memory	166
FMPRB	Floating Multiply and Round to Both	167

**FDVR Floating Divide and Round**



If the magnitude of the fraction in AC is greater than or equal to twice that of the fraction in the operand specified by *M*, set Overflow, Floating Overflow and No Divide, and go immediately to the next instruction without affecting the original AC or memory operand in any way.

If the division can be performed, floating divide AC by the operand specified by *M*, calculating a quotient fraction of 28 bits (this includes an extra bit for rounding). If the fraction is zero, clear the specified destination. Otherwise round the fraction using the extra bit calculated. If the original

Division fails if the divisor is zero, but the no-divide condition can otherwise be satisfied only if at least one operand is unnormalized.

operands were normalized, the single length quotient will already be normalized; if it is not, normalize it bringing 0s into bit positions vacated at the right. Test for exponent overflow or underflow as described above, and place the result in the specified destination.

FDVR	Floating Divide and Round	174
FDVRI	Floating Divide and Round Immediate	175
FDVRM	Floating Divide and Round to Memory	176
FDVRB	Floating Divide and Round to Both	177

### Single Precision without Rounding

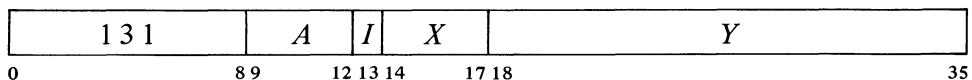
Instructions that do not round are faster for processing floating point numbers with fractions containing fewer than 27 significant bits. On the other hand the long mode provides double precision (software format) or allows the programmer to use his own method of rounding. Besides the four usual arithmetic operations with normalization, there are two nonnormalizing instructions that facilitate software double precision arithmetic [§2.11 gives examples of double precision floating point routines]. These two instructions have no modes.

Note that this instruction can be used to negate numbers in software double precision format only; for the KI10 hardware double precision format, the program must use the double moves.

Usually the double length number is in two adjacent accumulators, and  $E$  equals  $A+1$ . There is no overflow test, as negating a correctly formatted floating point number cannot cause overflow.

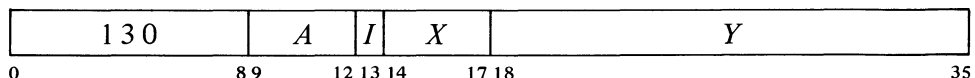
DFN AC,AC is undefined.

#### DFN Double Floating Negate



Negate the double length floating point number composed of the contents of AC and location  $E$  with AC on the left. Do this by taking the two's complement of the number whose sign is AC bit 0, whose exponent is in AC bits 1–8, and whose fraction is the 54-bit string in bits 9–35 of AC and location  $E$ . Place the high order word of the result in AC; place the low order part of the fraction in bits 9–35 of location  $E$  without altering the original contents of bits 0–8 of that location.

#### UFA Unnormalized Floating Add



Floating add the contents of location  $E$  to AC. If the double length fraction in the sum is zero, clear accumulator  $A+1$ . Otherwise normalize the sum only if the magnitude of its fractional part is  $\geq 1$ , and place the high order part of the result in AC  $A+1$ . The original contents of AC and  $E$  are unaffected.

The caution given below for FAD applies to this instruction as well.

NOTE

The result is placed in accumulator  $A+1$ . This is the only arithmetic instruction that stores the result in a second accumulator, leaving the original operands intact.

If the exponent of the sum following the one-step normalization is  $> 127$ , set Overflow and Floating Overflow; the result stored has an exponent 256 less than the correct one.

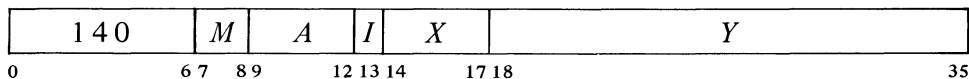
The exponent of the sum is equal to that of the larger summand unless addition of the fractions overflows, in which case it is greater by 1. Exponent overflow can occur only in the latter case.

The remaining single precision floating point instructions perform the four standard arithmetic operations with normalization but without rounding. All use AC and the contents of location  $E$  as operands and have four modes.

<i>Mode</i>	<i>Suffix</i>	<i>Effect</i>
Basic		High order word of result stored in AC.
Long	L	In addition, subtraction and multiplication, the two-word result (in the software double length format described in §1.1) is stored in accumulators $A$ and $A+1$ . In division the dividend is the double length word in $A$ and $A+1$ ; the single length quotient is stored in AC, the remainder in AC $A+1$ .
Memory	M	High order word of result stored in $E$ .
Both	B	High order word of result stored in AC and $E$ .

In each of these instructions, the exponent that results from normalization is tested for overflow or underflow. If the exponent is  $> 127$ , set Overflow and Floating Overflow; the result stored has an exponent 256 less than the correct one. If  $< -128$ , set Overflow, Floating Overflow and Floating Underflow; the result stored has an exponent 256 greater than the correct one.

**FAD Floating Add**



Floating add the contents of location  $E$  to AC. If the double length fraction in the sum is zero, clear the destination specified by  $M$ , clearing both accumulators in long mode. Otherwise normalize the double length sum bringing 0s into bit positions vacated at the right, test for exponent overflow or

*CAUTION*

In single precision addition the term with the smaller exponent is right shifted in a double

length register, specifically a register with 54 magnitude bits. Now if the difference in the exponents is  $< 54$ , there is at least one significant bit after the shift (assuming normalized operands); and if the difference is  $> 64$ , the hardware throws the term away by substituting zero. But when the exponent difference lies in the range 54 to 64, the procedure disposes of all significant bits without actually substituting zero. This means that if the shifted term is positive it appears in the addition as all 0s, but if negative it appears as all 1s. The latter case gives an answer that is less by one LSB.

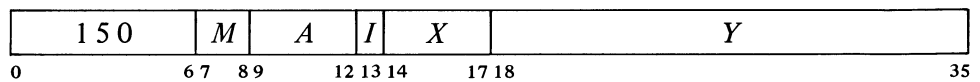
The caution given above for addition applies also to subtraction, which is done by adding with the minuend negated. Here the lesser answer (as against a true zero substitution) occurs when the term with the smaller exponent is negative after the minuend negation, *ie* when it is a negative subtrahend but a positive minuend.

underflow as described above, and place the high order word of the result in the specified destination.

- ▲ In long mode if the exponent of the sum is  $< -101$  ( $-128 + 27$ ) or the low order half of the fraction is zero, clear AC  $A+1$ . Otherwise place a low order word for a double length result in  $A+1$  by putting a 0 in bit 0, an exponent in positive form 27 less than the exponent of the sum in bits 1-8, and the low order part of the fraction in bits 9-35.

<b>FAD</b>	Floating Add	140
<b>FADL</b>	Floating Add Long	141
<b>FADM</b>	Floating Add to Memory	142
<b>FADB</b>	Floating Add to Both	143

**FSB Floating Subtract**

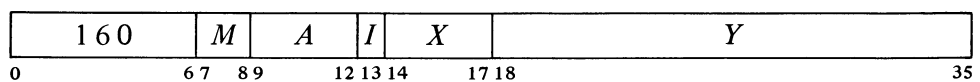


Floating subtract the contents of location  $E$  from AC. If the double length fraction in the difference is zero, clear the destination specified by  $M$ , clearing both accumulators in long mode. Otherwise normalize the double length difference bringing 0s into bit positions vacated at the right, test for exponent overflow or underflow as described above, and place the high order word of the result in the specified destination.

- ▲ In long mode if the exponent of the difference is  $< -101$  ( $-128 + 27$ ) or the low order half of the fraction is zero, clear AC  $A+1$ . Otherwise place a low order word for a double length result in  $A+1$  by putting a 0 in bit 0, an exponent in positive form 27 less than the exponent of the difference in bits 1-8, and the low order part of the fraction in bits 9-35.

<b>FSB</b>	Floating Subtract	150
<b>FSBL</b>	Floating Subtract Long	151
<b>FSBM</b>	Floating Subtract to Memory	152
<b>FSBB</b>	Floating Subtract to Both	153

**FMP Floating Multiply**



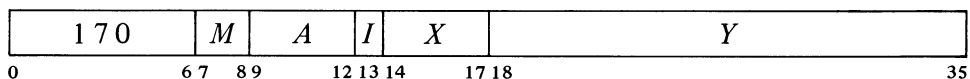
Floating multiply AC by the contents of location  $E$ . If the double length fraction in the product is zero, clear the destination specified by  $M$ , clearing both accumulators in long mode. Otherwise normalize the double length

product bringing 0s into bit positions vacated at the right, test for exponent overflow or underflow as described above, and place the high order word of the result in the specified destination.

In long mode if the exponent of the product is  $> 154$  ( $127 + 27$ ) or  $< -101$  ( $-128 + 27$ ) or the low order half of the fraction is zero, clear AC  $A+1$ . Otherwise place a low order word for a double length result in  $A+1$  by putting a 0 in bit 0, an exponent in positive form 27 less than the exponent of the product in bits 1-8, and the low order part of the fraction in bits 9-35.

FMP	Floating Multiply	160
F MPL	Floating Multiply Long	161
FMPM	Floating Multiply to Memory	162
FMPB	Floating Multiply to Both	163

**FDV Floating Divide**



If the magnitude of the fraction in AC (in long mode, the high order part of the magnitude of the double length fraction in accumulators  $A$  and  $A+1$ ) is greater than or equal to twice the magnitude of the fraction in location  $E$ , set Overflow, Floating Overflow and No Divide, and go immediately to the next instruction without affecting the original AC or memory operand in any way.

If division can be performed, floating divide the AC operand by the contents of location  $E$ . In long mode the AC operand (the dividend) is the double length number in accumulators  $A$  and  $A+1$ ; in other modes it is the single word in AC. Calculate a quotient fraction of 27 bits. If the fraction is zero, clear the destination specified by  $M$ , clearing both accumulators in long mode if the double length dividend was zero. A quotient with a nonzero fraction will already be normalized if the original operands were normalized; if it is not, normalize it bringing 0s into bit positions vacated at the right. Test for exponent overflow or underflow as described above, and place the single length quotient part of the result in the specified destination.

In long mode calculate the exponent for the fractional remainder from the division according to the relative magnitudes of the fractions in dividend and divisor: if the dividend was greater than or equal to the divisor, the exponent of the remainder is 26 less than that of the dividend, otherwise it is 27 less. If the remainder exponent is  $< -128$  or the fraction is zero, clear AC  $A+1$ . Otherwise place the floating point remainder (exponent and fraction) with the sign of the dividend in AC  $A+1$ .

FDV	Floating Divide	170
FDVL	Floating Divide Long	171
FDVM	Floating Divide to Memory	172
FDVB	Floating Divide to Both	173

Division fails if the divisor is zero, but the no-divide condition can otherwise be satisfied only if at least one operand is unnormalized.

In long mode a nonzero unnormalized dividend whose entire high order fraction is zero produces a zero quotient. In this case the second AC is cleared in the KI10 but may receive rubbish in the KA10.

### Double Precision Operations

In the KA10 these instructions are trapped as unassigned codes.

Although double precision floating point arithmetic can be done by routines using the single precision instructions and the software double length format, the KI10 has instructions specifically for handling double length operands in the hardware double precision format described in §1.1. Four of the instructions use two double length operands, perform the standard arithmetic operations, and store double length results. The other four instructions each move one double length operand between the accumulators and memory, either unchanged or negated.

All of these instructions address a pair of adjacent accumulators and a pair of adjacent memory locations. The accumulators have addresses  $A$  and  $A+1 \pmod{20_8}$  just as they do for the double length operands used in some shift, rotate and single precision arithmetic instructions. The memory locations have addresses  $E$  and  $E+1 \pmod{2^{18}}$ , where the second address is 0 if  $E$  is 77777.

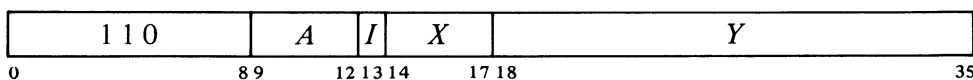
For the two instructions that simply move a pair of words without altering them, the format of those words is actually irrelevant. The other six instructions process each word pair as a double length number in the hardware floating point format. Hence they ignore bit 0 in the low order word of every operand and clear that bit in the result.

The four nonmove instructions perform the standard arithmetic operations. All use two double length operands in the hardware double precision format, one from the accumulators and one from memory. Addition and subtraction always normalize the result; in multiplication and division the result is guaranteed to be normalized only if the original operands are normalized. In all cases the result, rounded except in division, is placed in the accumulators. The rounding function is the same as that used in single precision: if the part of the answer being dropped (the low order part of the fraction) is greater than or equal in magnitude to one half the LSB of the double length part being retained, the magnitude of the latter part is increased by one LSB (with appropriate adjustment for a twos complement negative).

An arithmetic instruction executed as an interrupt instruction can set no flags.

In each of these instructions, the exponent that results from normalization and rounding (if done) is tested for overflow or underflow. If the exponent is  $> 127$ , set Overflow and Floating Overflow; the result stored has an exponent 256 less than the correct one. If  $< -128$ , set Overflow, Floating Overflow and Floating Underflow; the result stored has an exponent 256 greater than the correct one. Setting Overflow also sets the Trap 1 flag.

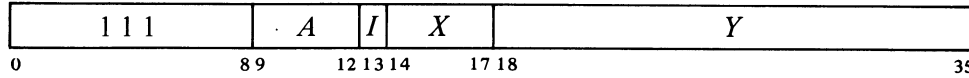
#### DFAD Double Floating Add



Floating add the operand of locations  $E$  and  $E+1$  to the operand of accumulators  $A$  and  $A+1$ . If the high order 70 bits of the fraction in the

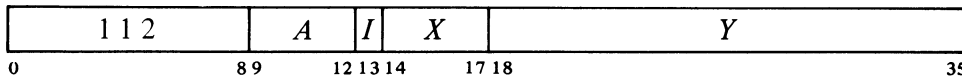
sum are zero, clear  $A$  and  $A+1$ . Otherwise normalize the triple length sum bringing 0s in at the right, round the high order double length part, test for exponent overflow or underflow as described above, and place the result in ACs  $A$  and  $A+1$ .

**DFSB Double Floating Subtract**



Floating subtract the operand of locations  $E$  and  $E+1$  from the operand of accumulators  $A$  and  $A+1$ . If the high order 70 bits of the fraction in the difference are zero, clear  $A$  and  $A+1$ . Otherwise normalize the triple length difference bringing 0s into bit positions vacated at the right, round the high order double length part, test for exponent overflow or underflow as described above, and place the result in ACs  $A$  and  $A+1$ .

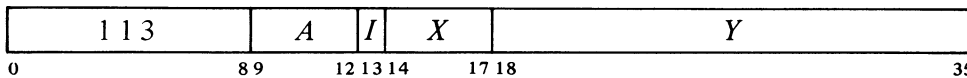
**DFMP Double Floating Multiply**



Floating multiply the operand of accumulators  $A$  and  $A+1$  by the operand of locations  $E$  and  $E+1$ . If the high order 70 bits of the fraction in the product are zero, clear  $A$  and  $A+1$ . Otherwise, if there are any bits of significance among the high order 35, do at most one normalization shift if required; if the high order 35 bits are zero, shift the fraction left 35 places (adjusting the exponent), and then do at most one normalization shift if required. Round the high order double length part, test for exponent overflow and underflow as described above, and place the result in ACs  $A$  and  $A+1$ .

The 35-bit shift can be done only if the original operands are unnormalized.

**DFDV Double Floating Divide**



If the magnitude of the fraction in the operand of accumulators  $A$  and  $A+1$  is greater than or equal to twice that of the fraction in the operand of locations  $E$  and  $E+1$ , set Overflow, Floating Overflow, No Divide and Trap 1, and go immediately to the next instruction without affecting the original AC or memory operands in any way.

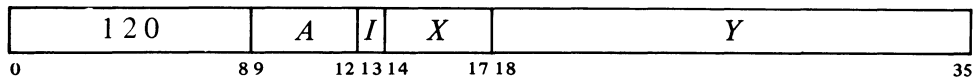
If the division can be performed, floating divide the AC operand by the memory operand, calculating a quotient fraction of 62 bits. If the fraction

Division fails if the divisor is zero, but the no-divide condition can otherwise be satisfied only if at least one operand is unnormalized.

A nonzero quotient is normalized if the original operands are normalized.

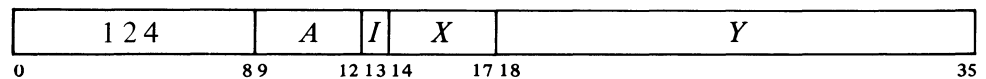
is zero, clear  $A$  and  $A+1$ . Otherwise test for exponent overflow or underflow as described above, and place the double length quotient part of the result in ACs  $A$  and  $A+1$  (the remainder is lost).

#### DMOVE Double Move



Move the contents of locations  $E$  and  $E+1$  respectively to accumulators  $A$  and  $A+1$ . The memory locations are unaffected, the original contents of the ACs are lost.

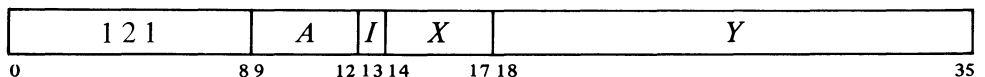
#### DMOVEM Double Move to Memory



Move the contents of accumulators  $A$  and  $A+1$  respectively to locations  $E$  and  $E+1$ . The ACs are unaffected, the original contents of the memory locations are lost.

Do not use the instruction `DMOVEM AC,AC+1`. At present the processor places AC in both  $AC+1$  and  $AC+2$ , but this result is not guaranteed.

#### DMOVN Double Move Negative

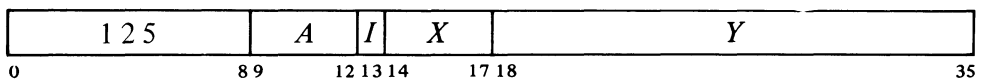


Negate the double length floating point number taken from locations  $E$  and  $E+1$ , and move it to accumulators  $A$  and  $A+1$ . The memory locations are unaffected, the original contents of the ACs are lost.

Note that these two instructions can be used to negate numbers in hardware double precision format only; for software double precision, the program must use DFN.

Note also that there is no overflow test, as negating a correctly formatted floating point number cannot cause overflow.

#### DMOVNM Double Move Negative to Memory



Negate the double length floating point number taken from accumulators  $A$  and  $A+1$ , and move it to locations  $E$  and  $E+1$ . The ACs are unaffected, the original contents of the memory locations are lost.

Do not use the instruction `DMOVNM AC,AC+1`. At pre-



Although the configuration of the operands is irrelevant in DMOVE and DMOVEM, none of the above instructions is available in the KA10. Therefore unless a program is actually doing floating point arithmetic in the hardware double precision format, it is recommended that the double moves not be used in KI10 programs so they will be compatible with the KA10. Simply to move a two-word operand unaltered requires two one-word moves. To negate a two-word operand that is actually in the hardware format requires a somewhat longer substitution; eg this sequence is equivalent to DMOVN AC,E.

sent the processor places the negative of AC (the complement, if AC+1 originally contains zero) into AC+1, and the negative of that into AC+2, but this result is not guaranteed.

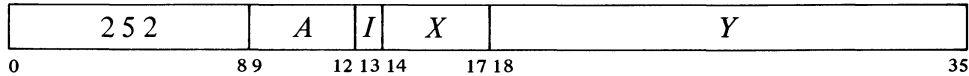
```

SETCM AC,E      ;Take ones complement of high word
MOVN  AC+1,E+1  ;Take twos complement of low word
TLZ   AC+1,400000 ;Clear bit 0
SKIPN AC+1      ;If low part is zero, change high word
ADDI  AC,1      ;to twos complement
    
```

### 2.7 ARITHMETIC TESTING

These instructions may jump or skip depending on the result of an arithmetic test and may first perform an arithmetic operation on the test word. Two of the instructions have no modes.

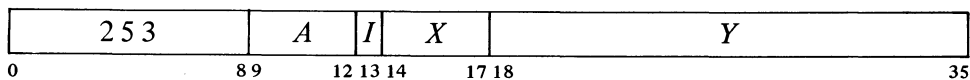
#### **AOBJP Add One to Both Halves of AC and Jump if Positive**



Add one to each half of AC and place the result back in AC. If the result is greater than or equal to zero (*ie* if bit 0 is 0, and hence a negative count in the left half has reached zero or a positive count has not yet reached  $2^{17}$ ), take the next instruction from location *E* and continue sequential operation from there.

Note: The KA10 increments the two halves of AC by adding  $1\ 000001_8$  to the entire register. In the KI10 the two halves are handled independently.

#### **AOBJN Add One to Both Halves of AC and Jump if Negative**



Add one to each half of AC and place the result back in AC. If the result is less than zero (*ie* if bit 0 is 1, and hence a negative count in the left half has not yet reached zero or a positive count has reached  $2^{17}$ ), take the next instruction from location *E* and continue sequential operation from there.

Note: The KA10 increments the two halves of AC by adding  $1\ 000001_8$  to the entire register. In the KI10 the two halves are handled independently.

In the KA10, incrementing both halves of AC together is effected by adding  $1\ 000001_8$ . A count of  $-2$  in AC left is therefore increased to zero if  $2^{18} - 1$  is incremented in AC right.

These two instructions allow the program to keep a control count in the left half of an index register and require only one data transfer to initialize. Problem: Add 3 to each location in a table of  $N$  entries starting at TAB. Only four instructions are required.

MOVSI	XR, $-N$	;Put $-N$ in XR left (clear XR right)
MOVEI	AC, 3	;Put 3 in AC
ADDM	AC, TAB(XR)	;Add 3 to entry
AOBJN	XR, $-1$	;Update XR and go back unless all ;entries accounted for

The eight remaining instructions jump or skip if the operand or operands satisfy a test condition specified by the mode.

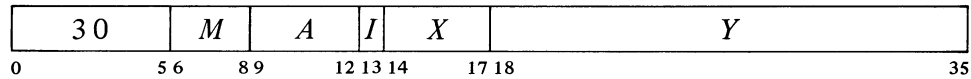
<i>Mode</i>	<i>Suffix</i>
Never	
Less	L
Equal	E
Less or Equal	LE
Always	A
Greater or Equal	GE
Not Equal	N
Greater	G

Instructions with one operand compare AC or the contents of location  $E$  with zero, those with two compare AC with  $E$  or the contents of location  $E$ . The processor always makes the comparison even though the result is used in only six of the modes. If the mnemonic has no suffix there is never any program control function, and the instruction may be a no-op; an A suffix produces an unconditional jump or skip – the action is always taken regardless of how the two quantities compare.

The last four of these instructions perform arithmetic operations, which are checked for overflow. In the KI10 any condition that sets Overflow also sets the Trap 1 flag.

In the KI10 an arithmetic instruction executed as an interrupt instruction can set no flags.

**CAI Compare AC Immediate and Skip if Condition Satisfied**

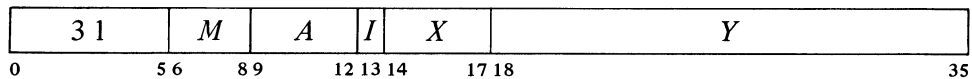


Compare AC with *E* (ie with the word 0, *E*) and skip the next instruction in sequence if the condition specified by *M* is satisfied.

<b>CAI</b>	Compare AC Immediate but Do Not Skip	300
<b>CAIL</b>	Compare AC Immediate and Skip if AC Less than <i>E</i>	301
<b>CAIE</b>	Compare AC Immediate and Skip if Equal	302
<b>CAILE</b>	Compare AC Immediate and Skip if AC Less than or Equal to <i>E</i>	303
<b>CAIA</b>	Compare AC Immediate but Always Skip	304
<b>CAIGE</b>	Compare AC Immediate and Skip if AC Greater than or Equal to <i>E</i>	305
<b>CAIN</b>	Compare AC Immediate and Skip if Not Equal	306
<b>CAIG</b>	Compare AC Immediate and Skip if AC Greater than <i>E</i>	307

CAI is a no-op in which *I*, *X* and *Y* are reserved for future use and should be zero (at present *E* is ignored).

**CAM Compare AC with Memory and Skip if Condition Satisfied**

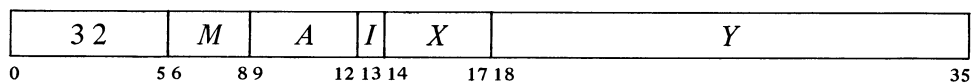


Compare AC with the contents of location *E* and skip the next instruction in sequence if the condition specified by *M* is satisfied. The pair of numbers compared may be either both fixed or both normalized floating point.

<b>CAM</b>	Compare AC with Memory but Do Not Skip	310
<b>CAML</b>	Compare AC with Memory and Skip if AC Less	311
<b>CAME</b>	Compare AC with Memory and Skip if Equal	312
<b>CAMLE</b>	Compare AC with Memory and Skip if AC Less or Equal	313
<b>CAMA</b>	Compare AC with Memory but Always Skip	314
<b>CAMGE</b>	Compare AC with Memory and Skip if AC Greater or Equal	315
<b>CAMN</b>	Compare AC with Memory and Skip if Not Equal	316
<b>CAMG</b>	Compare AC with Memory and Skip if AC Greater	317

CAM is a no-op that references memory.

**JUMP Jump if AC Condition Satisfied**



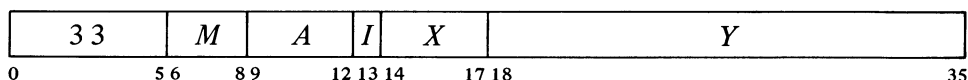
Compare AC (fixed or floating) with zero, and if the condition specified by

$M$  is satisfied, take the next instruction from location  $E$  and continue sequential operation from there.

JUMP is a no-op (instruction code 320 has this mnemonic for symmetry). In it,  $I$ ,  $X$  and  $Y$  are reserved for future use and should be zero (at present  $E$  is ignored).

JUMP	Do Not Jump	320
JUMPL	Jump if AC Less than Zero	321
JUMPE	Jump if AC Equal to Zero	322
JUMPLE	Jump if AC Less than or Equal to Zero	323
JUMPA	Jump Always	324
JUMPG	Jump if AC Greater than or Equal to Zero	325
JUMPN	Jump if AC Not Equal to Zero	326
JUMPG	Jump if AC Greater than Zero	327

### SKIP Skip if Memory Condition Satisfied



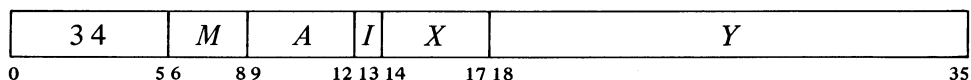
If  $A$  is zero, SKIP is a no-op; otherwise it is equivalent to MOVE. (Instruction code 330 has mnemonic SKIP for symmetry.)

Compare the contents (fixed or floating) of location  $E$  with zero, and skip the next instruction in sequence if the condition specified by  $M$  is satisfied. If  $A$  is nonzero also place the contents of location  $E$  in AC.

SKIP	Do Not Skip	330
SKIPL	Skip if Memory Less than Zero	331
SKIPE	Skip if Memory Equal to Zero	332
SKIPL	Skip if Memory Less than or Equal to Zero	333
SKIP	Skip Always	334
SKIPGE	Skip if Memory Greater than or Equal to Zero	335
SKIPN	Skip if Memory Not Equal to Zero	336
SKIPG	Skip if Memory Greater than Zero	337

SKIPA is a convenient way to load an accumulator and skip over an instruction upon entering a loop.

### AOJ Add One to AC and Jump if Condition Satisfied

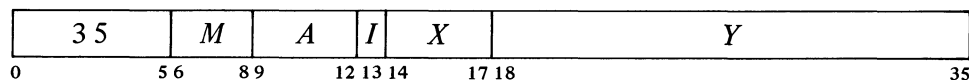


Increment AC by one and place the result back in AC. Compare the result with zero, and if the condition specified by  $M$  is satisfied, take the next instruction from location  $E$  and continue sequential operation from there. If AC originally contained  $2^{35} - 1$ , set the Overflow and Carry 1 flags; if  $-1$ , set Carry 0 and Carry 1.

AOJ	Add One to AC but Do Not Jump	340
AOJL	Add One to AC and Jump if Less than Zero	341
AOJE	Add One to AC and Jump if Equal to Zero	342
AOJLE	Add One to AC and Jump if Less than or Equal to Zero	343

A0JA	Add One to AC and Jump Always	344
A0JGE	Add One to AC and Jump if Greater than or Equal to Zero	345
A0JN	Add One to AC and Jump if Not Equal to Zero	346
A0JG	Add One to AC and Jump if Greater than Zero	347

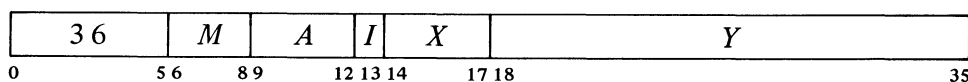
**AOS      Add One to Memory and Skip if Condition Satisfied**



Increment the contents of location *E* by one and place the result back in *E*. Compare the result with zero, and skip the next instruction in sequence if the condition specified by *M* is satisfied. If location *E* originally contained  $2^{35} - 1$ , set the Overflow and Carry 1 flags; if  $-1$ , set Carry 0 and Carry 1. If *A* is nonzero also place the result in AC.

AOS	Add One to Memory but Do Not Skip	350
AOSL	Add One to Memory and Skip if Less than Zero	351
AOSE	Add One to Memory and Skip if Equal to Zero	352
AOSLE	Add One to Memory and Skip if Less than or Equal to Zero	353
AOSA	Add One to Memory and Skip Always	354
AOSGE	Add One to Memory and Skip if Greater than or Equal to Zero	355
AOSN	Add One to Memory and Skip if Not Equal to Zero	356
AOSG	Add One to Memory and Skip if Greater than Zero	357

**SOJ      Subtract One from AC and Jump if Condition Satisfied**

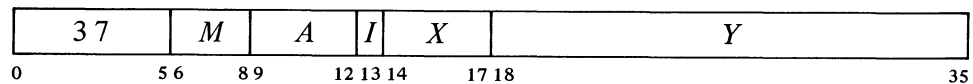


Decrement AC by one and place the result back in AC. Compare the result with zero, and if the condition specified by *M* is satisfied, take the next instruction from location *E* and continue sequential operation from there. If AC originally contained  $-2^{35}$ , set the Overflow and Carry 0 flags; if any other nonzero number, set Carry 0 and Carry 1.

SOJ	Subtract One from AC but Do Not Jump	360
SOJL	Subtract One from AC and Jump if Less than Zero	361
SOJE	Subtract One from AC and Jump if Equal to Zero	362
SOJLE	Subtract One from AC and Jump if Less than or Equal to Zero	363

SOJA	Subtract One from AC and Jump Always	364
SOJGE	Subtract One from AC and Jump if Greater than or Equal to Zero	365
SOJN	Subtract One from AC and Jump if Not Equal to Zero	366
SOJG	Subtract One from AC and Jump if Greater than Zero	367

**SOS Subtract One from Memory and Skip if Condition Satisfied**



Decrement the contents of location *E* by one and place the result back in *E*. Compare the result with zero, and skip the next instruction in sequence if the condition specified by *M* is satisfied. If location *E* originally contained  $-2^{35}$ , set the Overflow and Carry 0 flags; if any other nonzero number, set Carry 0 and Carry 1. If *A* is nonzero also place the result in AC.

SOS	Subtract One from Memory but Do Not Skip	370
SOSL	Subtract One from Memory and Skip if Less than Zero	371
SOSE	Subtract One from Memory and Skip if Equal to Zero	372
SOSLE	Subtract One from Memory and Skip if Less than or Equal to Zero	373
SOSA	Subtract One from Memory and Skip Always	374
SOSGE	Subtract One from Memory and Skip if Greater than or Equal to Zero	375
SOSN	Subtract One from Memory and Skip if Not Equal to Zero	376
SOSG	Subtract One from Memory and Skip if Greater than Zero	377

Some of these instructions are useful for determining the relative values of fixed and floating point numbers; others are convenient for controlling iterative processes by counting. AOSE is especially useful in an interlock procedure in a multiprocessor system. Suppose memory contains a routine that must be available to two processors but cannot be used by both at once. When one processor finishes the routine it sets location LOCK to  $-1$ . Either processor can then test the interlock and make it busy with no possibility of letting the other one in, as AOSE cannot be interrupted once it starts to modify the addressed location.

This procedure is invalid in the KA10 if the programmer

```

AOSE  LOCK      ;Skip to interlocked code only if
JRST  .-1       ;LOCK is zero after addition
:
:
:
SETOM LOCK      ;Unlock

```

is making use of the drum split feature (which is not used by any DEC equipment).

Since it takes several days to count to  $2^{36}$ , it is alright to keep testing the lock.

## 2.8 LOGICAL TESTING AND MODIFICATION

These eight instructions use a mask to modify and/or test selected bits in AC. The bits are those that correspond to 1s in the mask and they are referred to as the “masked bits”. The programmer chooses the mask, the way in which the masked bits are to be modified, and the condition the masked bits must satisfy to produce a skip.

The basic mnemonics are three letters beginning with T. The second letter selects the mask and the manner in which it is used.

<i>Mask</i>	<i>Letter</i>	<i>Effect</i>
Right	R	AC right is masked by <i>E</i> (AC is masked by the word 0, <i>E</i> )
Left	L	AC left is masked by <i>E</i> (AC is masked by the word <i>E</i> ,0)
Direct	D	AC is masked by the contents of location <i>E</i>
Swapped	S	AC is masked by the contents of location <i>E</i> with left and right halves interchanged

The third letter determines the way in which those bits selected by the mask are modified.

<i>Modification</i>	<i>Letter</i>	<i>Effect on AC</i>
No	N	None
Zeros	Z	Places 0s in all masked bit positions
Complement	C	Complements all masked bits
Ones	O	Places 1s in all masked bit positions

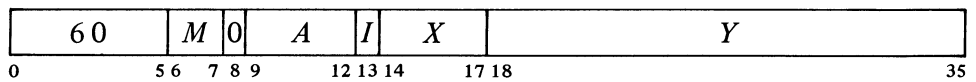
An additional letter may be appended to indicate the mode, which specifies the condition the masked bits must satisfy to produce a skip.

These mode names are consistent with those for arithmetic testing and derive from the test method, which ands AC with the mask and tests whether the result is equal to zero or is not equal to zero. The programmer may find it convenient to think of the modes as Every and Not Every: every masked bit is 0 or not every masked bit is 0.

<i>Mode</i>	<i>Suffix</i>	<i>Effect</i>
Never		Never skip
Equal	E	Skip if all masked bits equal 0
Always	A	Always skip
Not Equal	N	Skip if not all masked bits equal 0 (at least one bit is 1)

If the mnemonic has no suffix there is never any skip, and the instruction is a no-op if there is also no modification; an A suffix produces an unconditional skip – the skip always occurs regardless of the state of the masked bits. Note that the skip condition must be satisfied by the state of the masked bits *prior* to any modification called for by the instruction.

**TRN Test Right, No Modification, and Skip if Condition Satisfied**

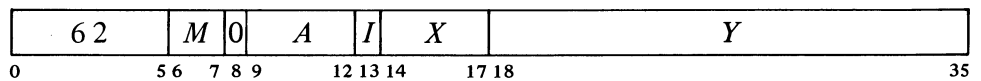


If the bits in AC right corresponding to 1s in *E* satisfy the condition specified by *M*, skip the next instruction in sequence. AC is unaffected.

TRN is a no-op in which *I*, *X* and *Y* are reserved for future use and should be zero (at present *E* is ignored).

TRN	Test Right, No Modification, but Do Not Skip	600
TRNE	Test Right, No Modification, and Skip if All Masked Bits Equal 0	602
TRNA	Test Right, No Modification, but Always Skip	604
TRNN	Test Right, No Modification, and Skip if Not All Masked Bits Equal 0	606

**TRZ Test Right, Zeros, and Skip if Condition Satisfied**

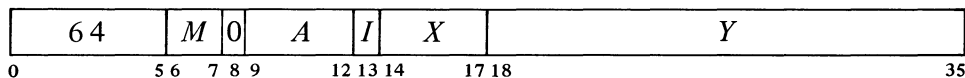


If the bits in AC right corresponding to 1s in *E* satisfy the condition specified by *M*, skip the next instruction in sequence. Change the masked AC bits to 0s; the rest of AC is unaffected.

TRZ	Test Right, Zeros, but Do Not Skip	620
TRZE	Test Right, Zeros, and Skip if All Masked Bits Equaled 0	622
TRZA	Test Right, Zeros, but Always Skip	624
TRZN	Test Right, Zeros, and Skip if Not All Masked Bits Equaled 0	626



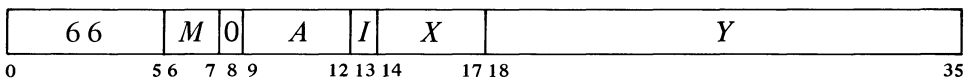
**TRC Test Right, Complement, and Skip if Condition Satisfied**



If the bits in AC right corresponding to 1s in *E* satisfy the condition specified by *M*, skip the next instruction in sequence. Complement the masked AC bits; the rest of AC is unaffected.

TRC	Test Right, Complement, but Do Not Skip	640
TRCE	Test Right, Complement, and Skip if All Masked Bits Equaled 0	642
TRCA	Test Right, Complement, but Always Skip	644
TRCN	Test Right, Complement, and Skip if Not All Masked Bits Equaled 0	646

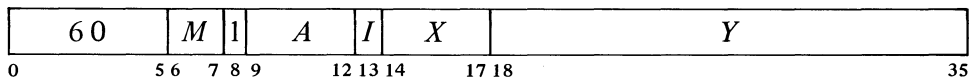
**TRO Test Right, Ones, and Skip if Condition Satisfied**



If the bits in AC right corresponding to 1s in *E* satisfy the condition specified by *M*, skip the next instruction in sequence. Change the masked AC bits to 1s; the rest of AC is unaffected.

TRO	Test Right, Ones, but Do Not Skip	660
TROE	Test Right, Ones, and Skip if All Masked Bits Equaled 0	662
TROA	Test Right, Ones, but Always Skip	664
TRON	Test Right, Ones, and Skip if Not All Masked Bits Equaled 0	666

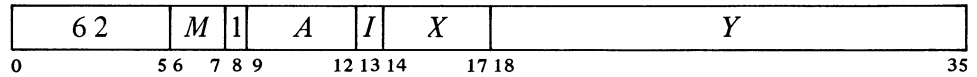
**TLN Test Left, No Modification, and Skip if Condition Satisfied**



If the bits in AC left corresponding to 1s in *E* satisfy the condition specified by *M*, skip the next instruction in sequence. AC is unaffected.

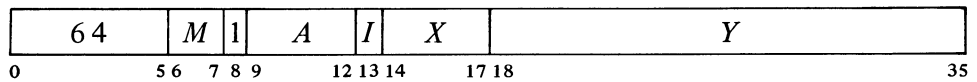
TLN	Test Left, No Modification, but Do Not Skip	601
TLNE	Test Left, No Modification, and Skip if All Masked Bits Equal 0	603
TLNA	Test Left, No Modification, but Always Skip	605
TLNN	Test Left, No Modification, and Skip if Not All Masked Bits Equal 0	607

TLN is a no-op in which *I*, *X* and *Y* are reserved for future use and should be zero (at present *E* is ignored).

**TLZ Test Left, Zeros, and Skip if Condition Satisfied**

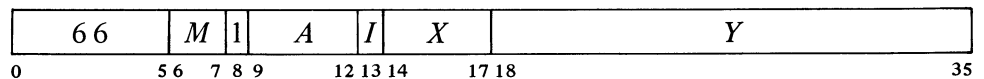
If the bits in AC left corresponding to 1s in *E* satisfy the condition specified by *M*, skip the next instruction in sequence. Change the masked AC bits to 0s; the rest of AC is unaffected.

<b>TLZ</b>	Test Left, Zeros, but Do Not Skip	621
<b>TLZE</b>	Test Left, Zeros, and Skip if All Masked Bits Equaled 0	623
<b>TLZA</b>	Test Left, Zeros, but Always Skip	625
<b>TLZN</b>	Test Left, Zeros, and Skip if Not All Masked Bits Equaled 0	627

**TLC Test Left, Complement, and Skip if Condition Satisfied**

If the bits in AC left corresponding to 1s in *E* satisfy the condition specified by *M*, skip the next instruction in sequence. Complement the masked AC bits; the rest of AC is unaffected.

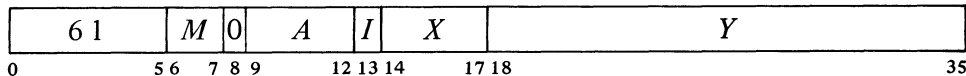
<b>TLC</b>	Test Left, Complement, but Do Not Skip	641
<b>TLCE</b>	Test Left, Complement, and Skip if All Masked Bits Equaled 0	643
<b>TLCA</b>	Test Left, Complement, but Always Skip	645
<b>TLCN</b>	Test Left, Complement, and Skip if Not All Masked Bits Equaled 0	647

**TLO Test Left, Ones, and Skip if Condition Satisfied**

If the bits in AC left corresponding to 1s in *E* satisfy the condition specified by *M*, skip the next instruction in sequence. Change the masked AC bits to 1s; the rest of AC is unaffected.

<b>TLO</b>	Test Left, Ones, but Do Not Skip	661
<b>TLOE</b>	Test Left, Ones, and Skip if All Masked Bits Equaled 0	663
<b>TLOA</b>	Test Left, Ones, but Always Skip	665
<b>TLON</b>	Test Left, Ones, and Skip if Not All Masked Bits Equaled 0	667

**TDN Test Direct, No Modification, and Skip if Condition Satisfied**

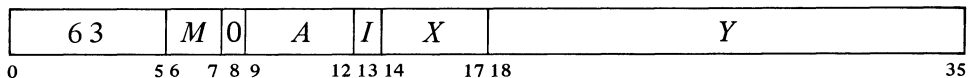


If the bits in AC corresponding to 1s in the contents of location *E* satisfy the condition specified by *M*, skip the next instruction in sequence. AC is unaffected.

TDN	Test Direct, No Modification, but Do Not Skip	610
TDNE	Test Direct, No Modification, and Skip if All Masked Bits Equal 0	612
TDNA	Test Direct, No Modification, but Always Skip	614
TDNN	Test Direct, No Modification, and Skip if Not All Masked Bits Equal 0	616

TDN is a no-op that references memory.

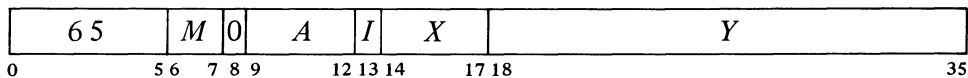
**TDZ Test Direct, Zeros, and Skip if Condition Satisfied**



If the bits in AC corresponding to 1s in the contents of location *E* satisfy the condition specified by *M*, skip the next instruction in sequence. Change the masked AC bits to 0s; the rest of AC is unaffected.

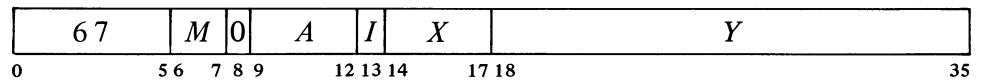
TDZ	Test Direct, Zeros, but Do Not Skip	630
TDZE	Test Direct, Zeros, and Skip if All Masked Bits Equaled 0	632
TDZA	Test Direct, Zeros, but Always Skip	634
TDZN	Test Direct, Zeros, and Skip if Not All Masked Bits Equaled 0	636

**TDC Test Direct, Complement, and Skip if Condition Satisfied**



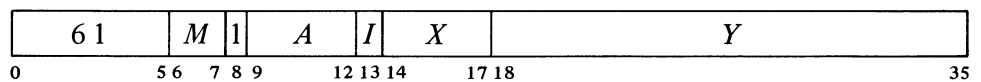
If the bits in AC corresponding to 1s in the contents of location *E* satisfy the condition specified by *M*, skip the next instruction in sequence. Complement the masked AC bits; the rest of AC is unaffected.

TDC	Test Direct, Complement, but Do Not Skip	650
TDCE	Test Direct, Complement, and Skip if All Masked Bits Equaled 0	652
TDCA	Test Direct, Complement, but Always Skip	654
TDCN	Test Direct, Complement, and Skip if Not All Masked Bits Equaled 0	656

**TDO Test Direct, Ones, and Skip if Condition Satisfied**

If the bits in AC corresponding to 1s in the contents of location *E* satisfy the condition specified by *M*, skip the next instruction in sequence. Change the masked AC bits to 1s; the rest of AC is unaffected.

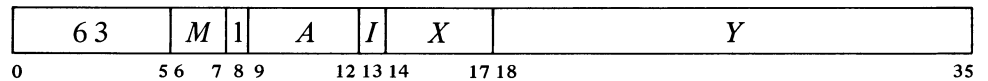
<b>TDO</b>	Test Direct, Ones, but Do Not Skip	670
<b>TDOE</b>	Test Direct, Ones, and Skip if All Masked Bits Equaled 0	672
<b>TDOA</b>	Test Direct, Ones, but Always Skip	674
<b>TDON</b>	Test Direct, Ones, and Skip if Not All Masked Bits Equaled 0	676

**TSN Test Swapped, No Modification, and Skip if Condition Satisfied**

If the bits in AC corresponding to 1s in the contents of location *E* with its left and right halves swapped satisfy the condition specified by *M*, skip the next instruction in sequence. AC is unaffected.

TSN is a no-op that references memory.

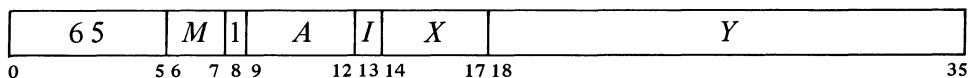
<b>TSN</b>	Test Swapped, No Modification, but Do Not Skip	611
<b>TSNE</b>	Test Swapped, No Modification, and Skip if All Masked Bits Equal 0	613
<b>TSNA</b>	Test Swapped, No Modification, but Always Skip	615
<b>TSNN</b>	Test Swapped, No Modification, and Skip if Not All Masked Bits Equal 0	617

**TSZ Test Swapped, Zeros, and Skip if Condition Satisfied**

If the bits in AC corresponding to 1s in the contents of location *E* with its left and right halves swapped satisfy the condition specified by *M*, skip the next instruction in sequence. Change the masked AC bits to 0s; the rest of AC is unaffected.

<b>TSZ</b>	Test Swapped, Zeros, but Do Not Skip	631
<b>TSZE</b>	Test Swapped, Zeros, and Skip if All Masked Bits Equaled 0	633
<b>TSZA</b>	Test Swapped, Zeros, but Always Skip	635
<b>TSZN</b>	Test Swapped, Zeros, and Skip if Not All Masked Bits Equaled 0	637

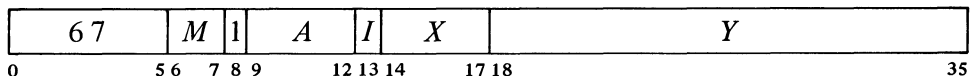
**TSC Test Swapped, Complement, and Skip if Condition Satisfied**



If the bits in AC corresponding to 1s in the contents of location *E* with its left and right halves swapped satisfy the condition specified by *M*, skip the next instruction in sequence. Complement the masked AC bits; the rest of AC is unaffected.

TSC	Test Swapped, Complement, but Do Not Skip	651
TSC	Test Swapped, Complement, and Skip if All Masked Bits Equaled 0	653
TSCA	Test Swapped, Complement, but Always Skip	655
TSCN	Test Swapped, Complement, and Skip if Not All Masked Bits Equaled 0	657

**TSO Test Swapped, Ones, and Skip if Condition Satisfied**



If the bits in AC corresponding to 1s in the contents of location *E* with its left and right halves swapped satisfy the condition specified by *M*, skip the next instruction in sequence. Change the masked AC bits to 1s; the rest of AC is unaffected.

TSO	Test Swapped, Ones, but Do Not Skip	671
TSOE	Test Swapped, Ones, and Skip if All Masked Bits Equaled 0	673
TSOA	Test Swapped, Ones, but Always Skip	675
TSON	Test Swapped, Ones, and Skip if Not All Masked Bits Equaled 0	677

With these instructions any bit throughout all of memory can be used as a program flag, although an ordinary memory location containing flags must be moved to an accumulator for testing or modification. The usual procedure, since locations 1-17 are addressable as index registers, is to use AC 0 as a register of flags (often addressed symbolically as F).

Unless one frequently tests flags in both halves of F simultaneously, it is generally most convenient to select bits by 1s right in the address part of the instruction word. A given bit selected by a half word mask *M* is then set by one of these:

TRO *F, M*      TLO *F, M*

and tested and cleared by one of these:

TRZE *F, M*      TRZN *F, M*      TLZE *F, M*      TLZN *F, M*

Suppose we wish to skip if both bits 34 and 35 are 1 in location L. The following suffices.

```
SETCM  F,L
TRNE   F,3
```

We can refer to a flag in a given bit position within a word as flag  $X$ , where  $X$  is a binary number containing a single 1 in the same bit position as the flag. This sequence determines whether flags  $X$  and  $Y$  in the right half of accumulator F are both on:

```
TRC    F, X + Y    ;Complement flags X and Y
TRCE   F, X + Y    ;Test both and restore original states
...    ;Do this if not both on
...    ;Skip to here if both on
```

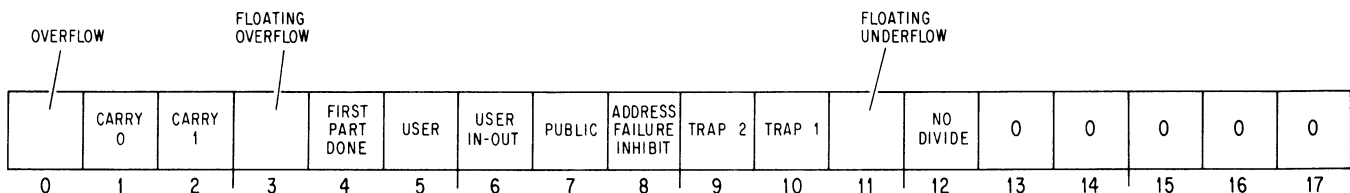
## 2.9 PROGRAM CONTROL

The program control class of instructions includes the unimplemented user operations [*discussed in the next section*] and the arithmetic and logical test instructions. Some instructions in this class are no-ops, as are a few of the instructions for performing logical operations. The most commonly used no-op is JFCL, which is discussed below. No-ops among the instructions previously discussed are SETA, SETAI, SETMM, CAI, CAM, JUMP, TRN, TLN, TDN, TSN. Of these, SETA, SETAI, CAI, JUMP, TRN and TLN do not use the calculated effective address to reference memory.

KA10 instruction codes 247 and 257 are reserved for instructions installed specially for a particular system. They execute as no-ops when run on a KA10 that contains no special hardware for them, but for program compatibility it is advised that they not be used regularly as no-ops.

The present section treats all program control instructions other than those mentioned above and in-out instructions that test input conditions [§2.12]. All but one of these are jumps, although the exception causes the processor to execute an instruction at an arbitrary location and may therefore be regarded as a jump with an immediate and automatic return. Also, all but two of the jumps are unconditional; one exception tests various flags, the other tests an accumulator.

Several of the jump instructions save the current contents of the program counter PC in the right half of an accumulator or memory location and save the states of various flags in the left half. The bits saved in the left half of



Note that nothing is stored in bits 13–17, so when the PC word is addressed indirectly it can produce neither indexing nor further indirect addressing.

this PC word in KI10 user mode are as shown here. In the KA10, bits 7–10 are not used. In KI10 executive mode, bit 6 receives the same flag although it has a different meaning, and bit 0 receives a different flag altogether [*see below*]. In either processor all unused bit positions are cleared.

The following lists the left PC-word bit positions that receive information and explains the meaning of the flags at the time they are saved. Certain

instructions can set up these flags to restore them to their original states following an interruption or to control specific situations. The explanations assume the flags reflect normal circumstances – not arbitrary rigging. In the following an *X* in a mnemonic indicates any letter (or none) that may appear in the given position to specify the mode, *eg* *ADDX* comprises *ADD*, *ADDI*, *ADDM*, *ADDB*.

*Bit* *Meaning of a 1 in the Bit*

- 0 Overflow – any of the following has occurred:
- A single instruction has set one of the carry flags (bits 1 and 2) without setting the other.
  - An *ASH* or *ASHC* has left shifted a 1 out of bit 1 in a positive number or a 0 out in a negative number.
  - An *MULX* has multiplied  $-2^{35}$  by itself (product  $2^{70}$ ).
  - An *IMULX* has multiplied two numbers with product  $\geq 2^{35}$  or  $< -2^{35}$ .
  - An *FIX* or *FIXR* has fetched an operand with exponent  $> 35$ . Floating Overflow has been set (bit 3).
  - No Divide has been set (bit 12).
- 1 Carry 0 – if set without Carry 1 (bit 2) being set, causes Overflow to be set and indicates that one of the following has occurred:
- An *ADDX* has added two negative numbers with sum  $< -2^{35}$ .
  - An *SUBX* has subtracted a positive number from a negative number with difference  $< -2^{35}$ .
  - An *SOJX* or *SOSX* has decremented  $-2^{35}$ .
- But if set with Carry 1, indicates that one of these nonoverflow events has occurred:
- In an *ADDX* both summands were negative, or their signs differed and their magnitudes were equal or the positive one was the greater in magnitude.
  - In an *SUBX* the signs of the operands were the same and AC was the greater or the two were equal, or the signs of the operands differed and AC was negative.
  - An *AOJX* or *AOSX* has incremented  $-1$ .
  - An *SOJX* or *SOSX* has decremented a nonzero number other than  $-2^{35}$ .
  - An *MOVNX* has negated zero.
- 2 Carry 1 – if set without Carry 0 (bit 1) being set, causes Overflow to be set and indicates that one of the following has occurred:
- An *ADDX* has added two positive numbers with sum  $\geq 2^{35}$ .
  - An *SUBX* has subtracted a negative number from a positive number with difference  $\geq 2^{35}$ .

In user mode, bit 0 reflects the state of Overflow. But when the flags are saved in KI10 executive mode, bit 0 represents the Disable Bypass flag, which the Monitor uses to control certain aspects of the execution of an instruction by an executive XCT [see below and §2.15]. Although these are two separate flags that are read in different circumstances, when a PC word is used to restore or set up the flags, bit 0 conditions both of them.

Remember [§2.5], overflow is determined directly from the carries, not from the flags. The carry flags give meaningful information only if no more than one instruction that can set them occurs between clearing and reading them.

An AOJX or AOSX has incremented  $2^{35} - 1$ .

An MOVNX or MOVMX has negated  $-2^{35}$ .

But if set with Carry 0, indicates that one of the nonoverflow events listed under Carry 0 has occurred.

- 3 Floating Overflow – any of the following has set Overflow:
  - In a floating point instruction, the exponent of the result was  $> 127$ .
  - DMOVNM or DFN, the exponent of the result was  $> 127$ .
  - Floating Underflow (bit 11) has been set.
  - No Divide (bit 12) has been set in an FDVX, FDVRX or DFDV.
- 4 First Part Done – the processor is responding to a priority interrupt between the parts of a two-part instruction or to a page failure in the second part. A 1 in this bit indicates that the first part has been completed, and this fact should be taken into account when the processor restarts the instruction at the beginning upon the return to the interrupted program. *Eg* if an ILDB or IDPB is interrupted after the processing of the pointer but before the processing of the byte, the pointer now points not to the last byte, but rather to the byte that should be handled at the return [§2.13]. Thus when the processor restarts the instruction, it must retrieve the pointer but *not* increment it.
  - Besides indicating a priority interrupt in the middle of a byte instruction, the KI10 First Part Done indicates a page failure in the processing of a byte, in the transfer of the second (low order) word in a DMOVEM or DMOVNM, or in a noninterrupt data IO instruction that results from a block IO instruction (following the processing of the pointer [§2.12]).
- 5 User – the processor is in user mode [§§2.15, 2.16].
- 6 User In-out – even with the processor in user mode, there are no instruction restrictions (but memory restrictions still apply).
- 7 Public (KI10 only) – the last instruction performed was fetched from a public area of memory, *ie* the processor is in user mode public or executive mode supervisor.
- 8 Address Failure Inhibit (KI10 only) – an address failure cannot occur during the next instruction [§2.15].
- 9 Trap 2 (KI10 only) – if bit 10 is not also set, pushdown overflow has occurred. Unless the executive paging system is disabled, the setting of this flag immediately causes a trap as explained at the end of this section. At present, bits 9 and 10 cannot be set together by any hardware condition.
- 10 Trap 1 (KI10 only) – if bit 9 is not also set, arithmetic overflow has occurred. Unless the executive paging system is disabled, the setting of this flag immediately causes a trap as explained at the end of this section. At present, bits 9 and 10 cannot be set together by any hardware condition.

Floating point instructions that cannot overflow are FLTR, DMOVN, DMOVNM and DFN.

Although this flag is set upon completion of the first part of every interruptable two-part instruction, it is seldom relevant to the programmer as it is always cleared by the completion of the second part. The flag is seen only in an interruption, and its effect on the repeated first part is automatic provided only that it is properly restored at the return.

In the KA10, User In-out is applicable only to user mode [§2.16]. In the KI10 this flag has the stated effect when the processor is in user mode, but is used in executive mode to control certain aspects of the execution of an instruction by an executive XCT [see below and §2.15].



11 Floating Underflow – in a floating point instruction, the exponent of the result was  $< -128$  and Overflow and Floating Overflow have been set.

12 No Divide – any of the following has set Overflow:

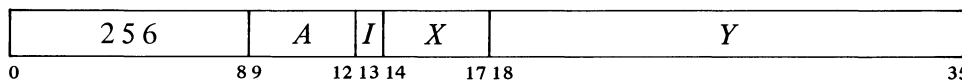
In a DIVX the dividend was greater than or equal to the divisor.

In an IDIVX the divisor was zero, or the dividend was  $-2^{35}$  and the divisor was  $\pm 1$ .

In an FDVX, FDVRX or DFDV the divisor was zero, or the dividend fraction was greater than or equal to twice the divisor fraction in magnitude; in either case Floating Overflow has been set.

If normalized operands are used, only a zero divisor can cause floating division to fail.

**XCT          Execute**



In user mode or in the KA10, execute the contents of location *E* as an instruction. Any instruction may be executed, including another XCT. If an XCT executes a skip instruction, the skip is relative to the location of the XCT (the first XCT if there are several in a chain). If an XCT executes a jump, program flow is altered as specified by the jump (no matter how many XCTs precede a jump instruction, when PC is saved it contains an address one greater than the location of the first XCT in the chain).

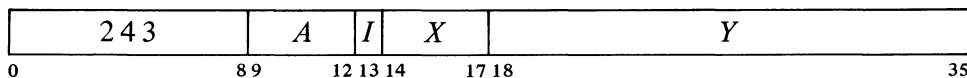
In KI10 executive mode this instruction performs as stated only when *A* is zero. Nonzero *A* results in a so called “executive XCT”, whose ramifications are far more widespread than indicated here [for details refer to §2.15].

In user mode and in the KA10, the *A* portion of this instruction is ignored. It should then be zero for compatibility with KI10 executive mode and possible future use even in user mode.

*CAUTION*

In concealed or kernel mode, an XCT that executes an instruction in a public page places the processor in public or supervisor mode. Hence unless the executed instruction changes PC to a public area, the instruction following the XCT must be a valid entry point back into the concealed area or a page failure, in particular a proprietary violation, will result. A valid entry point is one containing a particular form of the JRST instruction described below.

**JFFO          Jump if Find First One**



If AC contains zero, clear AC *A*+1 and go on to the next instruction in sequence.

If AC is not zero, count the number of leading 0s in it (0s to the left of the leftmost 1), and place the count in AC *A*+1. Take the next instruction from location *E* and continue sequential operation from there.

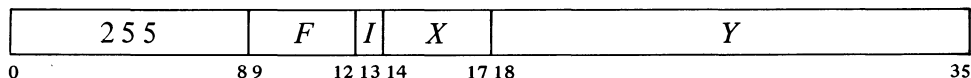
In either case AC is unaffected, the original contents of AC *A*+1 are lost.

Note that when AC is negative, the second accumulator is cleared, just as it would be if AC were zero.

To left-normalize a positive integer in AC: ▲

JFFO AC, +1  
LSH AC, -1(AC+1)

**JFCL          Jump on Flag and Clear**



If any flag specified by *F* is set, clear it and take the next instruction from

location *E*, continuing sequential operation from there. Bits 9–12 are programmed as follows.

<i>Bit</i>	<i>Flag Selected by a 1</i>
9	Overflow
10	Carry 0
11	Carry 1
12	Floating Overflow

This instruction can be used simply to clear the selected flags by having the jump address point to the next consecutive location, as in

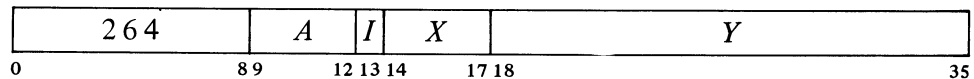
JFCL 17, +1

which clears all four flags without disrupting the normal program sequence. A JFCL that selects no flag is the fastest no-op as it neither fetches nor stores an operand, and bits 18–35 of the instruction word can be used to store information.

To select one or a combination of these flags (which are among those described above) the programmer can specify the equivalent of an AC address that places 1s in the appropriate bits, but MACRO recognizes mnemonics for some of the 13-bit instruction codes (bits 0–12).

JFCL	JFCL 0,	No-op	25500
JOV	JFCL 10,	Jump on Overflow	25540
JCRY0	JFCL 4,	Jump on Carry 0	25520
JCRY1	JFCL 2,	Jump on Carry 1	25510
JCRY	JFCL 6,	Jump on Carry 0 or 1	25530
JFOV	JFCL 1,	Jump on Floating Overflow	25504

### JSR            Jump to Subroutine

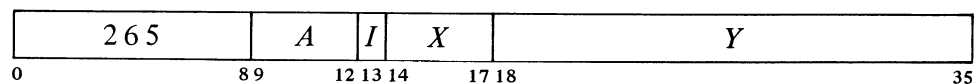


The *A* portion of this instruction is reserved for future use and should be zero (at present it is ignored).

Place the current contents of the flags (as described above) in the left half of location *E* and the contents of PC in the right half (at this time PC contains an address one greater than the location of the JSR instruction). Take the next instruction from location *E* + 1 and continue sequential operation from there. The flags are unaffected except First Part Done, Address Failure Inhibit, and the trap flags, which are cleared.

While the processor is in user mode, if this instruction is executed as an interrupt instruction or by a KA10 MUUO, bit 5 of the PC word stored is 1 and the processor leaves user mode, clearing Public. (In the KI10 an interrupt that is not dismissed automatically returns control to kernel mode.)

### JSP            Jump and Save PC



Place the current contents of the flags (as described above) in AC left and



no public program can clear Public for itself. As an example, setting First Part Done prevents incrementing in the next ILDB, IDPB or noninterrupt KI10 block IO instruction provided there is no intervening JSR, JSP or PUSHJ. Note that if overflow traps are enabled, setting a trap flag immediately causes one.

by the Monitor, *ie* if User is clear. A 1 in bit 7 sets Public, but a 0 clears it only if the JRST is being performed in executive mode with a 1 in bit 5 (*ie* User is being set). These conditions imply that the processor is entering user mode: hence the user cannot enter concealed mode by clearing Public; and although the supervisor can place the processor in user mode concealed, it cannot use this procedure to enter kernel mode.

12 *KA10*. Enter User mode. The user program starts at relocated location *E*.

*KI10*. The instruction is simply a jump except when fetched from a nonpublic area, in which case it clears Public. In other words a location containing a JRST 1, is a valid entry to a nonpublic area and the instruction places the processor in concealed or kernel mode.

To produce one or a combination of these functions the programmer can specify the equivalent of an AC address that places 1s in the appropriate bits, but MACRO recognizes mnemonics for the most important 13-bit instruction codes (bits 0–12).

JRST	JRST 0,	Jump	25400
	JRST 10,	Jump and Restore Interrupt Channel	25440
HALT	JRST 4,	Halt	25420
JRSTF	JRST 2,	Jump and Restore Flags	25410
PORTAL	JRST 1,	Allow Nonpublic Entry (KI10) Jump to User Program (KA10)	25404
JEN	JRST 12,	Jump and Enable	25450

JEN completes an interrupt by restoring the channel and restoring the flags for the interrupted program.

In a JRSTF or JEN the flags are restored from bits 0–12 of the final word retrieved in the effective address calculation; hence any JRST with a 1 in bit 11 must use indirect addressing or indexing, which takes extra time. If the PC word was stored in AC (as in a JSP), a common procedure is to use AC to index a zero address (*eg*, JRSTF (AC)), so its right half becomes the effective (jump) address. If the PC word was stored in core (as in a JSR), one must address it indirectly (remember, bits 13–17 of the PC word are clear, so again its right half is the effective address). A JRSTF (AC) is considerably faster than a JRSTF @PCWORD.

#### CAUTION

Giving a JRSTF or JEN without indexing or indirect addressing restores the flags from the instruction code itself.

While the KA10 is in user mode, if this instruction is executed as an interrupt instruction or by an MUUO, bit 5 of the PC word stored is 1 and the processor leaves user mode.



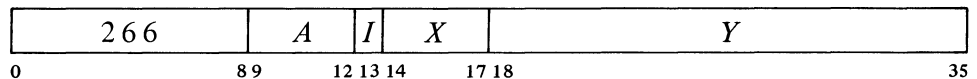
The subroutine can use *T* as a byte pointer which already addresses the first word of data. For the print routine, characters are loaded into another accumulator *CH*.

```

PRINT:  HRLI    T,440700    ;Initialize left half of pointer
        ILDB   CH,T        ;Increment pointer and load byte
        JUMPE  CH,1(T)     ;Upon reaching zero character return
                                ;to one beyond last data word
        :
        :
        JRST   PRINT+1    ;Get next character

```

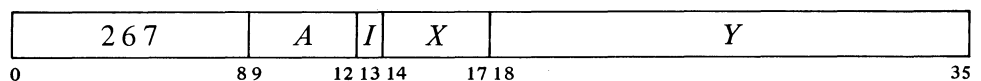
### JSA            Jump and Save AC



Place *AC* in location *E*, the effective address *E* in *AC* left, and the contents of *PC* in *AC* right (at this time *PC* contains an address one greater than the location of the *JSA* instruction). Take the next instruction from location *E* + 1 and continue sequential operation from there. The original contents of *E* are lost.

While the *KA10* is in user mode, if this instruction is executed as an interrupt instruction or by an *MUO*, bit 5 of the *PC* word stored is 1 and the processor leaves user mode.

### JRA            Jump and Restore AC



Place the contents of the location addressed by *AC* left into *AC*. Take the next instruction from location *E* and continue sequential operation from there.

A *JSA* combines advantages of the *JSR* and *JSP*. *JSA* does modify memory, but it saves *PC* in an accumulator without losing its previous contents (at a cost of not saving the flags). It is thus convenient for multiple-entry subroutines. In a subroutine called by a *JSR*, the returning *JRST* must refer to the (single) entry point. Since a *JRA* can retrieve the original *PC* by addressing *AC* as an index register, it is independent of any entry point

without tying up an accumulator to the extent a JSP would.

The accumulator contents saved by a JSA are restored by a JRA paired with it despite intervening JSA-JRA pairs. Hence these instructions are especially useful for nesting subroutines, as shown by this example.

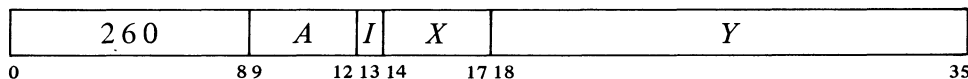
```

      :                               ;Main program
      :
      JSA   17,S1   ;Call to first subroutine (A)
      :
S1:    0           ;First subroutine starts here
      :
      JSA   17,S2   ;Call to second subroutine (B)
      :
      JRA   17,(17) ;Return to A + 1 in main program
S2:    0           ;Second subroutine starts here
      :
      JSA   17,S3   ;Call to third subroutine (C)
      :
      JRA   17,(17) ;Return to B + 1 in first subroutine
S3:    0           ;Third subroutine starts here
      :
      JRA   17,(17) ;Return to C + 1 in second subroutine

```

To call the next deeper subroutine at any level, a JSA places  $E$  and PC in the left and right of AC 17, saves the previous contents of AC 17 in  $E$  (the first subroutine location), and jumps to  $E + 1$ . To return to the next higher level, a JRA restores the previous contents of AC 17 from the location addressed by AC 17 left (the first subroutine location) and jumps to the location addressed by AC 17 right (the location following the JSA in the higher subroutine). If  $N$  lines of data for the next subroutine follow a JSA, the return to the location following the data is made by giving a JRA 17, $N$ (17).

### **PUSHJ      Push Down and Jump**



Add one to each half of AC and place the result back in AC. If the addition causes the count in AC left to reach zero, set the Pushdown Overflow flag in the KA10, set the Trap 2 flag in the KI10. Then place the current contents of the flags (as described above) in the left half of the location now addressed by AC right and the contents of PC in the right half of that location (at this time PC contains an address one greater than the location of the PUSHJ instruction). Take the next instruction from location  $E$  and continue sequential operation from there.

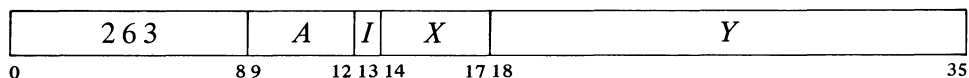
In the KI10 a PUSHJ executed as an interrupt instruction cannot set Trap 2.

The flags are unaffected except First Part Done, Address Failure Inhibit, and the trap flags, which are cleared. However, pushdown overflow overrides the Trap 2 clear, so if the list overflows, Trap 2 sets and the KI10 traps instead of jumping. The original contents of the location added to the list are lost.

Note: The KA10 increments the two halves of AC by adding  $1\ 000001_8$  to the entire register. In the KI10 the two halves are handled independently.

While the processor is in user mode, if this instruction is executed as an interrupt instruction or by a KA10 MUUO, bit 5 of the PC word stored is 1 and the processor leaves user mode, clearing Public. (In the KI10 an interrupt that is not dismissed automatically returns control to kernel mode.)

### POPJ Pop Up and Jump



Subtract one from each half of AC and place the result back in AC. If the subtraction causes the count in AC left to reach  $-1$ , set the Pushdown Overflow flag in the KA10, set the Trap 2 flag in the KI10. Take the next instruction from the location addressed by the right half of the location that was addressed by AC right *prior* to the decrementing, and continue sequential operation from there.

Note: The KA10 decrements the two halves of AC by subtracting  $1\ 000001_8$  from the entire register. In the KI10 the two halves are handled independently.

*I*, *X* and *Y* are reserved for future use and should be zero (at present *E* is ignored). In the KI10 a POPJ executed as an interrupt instruction cannot set Trap 2.

#### CAUTION

The jump is completed before the processor responds to overflow, whether by trap or interrupt. Hence it is impossible to determine the location of the POPJ that caused the overflow.

The address of the top item in the pushdown list is kept in the right half of the pointer in AC, and the program can keep a control count in the left half. In the KA10, incrementing and decrementing both halves of AC together is effected by adding and subtracting  $1\ 000001_8$ . Hence a count of  $-2$  in AC left is increased to zero if  $2^{18} - 1$  is incremented in AC right, and conversely, 1 in AC left is decreased to  $-1$  if zero is decremented in AC right.

Since the pushdown list is independent of the subroutine called, PUSHJ-POPJ can be used like JSA-JRA for multiple entries. Moreover, ordering by level is inherent in the structure of a pushdown list [§2.2], so paired PUSHJ-POPJ instructions are excellent for nesting subroutines: there can be any number of subroutines at any level, each with more subroutines nested within it. Recursive subroutines are also possible.

Unlike JSA-JRA, the pushdown instructions tie up an accumulator, but the usual procedure is to keep both data and jump addresses in a single list so only one AC is required for the most complex pushdown operations. The programmer must keep track of whether a given entry in the list is data or a PC word; in other words, every item inserted by a PUSH should be removed by a POP, and every PUSHJ should be matched by a POPJ. If flag



restoration is desired, the returning

POPJ P,

can be replaced by

POP P,AC  
JRSTF (AC)

which requires another accumulator. If the flags are not important, data may be stored in the left halves of the PC words in the stack, reducing the required pushdown depth.

By trapping or checking overflow and keeping a control count in AC left, the programmer can set a limit to the size of the list by starting the count negative, or he can prevent the program from extracting more items than there are in the list by starting the count at zero, but he cannot do both at once. If only jump addresses are kept in the list, the first procedure limits the depth of nesting. A technique to catch extra POPJs is to put a PC word addressing an error routine at the bottom of the list,

### Overflow Trapping

In the performance of a program there are many events that cannot be foreseen and whose occurrence requires special action by the program. There are instructions that test for the conditions produced by such events, but in say a long string of computations, it would be both cumbersome and time consuming to test for overflow at every step. It is far better simply to allow an event such as overflow to break right into the normal program sequence.

For situations of this nature, various internal conditions can act through the priority interrupt system. However the processor also has a trapping mechanism that allows conditions due directly to the program, and which are often permitted to happen as a matter of course, to interrupt the program sequence without recourse to the interrupt system. In some cases, traps are used to handle the restrictions that play a role in program and memory management [*as explained in later sections*], but here we are concerned specifically with action by the processor in response to overflow.

Overflow produced by an interrupt instruction cannot be detected. In any other circumstances, an instruction in which an arithmetic overflow condition occurs sets Overflow and Trap 1, and an instruction in which a pushdown overflow occurs sets Trap 2. At the completion of an instruction in which either trap flag is set, rather than going on to the next instruction as specified by PC, the processor instead executes an instruction taken from a particular location in the process table for the program (user or executive). The location as a function of the trap flags set is as follows.

<i>Trap Flags Set</i>	<i>Trap Type</i>	<i>Trap Number</i>	<i>Location</i>
Trap 1 only	Arithmetic overflow	1	421
Trap 2 only	Pushdown overflow	2	422
Trap 1 and 2	Not used by hardware	3	423

### NOTE

This feature is not available in the KA10. That processor is limited to the use of internal conditions that can act through the priority interrupt system [§2.14].

Note that it is the overflow condition that sets Trap 1 — not the state of the Overflow flag. Hence an overflow is trapped even if Overflow is already set.

Note also that the trap flags have no effect at all when executive paging is disabled [§2.15].

A trap can be produced artificially simply by setting up the trap flags with a JRSTF or MUUO. In this way the program can also use trap number

3, which at present cannot result from any hardware-detected condition (it is reserved for future use by DEC).

The location of the instruction that caused the overflow can be determined from PC unless the instruction jumped, in which case its location can be determined only for a PUSHJ, from the stack entry.

An arithmetic instruction that overflows on every iteration produces an infinite loop if used as a trap instruction for arithmetic overflow. A pushdown instruction in a pushdown overflow trap can overflow only once. (The memory allocated to a pushdown stack should have at least one extra location to handle this case — two extras if the program and the trap both use the same pointer.)

These are convenience mnemonics that mean nothing to the assembler. UUOs are also sometimes called “programmed operators”.

A trap instruction is executed in the same address space as the instruction that caused it. Overflow in a user instruction traps to a location in the user process table, and any addresses used in the instruction in that location are interpreted in the user address space. Thus a user program can handle its own traps, *eg* by requesting the Monitor to place a PUSHJ to a user routine in the trap location. An MUUO must be used if the Monitor is to handle a user-caused trap.

The trap instruction (the final instruction in an XCT and/or LUUO string) clears the trap flags, so the processor returns to the interrupted program unless the trap instruction changes PC. Thus the trap instruction can be a no-op (which ignores the trap), a skip, a jump, or anything else. However, should the trap instruction itself set a trap flag (not necessarily the same one), a second trap occurs.

An interrupt can occur between an instruction that overflows and the trap instruction, but the latter will be performed correctly upon the return provided the interrupt is dismissed automatically or the interrupt routine restores the flags properly. If a single instruction causes both overflow and a page failure, the latter has preference; but the overflow trap will be taken care of after the offending instruction has been restarted and completed successfully. A trap instruction that causes a page failure does not clear the trap flags; hence after the page failure is taken care of, the trap instruction will correctly handle the trap when it is restarted.

## 2.10 UNIMPLEMENTED OPERATIONS

Codes not assigned as specific instructions act as unimplemented operations, wherein the word given as an instruction is trapped and must be interpreted by a routine included for this purpose by the programmer. Codes in the range 001–077 are unimplemented user operations, or UUOs. Half of these (001–037) are for the local use of the user or Monitor (LUUOs); the other half (040–077) are set aside for user communication with the Monitor (MUUOs) and are interpreted by it (although they may be used by the Monitor as well). Codes 100 and above that are not used for instructions are regarded as the “unassigned codes”; 000 is not regarded as a legal code at all. Instructions that violate the instruction restrictions act in the same manner as MUUOs.

### Local Unimplemented User Operation

001–037	A	I	X	Y
0	8 9	12 13 14	17 18	35

Store the instruction code, *A* and the effective address *E* in bits 0–8, 9–12 and 18–35 respectively of location 40; clear bits 13–17. Execute the instruction contained in location 41. The original contents of location 40 are lost.

Every LUUO uses some pair of locations numbered 40 and 41, but which such pair depends upon the circumstances. An LUUO in a user program uses relocated locations 40 and 41 and is thus entirely a part of and under control of the user program. An LUUO in KA10 executive mode uses unrelocated locations. In KI10 executive mode an LUUO uses locations 40 and 41 in the executive process table.

The actions of MUUOs and unassigned codes depend to a considerable degree on the processor. All use at least two consecutive locations, where the first receives the information specified above for an LUUO (in the KI10 a third nonconsecutive location is also used). The unassigned codes are included so that the Monitor steps in when a user gives an incorrect code. The code 000 acts in exactly the same way as an MUUO but is not a standard communication code: it is included so that control returns to the Monitor should a user program wipe itself out.

**KI10.** MUUOs and unassigned codes in user or executive mode act in exactly the same way. They store the information specified above for an LUUO in location 424 of the user process table, save the flags and PC (the current PC word) in location 425, set up the flags and PC according to a new PC word taken from a third location, and restart the processor in normal sequence at the location then addressed by PC. In the PC word saved in location 425, bit 0 may represent either Overflow or Disable Bypass depending upon the mode the processor is in when the MUUO is given. If the MUUO is given directly by the program, the address in the right half of the PC word saved is one greater than the location of the MUUO; otherwise it depends upon the circumstances in which the MUUO is executed. The new PC word can be taken from among the eight locations in the user process table listed here depending upon the mode at the time the MUUO is given, and whether or not it is executed as the result of a trap (page failure or overflow).

<i>Mode</i>	<i>Execution</i>	<i>Location</i>
Kernel	No trap	430
Kernel	Trap	431
Supervisor	No trap	432
Supervisor	Trap	433
Concealed	No trap	434
Concealed	Trap	435
Public	No trap	436
Public	Trap	437

There are no restrictions on the manner in which the new PC word of an MUUO can set up the flags. It can switch the processor from any mode to any other. A 1 in bit 0 sets both Overflow and Disable Bypass; a 0 clears both. Hence bit 0 should be adjusted to produce the desired state in the flag that is relevant to the mode the processor is entering.

If a single memory serves as memory number 0 for two KA10 processors, the second (with the trap offset) uses unrelocated 140-141 and 160-161 respectively for each instance in which 40-41 and 60-61 are given here. The offset does not apply to user LUUOs as it is assumed the Monitor would relocate these to different physical blocks.

The unassigned codes are 100-107, 114-117, 123 and 247.

Note that even in a dedicated system, the program must still define a user process table.

Note that unless executive paging is disabled, setting a trap flag immediately causes a trap.

Note that in executive mode, LUUOs and MUUOs act identically.

Codes 247 and 257, although not assigned as specific instructions, are nonetheless not regarded as "unassigned" codes. They execute as no-ops unless implemented by special hardware.

**KA10.** MUUOs and unassigned codes, regardless of mode, perform exactly the operations given above for an LUUO with the exception that MUUOs use unrelocated 40-41 and unassigned codes use unrelocated 60-61 (140-141 and 160-161 for a second processor). The unassigned codes are 100-127. The codes 130-177, which are the floating point and byte manipulation instructions, are equivalent to the unassigned codes if unimplemented, *ie* if the hardware for them is not included. In this case all codes 100-177 trap to unrelocated 60-61.

The important point is that an MUUO or unassigned code results in executing an instruction in an unrelocated location, *ie* an instruction under the control of the Monitor. This would most likely be a jump that leaves user mode, saves the PC word and enters a routine to interpret the MUUO configuration. In the instruction descriptions, any reference to events resulting from execution by an MUUO should be taken to include the unassigned and illegal codes as well.

## 2.11 PROGRAMMING EXAMPLES

Before continuing to input-output and related subjects, let us consider some simple programs that demonstrate the use of a variety of the instructions described thus far.

### Processor Identification

The instruction repertoires of the KI10, the KA10, and the 166 processor used in the PDP-6 are very similar, and most programs require no changes to run on any of them. Because of minor differences and the fact that some instructions are not available on the earlier machines, a program that is to be compatible with all three should have some way of distinguishing which machine it is running on. This simple test suffices.

JFCL	17,+.1	;Clear flags
JRST	.+1	;Change PC
JFCL	1,PDP6	;PDP-6 has PC Change flag
MOVNI	AC,1	;Others do not, make AC all 1s
AOBJN	AC,+.1	;Increment both halves
JUMPN	AC,KA10	;KA10 if AC = 1000000
JRST	KI10	;KI10 if AC = 0 (no carry between halves)

### Parity

Parity procedures are used regularly to check the accuracy of stored information. Parity generation and checking is generally handled automatically by memory and high speed, block-oriented peripheral devices, but must be handled by the program for character-oriented devices. Consider 8-bit characters, for which the program carries out two procedures: for output it

generates a parity bit from seven data bits to produce an 8-bit character with parity; following input it checks the parity of the eight bits received. In either case however, the program can simply find the parity of an 8-bit character, by regarding the seven output data bits as eight including an irrelevant extra bit. The two procedures then differ only in the final action. In the first case the program uses the result to adjust the eighth bit for correct parity, whereas in the second it checks the result for an indication of error.

Assuming the character is right-justified in accumulator A, the simplest and quickest procedure would be to use A to index an XCT into a table, each of whose locations contains an instruction that adjusts the parity for output or jumps to a routine for erroneous input. This procedure would normally be unacceptable because of the very large memory requirements. However the table can be reduced to sixteen entries without excessive loss in speed, by exclusive oring the left and right halves of the character and indexing on the result (parity is invariant under the exclusive OR function, which essentially disposes of pairs of 1s). This example, which uses a second accumulator T for character manipulation, requires six memory references to generate odd parity.

```

PARITY:  MOVEI  T,(A)      ;Copy character in T
         LSH   T,-4       ;Line up halves
         XORI  T,(A)      ;Reduce paritywise to 4 bits
         ANDI  T,17       ;Wipe out unwanted bits
         XCT   PARTAB(T)  ;Execute indicated table item
         POPJ  P,

PARTAB:  XORI  A,200      ;0 - change high bit
         JFCL                ;1 - no-op
         JFCL                ;2
         XORI  A,200      ;3
         JFCL                ;4
         XORI  A,200      ;5
         XORI  A,200      ;6
         JFCL                ;7
         JFCL                ;10
         XORI  A,200      ;11
         XORI  A,200      ;12
         JFCL                ;13
         XORI  A,200      ;14
         JFCL                ;15
         JFCL                ;16
         XORI  A,200      ;17

```

We assume the rest of A, outside the character, is clear, as it would be were the character placed in A by a load-byte instruction or a DATAI. The next two examples, however, work even if the rest of A is not clear.

Numbers of memory references and locations given do not include those for the POPJ, which we will regard as subroutine overhead. Similarly every example also requires that the program give a PUSHJ to get to the subroutine.

To handle even parity, interchange the JFCLs and XORIs in the table, or change the MOVEI T,(A) to MOVEI T,200(A).

The next example does exactly the same thing but substitutes a little more computation for use of a table. In other words it takes a little more time (7½ memory references average) but less than half the memory.

```

PARITY:  MOVEI  T,200(A)      ;Copy character with high bit comple-
          LSH   T,-4          ;mented, then fold copy into 4 bits
          XORI  T,(A)         ;with opposite parity
          TRCE  T,14          ;Are left two both 0?
          TRNN  T,14          ;Or both 1?
          XORI  A,200         ;Yes, change high bit
          TRCE  T,3           ;Are right two both 0?
          TRNN  T,3           ;Or both 1?
          XORI  A,200         ;Yes, change for even, restore for odd
          POPJ  P,

```

For even parity change the address in the MOVEI from 200 to 0.

Finally let us consider the extreme of substituting computation for memory. Starting with the character *abcdefgh* right-justified in A, we first copy it in T and then duplicate it twice to the left producing

*abc def gha bcd efg hab cde fgh*

where the bits (in positions 12–35) are grouped corresponding to the octal digits in the word. Anding this with

001 001 001 001 001 001 001 001

retains only the least significant bit in each 3-bit set, so we can represent the result by

*cfadgbeh*

where each letter represents an octal digit having the same value (0 or 1) as the bit originally represented by the same letter. Multiplying this by  $11111111_8$  generates the following partial products:

```

          c f a d g b e h
         c f a d g b e h
        c f a d g b e h
       c f a d g b e h
      c f a d g b e h
     c f a d g b e h
    c f a d g b e h
   c f a d g b e h
  c f a d g b e h
 c f a d g b e h

```

Since any digit is at most 1, there can be no carry out of any column with fewer than eight digits unless there is a carry into it. Hence the octal digit produced by summing the center column (the one containing all the bits of the character) is even or odd as the sum of the bits is even or odd. Thus its least significant bit (bit 14 of the low order word in the product) is the parity of the character, 0 if even, 1 if odd.

The above may seem a very complicated procedure to do something trivial, but it is effected by this quite simple sequence:

```

PARITY:  MOVEI  T,(A)        ;Copy in T
          IMULI T,200401     ;Duplicate twice
          AND   T,ONES       ;Pick LSBs

```

```

IMUL   T,ONES   ;Generate product
TLNN   T,10     ;Is bit 14 odd?
XORI   A,200    ;No, change parity
POPJ   P,
:
:

```

ONES: 11111111

This procedure uses a minimum of both memory references and memory space, but takes considerably more time because the instructions themselves are slow.

The following table shows the trade-off of memory references against memory space for the above four procedures. The time is proportional to the number of references except in case 4.

	<i>References</i>	<i>Locations</i>
1.	2	257
2.	6	21
3.	7½	9
4.	7½	7

### Counting Ones

Suppose we wish to count the number of 1s in a word. We could of course check every bit in the word. But there is a quicker way if we remember that in any word and its two's complement the rightmost 1 is in the same position, both words are all 0s to the right of this 1, and no corresponding bits are the same to the left (the parts of both words at the left of the rightmost 1 are complements). Hence using the negative of a word as a mask for the word in a test instruction selects only the rightmost 1 for modification. The example uses three accumulators: the word being tested (which is lost) is in T, the count is kept in CNT, and the mask created in each step is stored in TEMP.

```

MOVEI  CNT,0     ;Clear CNT
MOVN   TEMP,T    ;Make mask to select rightmost 1
TDZE   T,TEMP    ;Clear rightmost 1 in T
AOJA   CNT,-2    ;Increase count and jump back
...

```

CNT is increased by one every time a 1 is deleted from T. After all 1s have been removed, the TDZE skips.

The preceding example uses little memory, but contains a loop so the time it takes is proportional to the number of 1s. The next example takes more memory but is constant; hence it is slower than the above when there are few 1s (less than eight), but is much faster when there are many. The word, which is lost, is in accumulator A, and the answer appears in accumulator

\*HAKMEM 140, item 169, page 79 (*Artificial Intelligence Memorandum, No. 239*, February 29, 1972, MIT Artificial Intelligence Laboratory).

$A+1$  (for convenience we let  $B = A+1$ ). The routine (due to Gosper, Mann and Leonard\*) has three distinct parts and is based on the fact that in a binary word, counting 1s is equivalent to calculating the sum of the digits. The first part, of seven instructions, manipulates the *octal* digits of the word so as to replace each digit by the number of 1s in it. Taking  $D$  as an octal digit and  $[x]$  as the largest integer contained in  $x$ , the algorithm used to make the substitution is

$$D - [D/2] - [D/4]$$

Of course the computer always acts in binary terms regardless of programmer interpretation. In this case the procedure carried out on each 3-bit piece  $abc$  is

$$abc - ab - a$$

The instructions effect this algorithm by shifting a copy of the word right one place, masking out the LSB of each shifted octal digit to prevent it from interfering with the next digit at the right (*ie* to isolate the digits), and subtracting the shifted word from the original. The same process is then repeated, this time masking out what was originally the middle bit in each digit. That this algorithm gives the correct substitution is evident from the following table, in which it is seen that the bottom number in a given column is the sum of the bits in the octal digit given at the top of the column.

<i>Original digit</i>	0	1	2	3	4	5	6	7
<i>Subtract</i>	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>2</u>	<u>2</u>	<u>3</u>	<u>3</u>
	0	1	1	2	2	3	3	4
<i>Subtract</i>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>
<i>Number of 1s</i>	0	1	1	2	1	2	2	3

We have now replaced the original word with a set of twelve numbers, whose sum is equal to the number of 1s in the original. The next three instructions add together pairs of adjacent numbers so as to replace the twelve by six whose sum is still the same. Since these new numbers are isolated in 6-bit pieces of the word, we shall revise our point of view, and regard them as digits in a number in base 64. Now any number is simply the sum of the values of its digits, *ie* of its digits each multiplied by an appropriate power of the base. Dividing each such summand by 1 less than the base gives the digit itself as remainder. Hence the third part of the routine just divides our 6-digit number by 63, producing in  $B$  a remainder that is the sum of the remainders from the individual digits, *ie* that is the sum of the digits.

In general terms this is the statement that the sum  $S$  of the digits in any number  $N$  in base  $b$  is  $N \bmod (b-1)$  — provided  $b$  is deliberately chosen such that  $S < b-1$ . The condition holds here of course as the number of 1s in a PDP-10 word is at most 36. And it is in fact to make this

```

MOVE  B,A           ;Copy in B
LSH   B,-1          ;Right one
AND   B,[333333,,333333] ;Masks out LSBs
SUB   A,B           ;D - [D/2]
LSH   B,-1          ;Right one again
AND   B,[333333,,333333] ;Mask out middle bits
SUBB  A,B           ;D - [D/2] - [D/4]; two copies

```



```

LSH      B,-3          ;Shift right one octal digit
ADD      A,B           ;Add numbers in digit pairs
AND      A,[070707,,070707] ;Throw out extra pair sums

IDIVI    A,77          ;Divide by 63, sum in B

```

condition hold that the routine converts from base 8 to base 64.

If it is known that the 1s in the word are entirely contained within bits 22–35 (the right fourteen bits), we can use the following somewhat shorter routine, which is faster than the loop for more than seven 1s. It first treats the number in quaternary, replacing each digit with the number of 1s in it, and then converts from quaternary to hexadecimal.

```

MOVEI    B,(A)
LSH      B,-1
ANDI     B,12525       ;Mask out LSBs
SUBB     A,B           ;D - D/2]; two copies

LSH      B,-2          ;Right one quaternary digit
ANDI     A,31463       ;Mask out some of digits in A
ANDI     B,31463       ;The rest in B
ADDI     A,(B)         ;Now combine digit pairs

IDIVI    A,17          ;Divide by 15, sum in B

```

Note that here we must get rid of one out of each set of two identical bit pairs before adding. This is because there can be digit overflow, *ie* a resulting hexadecimal digit can have more than two significant bits.

### Number Conversion

In the standard algorithm for converting a number  $N$  to its equivalent in base  $b$ , one performs the series of divisions

$$\begin{aligned}
 N/b &= q_1 + r_1/b & r_1 < b \\
 q_1/b &= q_2 + r_2/b & r_2 < b \\
 q_2/b &= q_3 + r_3/b & r_3 < b \\
 &\vdots \\
 q_{n-1}/b &= 0 + r_n/b & r_n < b
 \end{aligned}$$

The number in base  $b$  is then  $r_n \dots r_3 r_2 r_1$ . *Eg* the octal equivalent of 61 decimal is 75:

$$\begin{aligned}
 61/8 &= 7 + 5/8 \\
 7/8 &= 0 + 7/8
 \end{aligned}$$

The following decimal print routine converts a 36-bit positive integer in accumulator T to decimal and types it out. The contents of T and T+1 are destroyed. The routine is called by a PUSHJ P,DECPNT where P is the pushdown pointer.

```

DECPNT:  IDIVI    T,12          ;128 = 1010
          PUSH    P,T+1        ;Save remainder

```

```

SKIPE   T           ;All digits formed?
PUSHJ   P,DECPNT    ;No, compute next one
DECPN1: POP     P,T   ;Yes, take out in opposite order
        ADDI    T,60  ;Convert to ASCII (60 is code for 0)
        JRST   TTYOUT ;Type out

```

This routine repeats the division until it produces a zero quotient. Hence it suppresses leading zeros, but since it is executed at least once it outputs one "0" if the number is zero. The TTYOUT routine returns with a POPJ P, to DECPN1 until all digits are typed, then to the calling program.

Space can be saved in the pushdown stack by storing the computed digits in the left halves of the locations that contain the jump addresses. This is accomplished in the decimal print routine by changing

```
PUSH P,T+1 to HRLM T+1,(P)
```

and

```
POP P,T to HLRZ T,(P)
```

The routine can handle a 36-bit unsigned integer if the IDIVI T,12 is replaced by

MACRO interprets a number following  $\uparrow D$  as decimal.

```

LSHC    T,- $\uparrow D35$  ;Shift right 35 bits into T+1
LSH     T+1,-1         ;Vacate the T+1 sign bit
DIVI    T,12           ;Divide double length integer by 10

```

### Table Searching

Many data processing situations involve searching for information in tables and lists of all kinds. Suppose we wish to find a particular item in a table beginning at location TAB and containing  $N$  items. Accumulator T contains the item. The right half of A is used to index through the table, while the left half keeps a control count to signal when a search is unsuccessful.

```

MOVSI   A,-N         ;Put -N, 0 in A
CAMN    T,TAB(A)     ;Skip if current item not the one
JRST    FOUND        ;Item found
AOBJN   A,-2         ;Try next item until left count = 0
...     ;Item not in list

```

The location of the item (if found) is indicated by the number in the right half of A (its address is that quantity plus TAB). A slightly different procedure would be

```

HRLZI   A,-N
CAME    T,TAB(A)     ;Skip if current item is the one
AOBJN   A,-1
JUMPL   A,FOUND      ;Jump if left count < 0
...     ;Item not found

```

Locations used for a list can be scattered throughout memory if data is kept in the left half of each location and the right half addresses the next location in the list. The final location is indicated by a zero right half. The following routine finds the last half word item in the list. It is entered at FIND with the first location in the list addressed by the right half of accumulator T. At the end the final item is in T right.

```

      MOVE  T,(T)          ;Move next item to T
FIND:  TRNE  T,777777      ;Skip if AC right = 0
      JRST  .-2
      HLRZS T              ;Move final item to right

```

The following counts the length of the list in accumulator CNT.

```

      MOVEI CNT,0          ;Clear CNT
      JUMPE T,OUT          ;Jump out if T contains 0
      HRRZ  T,(T)          ;Get next address
      AOJA  CNT,.-2        ;Count and go back

```

### Double Precision Floating Point

The following are straightforward routines for handling double precision floating point arithmetic in software format, *ie* using single precision instructions, as would be required with a KA10 processor. [§2.6 describes the floating point instructions.]

```

DFAD:  UFA   A+1,M+1      ;Sum of low parts to A+2
      FADL  A,M           ;Sum of high parts to A, A+1
      UFA   A+1,A+2      ;Add low part of high sum to A+2
      FADL  A,A+2        ;Add low sum to high sum
      POPJ  P,
DFSB:  DFN   A,A+1       ;Negate double length operand
      PUSHJ P,DFAD       ;Call double floating add
      DFN   A,A+1       ;-(M - AC) = AC - M
      POPJ  P,
DFMP:  MOVEM A,A+2       ;Copy high AC operand in A+2
      FMPL  A+2,M+1     ;One cross product to A+2
      FMPL  A+1,M       ;Other to A+1
      UFA   A+1,A+2     ;Add cross products into A+2
      FMPL  A,M         ;High product to A, A+1
      UFA   A+1,A+2     ;Add low part to cross sum in A+2
      FADL  A,A+2       ;Add low sum to high part of product
      POPJ  P,

```

These routines are given to show the mechanics of double precision floating point operations. They produce correct results in all ordinary circumstances, but do not handle pathological cases.

A double precision division is of the form

$$\frac{A}{B} = \frac{a + c \times 2^{-27}}{b + d \times 2^{-27}}$$

Using the relationship

$$A/b = q + r \times 2^{-27}/b$$

where  $q$  and  $r$  are the quotient and remainder produced by FDVL, the following routine computes a double length quotient by the approximation

$$\frac{A}{B} \cong q + \frac{(r - qd) \times 2^{-27}}{b}$$

which gives a result correct to the next-to-last bit in the low order half.

DFDV:	FDVL	A,M	;Get high part of quotient
	MOVN	A+2,A	;Copy negative of quotient in A+2
	FMPR	A+2,M+1	;Multiply by low part of divisor
	UFA	A+1,A+2	;Add remainder
	FDVR	A+2,M	;Divide sum by high part of divisor
	FADL	A,A+2	;Add result to original quotient
	POPJ	P,	

*Proof:* Using the expansion

$$\frac{1}{x+y} = \frac{1}{x} \left[ 1 - \frac{y}{x} + \frac{y^2}{x^2} - \frac{y^3}{x^3} + \dots \right] \quad (y^2 < x^2)$$

and letting  $x = b$  and  $y = d2^{-27}$  gives

$$\frac{A}{B} = \left( q + \frac{r2^{-27}}{b} \right) \left[ 1 - \frac{d2^{-27}}{b} + \frac{d^2 2^{-54}}{b^2} - \frac{d^3 2^{-81}}{b^3} + \dots \right]$$

Multiplying out and gathering like terms gives

$$\frac{A}{B} = q + \frac{1}{b} (r - qd) 2^{-27} - \frac{d}{b^2} (r - qd) 2^{-54} + \frac{d^2}{b^3} (r - qd) 2^{-81} - \dots$$

where the first two terms on the right are those in the approximation given above.

The ratio of adjacent terms is

$$\frac{T_{n+1}}{T_n} = \frac{-d2^{-27}}{b}$$

In an alternating convergent series, the error due to truncation is smaller than the first term dropped. Therefore

$$|\text{Error}| < \frac{d2^{-27}}{b} T_n$$

Since the maximum value of  $d$  is less than 1 and the minimum value of  $b$  (normalized) is  $\frac{1}{2}$ ,

$$|\text{Error}| < T_n 2^{-26}$$

## 2.12 INPUT-OUTPUT

The input-output instructions govern all transfers of data to and from the peripheral equipment, and also perform many operations within the processor. An instruction in the in-out class is designated by 111 in bits 0-2, *ie* its left octal digit is 7. Bits 3-9 address the device that is to respond to the instruction. The format thus allows for 128 codes, two of which, 000 and 004 respectively, address the processor and priority interrupt, and are used for the console as well. The KA10 also uses the first two codes for the time share hardware, but the KI10 has a separate code, 010, for this purpose. A chart in Appendix A lists all devices for which codes have been assigned, and gives their mnemonics and DEC option numbers. Electrical and logical specifications of the IO bus are given in the interface manual.

Bits 13-35 are the same as in all other instructions: they are the *I*, *X*, and *Y* parts, which are used to calculate an effective address, set of conditions, or mask to be used in the execution of the instruction. The remaining bits, 10-12, select one of the following eight IO instructions.

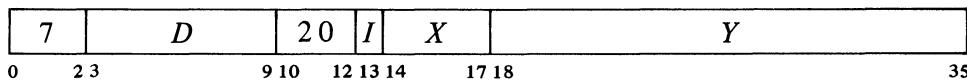
### NOTE

All instructions described in the remainder of this manual are in-out instructions, which are affected by the time share instruction restrictions. In the KA10 no in-out instruction can be performed by a user mode program unless the User In-out flag is set. In the KI10, in-out instructions using device codes 740 and above are not restricted. But an instruction using a device code under 740 cannot be performed by a user mode program unless User In-out is set and cannot be performed in supervisor mode at all (in-out is normally handled in kernel mode). Any in-out instruction that violates these restrictions does not perform the functions given for it in the instruction description. Instead it acts just like an MUUO [§2.10].

These restrictions will not be mentioned in the instruction descriptions, as they apply to *all* instructions from this point on.

Input and output for system users is normally handled by the Monitor using MUUOs and various software formats. For information on user procedures vis-a-vis Monitor handling of user IO requirements, refer to Chapters 4-6 of *DECsystem-10 Monitor Calls*, manual DEC-10-MRRx-D.

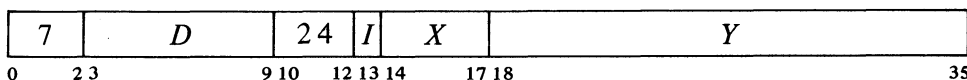
### CONO Conditions Out



Set up device *D* with the effective initial conditions *E*. The number of condition bits in *E* that are actually used depends on the device.

*E* will always be regarded as being bits 18-35, even though it is actually placed on both halves of the bus and many devices receive the information from the left half.

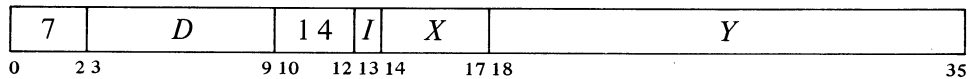
### CONI Conditions In



Read the input conditions from device *D* and store them in location *E*. The

number of condition bits stored depends on the device; the remaining bits in location *E* are cleared.

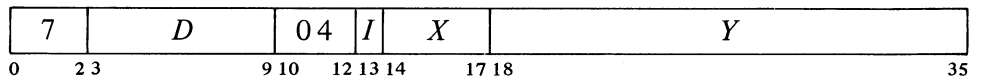
**DATAO Data Out**



Send the contents of location *E* to the data buffer in device *D*, and perform whatever control operations are appropriate to the device.

The amount of data actually accepted by the device depends on the size of its buffer, its mode of operation, etc. The original contents of location *E* are unaffected.

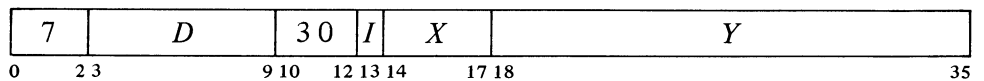
**DATAI Data In**



Move the contents of the data buffer in device *D* to location *E*, and perform whatever control operations are appropriate to the device.

The number of data bits stored depends on the size of the device buffer, its mode of operation, etc. Bits in location *E* that do not receive data are cleared.

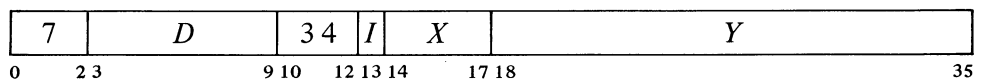
**CONSZ Conditions In and Skip if Zero**



Test the input conditions from device *D* against the effective mask *E*. If all condition bits selected by 1s in *E* are 0s, skip the next instruction in sequence.

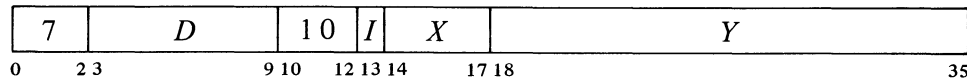
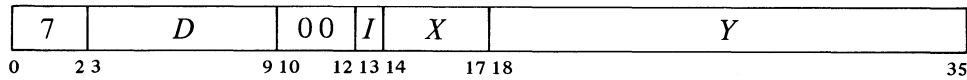
If the device supplies more than 18 condition bits, only the right 18 are tested.

**CONSO Conditions In and Skip if One**



Test the input conditions from device *D* against the effective mask *E*. If any condition bit selected by a 1 in *E* is 1, skip the next instruction in sequence.

If the device supplies more than 18 condition bits, only the right 18 are tested.

**BLKO      Block Out****BLKI      Block In**

Add one to each half of a pointer in location *E*, and place the result back in *E*. Then perform a data IO instruction in the same direction as the block IO instruction, using the right half of the incremented pointer as the effective address. If the given instruction is a BLKO, perform a DATAO; if a BLKI, perform a DATAI.

The remaining actions taken by this instruction depend on whether it is executed as a priority interrupt instruction [§2.13].

◆ *Not as an Interrupt Instruction.* If the addition has caused the count in the left half of the pointer to reach zero, go on to the next instruction in sequence. Otherwise skip the next instruction.

◆ *As an Interrupt Instruction.* If the addition has caused the count in the left half of the pointer to reach zero, execute the instruction in the second interrupt location for the channel. Otherwise dismiss the interrupt and return to the interrupted program.

Note: The KA10 increments the two halves of the pointer by adding  $1000001_8$  to the entire register. In the KI10 the two halves are handled independently.

The above eight instructions differ from one another in their total effect, but they are not all different with respect to any given device. A BLKO acts on a device in exactly the same way as a DATAO — the two differ only in counting and other operations carried out within the processor and memory. Similarly, no device can distinguish between a BLKI and a DATAI; and a device always supplies the same input conditions during a CONI, CONSZ or CONSO whether the program tests them or simply stores them.

Hence the eight instructions may be categorized as of four types, represented by the first four instructions described above. Moreover, a complete treatment of the programming of any device can be given in terms of these four instructions, two of which are for input and two for output. The four exhaust the types of information transfer that occur in the IO system, at least three of which are applicable to any given device. Thus all instruction descriptions in the rest of this manual will be of the CONO, CONI, DATAO and DATAI instructions combined with the various device codes. The discussion of each device will present timing information pertinent to device operation, as internal device timing is dependent only upon the device and not upon processor instruction time (which is given in Appendix D).

Every device requires initial conditions; these are sent by a CONO, which

A block IO instruction is effectively a whole in-out data handling subroutine. It keeps track of the block location, transfers each data word, and determines when the block is finished.

Initially the left half of the pointer contains the negative of the number of words in the block, the right half contains an address one less than that of the first word in the block.

The word “input” used without qualification always refers to the transfer of data from the peripheral equipment into the processor; “output” refers to the transfer in the opposite direction.

can supply up to eighteen bits of control information to the device control register. The program can determine the status of the device from up to thirty-six bits of input conditions that can be read by a CONI (but only the right eighteen can be tested by a CONSZ or CONSO). Some input bits simply reflect initial conditions sent by a previous CONO; others are set up by output conditions but are subject to subsequent adjustment by the device; and still others, such as status levels from a tape transport, have no direct connection with output conditions.

Data is moved in and out in characters of various sizes or in full 36-bit words. Each transfer between memory and a device data buffer requires a single DATAI or DATAO. Every device has a CONO and CONI, but it may have only one data instruction unless it is capable of both input and output. *Eg*, the paper tape reader has only a DATAI, the tape punch has only a DATAO, but the console terminal has both. (A high speed device, such as a disk file, can be connected to a direct-access processor, which in turn is connected directly to memory by a separate memory bus and handles data automatically. This eliminates the need for the program to give a DATAO or DATAI for each transfer.)

**A Typical IO Device.** Every device has a 7-bit device selection network, a priority interrupt assignment, and at least two flags, Busy and Done, or some equivalent. The selection network decodes bits 3–9 of the instruction so that only the addressed device responds to signals sent by the processor over the in-out bus. To use the device with the priority interrupt, the program must assign a channel to it. Then whenever an appropriate event occurs in the device, it requests an interrupt on the assigned channel.

The Busy and Done flags together denote the basic state of the device. When both are clear the device is idle. To place the device in operation, a CONO or DATAO sets Busy. If the device will be used for output, the program must give a DATAO that sends the first unit of data – a word or character depending on how the device handles information. When the device has processed a unit of data, it clears Busy and sets Done to indicate that it is ready to receive new data for output, or that it has data ready for input. In the former case the program would respond with a DATAO to send more data; in the latter, with a DATAI to bring in the data that is ready. If an interrupt channel has been assigned to the device, the setting of Done signals the program by requesting an interrupt; otherwise the program must keep testing Done to determine when the device is ready.

All devices function basically as described above even though the number of initial conditions varies considerably. Besides Busy and Done flags, the tape reader and punch have a Binary flag that determines the mode of operation of the device with respect to the data it processes – alphanumeric or binary. The terminal has no binary flag, but it has two Busy flags and two Done flags – one pair for input, another for output. A complicated device, such as magnetic tape, may require two device codes to handle the large number of conditions associated with it. Initial conditions for a tape system include a transport address and an actual command the tape control is to perform; input conditions include error flags and transport status levels.

Most IO devices involve motion of some sort, usually mechanical (in a display only the electron beam moves). With respect to mechanical motion

A DATAI that addresses an output-only device simply clears location *E*. DATAI PI, (code 70044) produces only this effect as the priority interrupt has no data for input. On the other hand a DATAO that addresses an input-only device is a no-op.

When the device code is undefined or the addressed device is not in the system, a DATAO, CONO or CONSO is a no-op, a CONSZ is an absolute skip, a DATAI or CONI clears location *E*.

Busy and Done both set is a meaningless situation.

Occasionally a device with a second code may use a DATAI or DATAO to transmit additional control or maintenance information.



there are two types of devices, those that stay in motion and those that do not. Magnetic tape is an example of the former type. Here the device executes a command (such as read, write, space forward) and the done flag indicates when the entire operation is finished. A separate data flag signals each time the device is ready for the program to give a DATAI or DATAO, but the tape keeps moving until an entire record or file has been processed.

Paper tape, on the other hand, stops after each transfer, but the program need not give a new CONO every time. The reader logic is set up so that a DATAI not only reads the data, but also clears Done and sets Busy. Hence if the instruction is given within a critical time, the tape moves continuously and only two CONOs are required for a whole series of transfers: one to start the tape, and one to stop it after the final DATAI.

Other devices operate in one or the other of these two ways but differ in various respects. The tape punch and terminal output are like the reader. Terminal input is initiated by the operator striking a key rather than by the program. The card reader reads an entire card on a single CONO, with a DATAI required for each column. The DECTape stays in motion, and the program must give a CONO to stop it or it will go all the way to the end zone.

### Readin Mode

This mode of processor operation provides a means of placing information in memory without relying on a program already in memory or loading one word at a time manually. Its principal use is to read in a short loader program which is then used for loading other information. A loader program should ordinarily be used rather than readin mode, as a loader can check the validity of the information read.

Pressing the readin key on the console activates readin mode by starting the processor in a special hardware sequence that simulates a DATAI followed by a series of BLKI instructions, all of which address the device whose code is selected by the readin device switches at the left just above the console operator panel. Various devices can be used, and for each there are special rules that must be followed. But the readin mode characteristics of any particular device are treated in the discussion of the device. Here we are concerned only with the general characteristics.

The information read is a block of data (such as a loader program) preceded by a pointer for the BLKI instructions. The left half of the pointer contains the negative of the number of words in the block, the right half contains an address one less than that of the location that is to receive the first word.

To read in, the operator must set up the device he is using, set its code into the readin device switches, and press the readin key. This key function first duplicates the action of the console reset key, which clears both the processor and the in-out equipment; in particular it places the processor in executive mode, and in the KI10 selects kernel mode with executive paging disabled, so all access will be to the first 256K of physical memory unpagged. Following this the processor places the device in operation, brings the first

At present readin is limited to paper tape, DECTape, and standard magnetic tape.

word (the pointer) into location 0, and then reads the data block, placing the words in the locations specified by the pointer. Data can be placed anywhere in the first 256K of memory (including fast memory) except in location 0. The operation affects none of memory except location 0 and the block area.

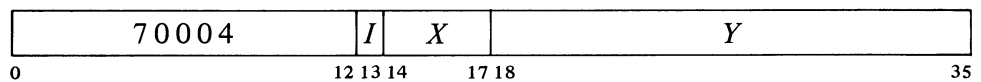
Upon completing the block, the processor leaves readin mode and begins normal operation. This is done in the KI10 by jumping to the location containing the last word in the block, in the KA10 by executing the last word as an instruction. In the KA10 the processor stops after executing the first instruction if the single instruction switch is on.

### Console-Program Communication

Neither the processor nor the priority interrupt system require all four types of IO instructions, so the program can make use of their device codes for communicating with the console. Both processors have two instructions that transfer data between console and program. But in the KI10, the program can actually operate some of the switches on the console. For this purpose it uses a data-out instruction with the device code for the paper tape reader (an input-only device). The KI10 program can also inspect the states of a number of operating and sense switches, but the bits for these are included in the left half words of the standard input conditions for the interrupt and processor [ §§2.13, 2.14].

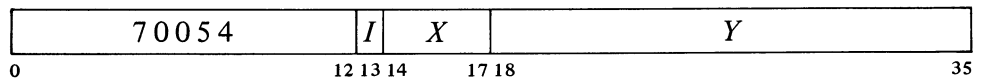
MACRO also recognizes the mnemonic RSW (Read Switches) as equivalent to DATAI APR,.

#### DATAI APR, Data In, Console



Read the contents of the console data switches into location *E*.

#### DATAO PI, Data Out, Console



Unless the console MI program disable switch is on, display the contents of location *E* in the console memory indicators and turn on the triangular light beside the words PROGRAM DATA just above the indicators (turn off the light beside MEMORY DATA).

Once the indicators have been loaded by the program, no address condition selected from the console [Appendix F] can load them until the operator turns on the MI program disable switch, executes a key function that references memory, or presses the reset key.



The request signal is generally derived from a flag that is set by various conditions in the device. Often associated with these flags are enabling flags, where the setting of some device condition flag can request an interrupt on the assigned channel only if the associated enabling flag is also set. The enabling flags are in turn controlled by the conditions supplied to the device by a CONO. *Eg* a device may have half a dozen flags to indicate various internal conditions that may require service by an interrupt; by setting up the associated enabling flags, the program can determine which conditions shall actually request interrupts in any given circumstances.

Note that there are therefore two orders of priority associated with an interrupt: first the channel, and then for all devices requesting interrupts simultaneously on the same channel, proximity to the processor on the bus. For priority purposes, all devices on the left bus are closer than those on the right bus.

conditions are satisfied. A single channel will shut out all others of lower priority if every time its service routine dismisses the interrupt, a device assigned to it is already waiting with another request. The program can usually trigger a request from a device but delay its acceptance by turning on the channel later.

Having accepted a request, the processor will do nothing further with it unless the priority interrupt system is on. But even with the system off, the processor will continue to accept requests on other channels; and when the system is finally turned on, it will respond as though all requests had just been accepted, handling the highest priority one first.

The way in which interrupts are handled, the conditions that affect them, and so forth depend upon the type of processor.

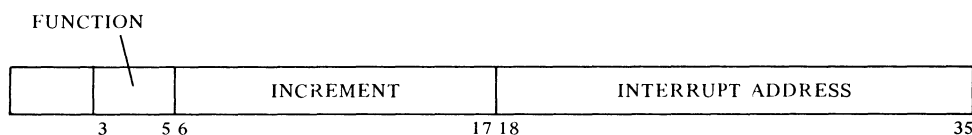
### KI10 Interrupt

A request made to an active channel is accepted immediately unless some channel is already waiting for an interrupt to start or an interrupt is starting for some channel. Once a request is accepted with the system on, the channel must wait for the interrupt to start. The processor however will delay any action on the request if it is already holding an interrupt for the same channel or for a channel with priority higher than those on which requests have been accepted (in other words if the current program is a higher priority interrupt routine). When a waiting channel has priority higher than the current program, the processor sends an interrupt-granted signal for the waiting channel that has highest priority. This action makes use of the IO bus. Should the bus be busy, the grant is sent as soon as the bus becomes available, taking precedence over any IO instruction that may also be waiting (note that in this situation the program actually stops). The grant signal goes out on the bus and is transmitted serially from one device to the next. Upon receiving the grant, a device that is not requesting an interrupt on the specified channel sends the signal on to the next device. A device that is requesting an interrupt on the specified channel terminates the signal path and sends an interrupt function word back to the processor.

Upon receipt of the function word, the processor stops the current program at the first allowable point to start an interrupt for the waiting channel for which the grant was made. Allowable stopping points are at the completion of an instruction, following the retrieval of an address word in an effective address calculation (including the second calculation using the pointer in a byte instruction), between transfers in a BLT, between steps in the calculation of the first part of the quotient in double floating division, and while an IO instruction is waiting for the bus. When an interrupt starts, PC points to the interrupted instruction, so that a correct return can later be made to the interrupted program.

The action taken by the processor in starting an interrupt depends upon the function specified by the function word returned to the processor. Two fixed locations in the executive process table are associated with each channel: locations  $40 + 2N$  and  $41 + 2N$ , where  $N$  is the channel number. Channel 1 uses locations 42 and 43, channel 2 uses 44 and 45, and so on to

channel 7 which uses 56 and 57. The processor starts a “standard” interrupt for channel  $N$  by executing the instruction in the first interrupt location for the channel, *ie* location  $40 + 2N$ . The fixed locations however need not be used. The interrupt function word sent by the device may specify a standard interrupt using the fixed locations, or an equivalent interrupt using a pair of locations specified by the function word, or some other interrupt function entirely. The format of the function word and the operations the processor performs in response to the function selected by bits 3–5 of the word are as follows.

*Bits 3–5**Interrupt Function*

- |   |  |
|---|--|
| 0 | Processor waiting. If no response, perform a standard interrupt (see function 1).  |
| 1 | Standard interrupt – execute the instruction in location $40 + 2N$ of the executive process table.   |
| 2 | Dispatch – execute the instruction in the location specified by bits 18–35.  |
| 3 | Increment – add the contents of bits 6–17 to the contents of the location specified by bits 18–35. The increment is a fixed point number in twos complement notation, bit 6 being the sign, and bit 17 corresponding to bit 35 of the memory word. |
| 4 | DATAO – do a DATAO for this device using the contents of bits 18–35 as the effective address.  |
| 5 | DATAI – do a DATAI for this device using the contents of bits 18–35 as the effective address.  |
| 6 | Not used – reserved by DEC.  |
| 7 | Not used – reserved by DEC.  |

A device designed originally for use with the KA10 will work when connected to the KI10 bus, where it always requests a standard interrupt by providing no response to the grant. Note that for simultaneous requests on a given channel, all KI10 devices that return a function word have priority over all KA10 devices and over any KI10 devices that do not return a function word. The last group includes the reader, punch and teletypewriter, which are contained in the processor, as well as the processor itself acting as a device [see *processor conditions*, §2.14].

At present, functions 6 and 7 produce standard interrupts.

Regardless of what mode the processor is in when an interrupt occurs, the interrupt operations are performed in kernel mode. No interrupt operation can set Overflow or either of the trap flags; hence an overflow trap can never occur as a direct result of an interrupt. A page failure that occurs in an interrupt operation is never trapped; instead it sets the In-out Page Failure flag, which requests an interrupt on the channel assigned to the processor [§2.14]. These considerations of course do not apply to a service routine called by an interrupt instruction.

**Interrupt Instructions.** An instruction executed in response to an interrupt request and not under control of PC is referred to elsewhere in this manual as being “executed as an interrupt instruction.” Some instructions, when so executed, have different effects than they do when performed in other circumstances. And the difference is not due merely to being per-

These locations may be the fixed ones for a standard interrupt or those given by the function word for a dispatch interrupt.

Satisfaction of the condition does not change PC, as this would skip the next instruction in the interrupted program. In effect the instruction skips back to the interrupted program by skipping the second interrupt location.

Note that the interpretation of a BLKI or BLKO as a skip instruction is consistent with the description given in §2.12, the condition being that the count is not zero.

formed in an interrupt location or in response (by the program) to an interrupt. To be an interrupt instruction, an instruction must be executed in the first or second interrupt location for a channel, in direct response by the hardware (rather than by the program) to a request on that channel. §2.12 describes the two ways a BLKO is performed. If a BLKO is contained in an interrupt routine called by a JSR, it is not “executed as an interrupt instruction” even in the unlikely event the routine is stored within the interrupt locations and the BLKO is executed by an XCT. The interrupt instructions executed in a standard or dispatch interrupt fall into three categories.

◆ *AOSX, SKIPX, SOSX, CONSX, BLKX*. If the skip condition specified by the instruction is satisfied, the processor dismisses the interrupt and returns immediately to the interrupted program (*ie* it returns control to the unchanged PC). If the skip condition is not satisfied, the processor executes the instruction contained in the second interrupt location.

#### CAUTION

In the second interrupt location, a skip instruction whose condition is not satisfied hangs up the processor, which will keep repeating the instruction until the condition is satisfied.

◆ *JSR, JSP, PUSHJ, MUUO*. The processor holds an interrupt on the channel, takes the next instruction from the location specified by the jump (as indicated by the newly changed PC), and enters either kernel mode or the mode specified by the new PC word of the MUUO. Hence the instruction is usually a jump to a service routine handled by the Monitor.

◆ *All Other Instructions*. In general the processor simply executes the instruction, dismisses the interrupt, and then returns to the interrupted program. If the instruction is a jump (other than those mentioned above), the processor jumps to the newly specified location; but it dismisses the interrupt and returns to the mode it was already in when the interrupt occurred. Hence it effectively returns to the interrupted program but in a different place, and the original contents of PC are lost.

Since the interrupt operations are performed in kernel mode regardless of the actual mode of the processor, an XCT is performed as an executive XCT [§2.15]. The ultimate effect of the XCT depends of course on the instruction executed – and its effect is as described here for the various categories.

#### CAUTION

Neither an LUUO, a BLT, a DMOVEM, nor a DMOVNM will function in a reasonable manner as an interrupt instruction. Therefore do not use them.

**Dismissing an Interrupt.** Unless the interrupt operation dismisses the interrupt automatically, the processor holds an interrupt until the program dismisses it, even if the interrupt routine is itself interrupted by a higher priority channel. Thus interrupts can be held on a number of channels simultaneously, but from the time an interrupt is started until it is dismissed,

no interrupt can be started on that channel or any channel of lower priority (requests, however, can be accepted on lower priority channels).

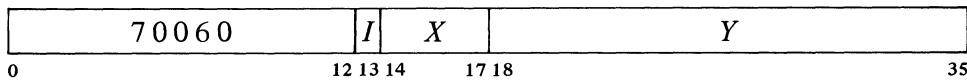
A routine dismisses the interrupt by using a JEN (JRST 12,) to return to the interrupted program (the interrupt system must be on when the JEN is given). This instruction restores the channel on which the interrupt is being held, so it can again accept requests, and interrupts can be started on it and lower priority channels. JEN also restores the flags, whose states were saved in the left half of the PC word if the routine was called by a JSR, JSP, PUSHJ, or MUUO. If flag restoration is not desired, a JRST 10, can be used instead.

*CAUTION*

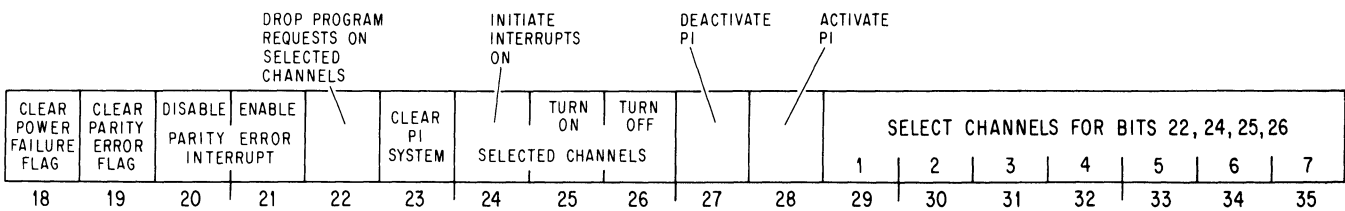
An interrupt routine must dismiss the interrupt when it returns to the interrupted program, or its channel and all channels of lower priority will be disabled, and the processor will treat the new program as a continuation of the interrupt routine.

**Priority Interrupt Conditions.** The program can control the priority interrupt system by means of condition IO instructions. The device code is 004, mnemonic PI.

**CONO PI, Conditions Out, Priority Interrupt**



Perform the functions specified by the effective conditions *E* as shown (a 1 in a bit produces the indicated function, a 0 has no effect).



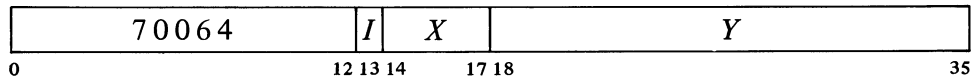
- 20 Prevent the setting of the Parity Error flag from requesting an interrupt on the channel assigned to the processor.
- 21 Enable the setting of the Parity Error flag to request an interrupt on the channel assigned to the processor.
- 22 On channels selected by 1s in bits 29–35, turn off any interrupt requests made previously by the program (via bit 24).
- 23 Deactivate the priority interrupt system, turn off all channels, eliminate all interrupt requests that have already been accepted but are still waiting, and dismiss all interrupts that are currently being held.

Bits 18–21 are actually for processor conditions [§2.14].

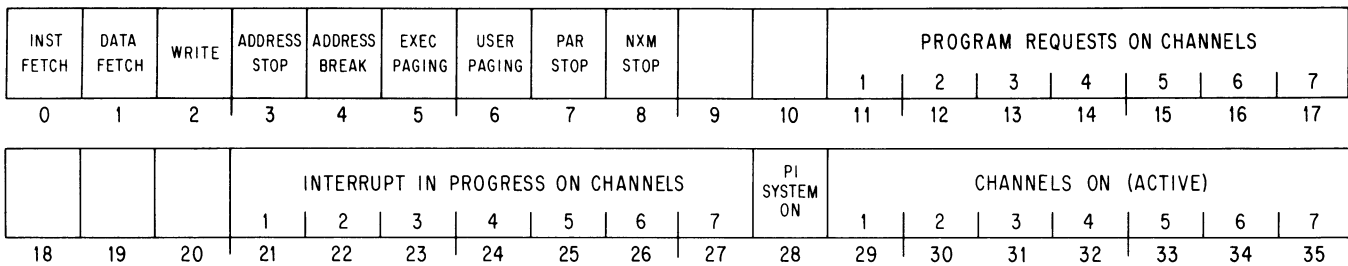
For other than the highest priority channel, the greater the number of higher priority channels active, the greater the amount of program time available both initially and between successive interrupts. If the program forces an interrupt on the lowest priority channel when all are active, there can be as much as 40  $\mu$ s of program time between the CONO PI, and its interrupt.

- 24 Request interrupts on channels selected by 1s in bits 29–35, and force the processor to accept them even on channels that are off. The request remains indefinitely, so as soon as an interrupt is completed on a given channel another is started, until the request is turned off by a CONO that selects the same channel and has a 1 in bit 22.  
Remember that the processor allows the program to continue while it grants an interrupt. Thus when this bit forces acceptance of a request, another program instruction or two may be performed before the interrupt, even on the highest priority channel. Moreover if the request is allowed to remain, additional instructions may be performed between successive interrupts.
- 25 Turn on the channels selected by 1s in bits 29–35 so interrupt requests can be accepted on them.
- 26 Turn off the channels selected by 1s in bits 29–35, so interrupt requests cannot be accepted on them unless made by a CONO PI, with a 1 in bit 24.
- 27 Deactivate the priority interrupt system. The processor can then still accept requests, but it can neither start nor dismiss an interrupt.
- 28 Activate the priority interrupt system so the processor can accept requests and can start, hold and dismiss interrupts.

**CONI PI, Conditions In, Priority Interrupt**



Read the status of the priority interrupt (and nine console operating switches) into location *E* as shown.



Channels that are active are indicated by 1s in bits 29–35; 1s in bits 21–27 indicate channels on which interrupts are currently being held; 1s in bits 11–17 indicate channels that are receiving interrupt requests generated by a CONO PI, with a 1 in bit 24. A 1 in bit 28 means the interrupt system is on.

The remaining conditions read by this instruction have nothing to do with the interrupt. Bits 0–8 reflect the settings of various console operating switches; for information on these switches refer to Appendix F1.



**Timing.** The time a device must wait for an interrupt to start depends on the number of channels in use, and how long the service routines are for devices on higher priority channels. If only one device is using interrupts, it need never wait longer than 10  $\mu$ s.

**Special Considerations.** On a return to an interrupted program, the processor always starts the interrupted instruction over from the beginning. This causes special problems in a BLT and in byte manipulation.

An interrupt can start following any transfer in a BLT. When one does, the BLT puts the pointer (which has counted off the number of transfers already made) back in AC. Then when the instruction is restarted following the interrupt, it actually starts with the next transfer. This means that if interrupts are in use, the programmer cannot use the accumulator that holds the pointer as an index register in the same BLT, he cannot have the BLT load AC except by the final transfer, and he cannot expect AC to be the same after the instruction as it was before.

An interrupt can also start in the second effective address calculation in a two-part byte instruction. When this happens, First Part Done is set. This flag is saved as bit 4 of a PC word, and if it is restored by the interrupt routine when the interrupt is dismissed, it prevents a restarted ILDB or IDPB from incrementing the pointer a second time. This means that the interrupt routine must check the flag before using the same pointer, as it now points to the next byte. Giving an ILDB or IDPB would skip a byte. And if the routine restores the flag, the interrupted ILDB or IDPB would process the same byte the routine did.

**Programming Suggestions.** The Monitor handles all interrupts for user programs. Even if the User In-out flag is set, a user program generally cannot reference the interrupt locations to set them up. Procedures for informing the Monitor of the interrupt requirements of a user program are discussed in the Monitor manual.

For those who do program priority interrupt routines, there are several rules to remember.

- ◆ No requests can be accepted, not even on higher priority channels, while an interrupt is starting. Therefore do not use lengthy effective address calculations in interrupt instructions.
- ◆ Most in-out devices are designed to drop an interrupt request when the program responds, usually with a DATAI or DATAO. If an interrupt is handled neither by a BLKI or BLKO interrupt instruction nor by a service routine, the programmer must make sure the device is configured to drop the request on receipt of whatever response the program does give.
- ◆ The interrupt instruction that calls the routine must save PC if there is to be a return to the interrupted program. Generally a JSR is used as it saves both PC and the flags, and it uses no accumulator.
- ◆ The principal function of an interrupt routine is to respond to the situation that caused the interrupt. *Eg* computations that can be performed outside the routine should not be included within it.

- ◆ If the routine uses a UWO it must first save the contents of the pair of locations that will be changed by it in case the interrupted program was in the process of handling a UWO of the same type. For an MUWO the routine must save locations 424 and 425 of the user process table. For an LUWO the routine must save location 40 in the executive process table and the location used by the UWO handler instruction to store the PC word.
- ◆ The routine must dismiss the interrupt (with a JEN) when returning to the interrupted program. The flags and UWO locations should be restored.

### KA10 Interrupt

A request made to an active channel is accepted at the next memory access unless the processor is starting an interrupt for any channel or holding an interrupt for the same channel. Once a request is accepted with the system on, the channel must wait for the interrupt to start. The processor however cannot start an interrupt if it is already holding an interrupt for a channel with priority higher than those on which requests have been accepted (in other words if the current program is a higher priority interrupt routine). When there is a higher priority channel waiting, the processor stops the current program at the first allowable point to start an interrupt for the waiting channel that has highest priority. Allowable stopping points are following the retrieval of an instruction, following the retrieval of an address word in an effective address calculation (including the second calculation using the pointer in a byte instruction), and between transfers in a BLT. When an interrupt starts, PC points to the interrupted instruction, so that a correct return can later be made to the interrupted program.

Two memory locations are associated with each channel: unrelocated locations  $40 + 2N$  and  $41 + 2N$ , where  $N$  is the channel number. Channel 1 uses locations 42 and 43, channel 2 uses 44 and 45, and so on to channel 7 which uses 56 and 57. The processor starts an interrupt for channel  $N$  by executing the instruction in location  $40 + 2N$ . Even though the processor may be in user mode when an interrupt occurs, interrupt instructions are performed in executive mode.

**Interrupt Instructions.** An instruction executed in response to an interrupt request and not under control of PC is referred to elsewhere in this manual as being “executed as an interrupt instruction.” Some instructions, when so executed, have different effects than they do when performed in other circumstances. And the difference is not due merely to being performed in an interrupt location or in response (by the program) to an interrupt. To be an interrupt instruction, an instruction must be executed in location  $40 + 2N$  or  $41 + 2N$ , in direct response by the hardware (rather than by the program) to a request on channel  $N$ . §2.12 describes the two ways a BLKO is performed. If a BLKO is contained in an interrupt routine called by a JSR, it is not “executed as an interrupt instruction” even in the unlikely event the routine is stored within the interrupt locations and the BLKO is executed by an XCT. There are two categories of interrupt instructions.

- ◆ *Non-IO Instructions.* After executing a non-IO interrupt instruction, the processor holds an interrupt on the channel and returns control to PC.

Interrupt locations for a second processor on the same memory are  $140 + 2N$  and  $141 + 2N$ .

Hence the instruction is usually a jump to a service routine. If the processor is in user mode and the interrupt instruction is a JSR, JSP, PUSHJ, JSA or JRST, the processor leaves user mode (the Monitor thus handles all interrupt routines [§2.16]).

If the interrupt instruction is not a jump, the processor continues the interrupted program while holding an interrupt – in other words it now treats the interrupted program as an interrupt routine. *Eg* the instruction might just move a word to a particular location. Such procedures are usually reserved for maintenance routines or very sophisticated programs.

◆ *Block or Data IO Instructions.* One or the other of two actions can result from executing one of these as an interrupt instruction.

If the instruction in  $40 + 2N$  is a BLKI or BLKO and the block is not finished (*ie* the count does not cause the left half of the pointer to reach zero), the processor dismisses the interrupt and returns to the interrupted program. The same action results if the instruction is a DATAI or DATAO.

If the instruction in  $40 + 2N$  is a BLKI or BLKO and the count does reach zero, the processor executes the instruction in location  $41 + 2N$ . This *cannot* be an IO instruction and the actions that result from its execution as an interrupt instruction are those given above for non-IO instructions.

#### CAUTION

The execution, as an interrupt instruction, of a CONO, CONI, CONSO or CONSZ in location  $40 + 2N$  or *any* IO instruction in location  $41 + 2N$  hangs up the processor.

**Dismissing an Interrupt.** Automatic dismissal of an interrupt occurs only in a DATAI or DATAO, or in a BLKI or BLKO with an incomplete block. Following any non-IO interrupt instruction, the processor holds an interrupt until the program dismisses it, even if the interrupt routine is itself interrupted by a higher priority channel. Thus interrupts can be held on a number of channels simultaneously, but from the time an interrupt is started until it is dismissed, no interrupt can be started on that channel or any channel of lower priority (requests, however, can be accepted on lower priority channels).

A routine dismisses the interrupt by using a JEN (JRST 12,) to return to the interrupted program (the interrupt system must be on when the JEN is given). This instruction restores the channel on which the interrupt is being held, so it can again accept requests, and interrupts can be started on it and lower priority channels. JEN also restores the flags, whose states were saved in the left half of the PC word if the routine was called by a JSR, JSP, or PUSHJ. If flag restoration is not desired, a JRST 10, can be used instead.

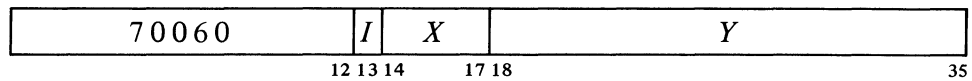
#### CAUTION

An interrupt routine must dismiss the interrupt when it returns to the interrupted program, or its channel and all channels of lower priority will be

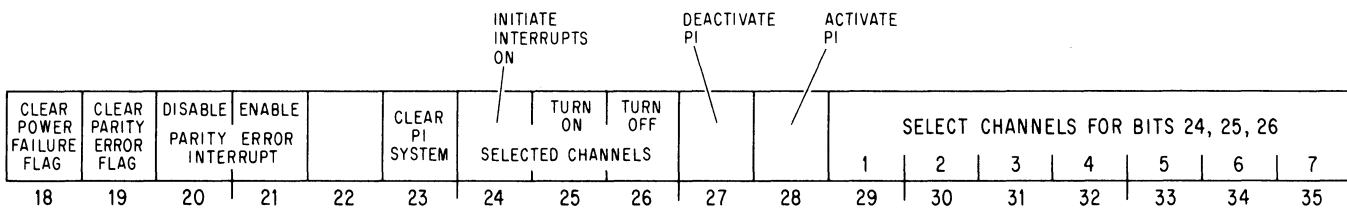
disabled, and the processor will treat the new program as a continuation of the interrupt routine.

**Interrupt Conditions.** The program can control the interrupt system by means of condition IO instructions. The device code is 004, mnemonic PI.

**CONO PI, Conditions Out, Priority Interrupt**



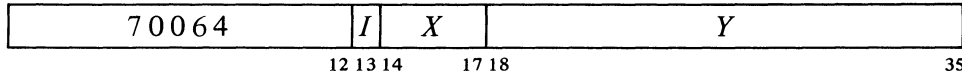
Perform the functions specified by the effective conditions *E* as shown (a 1 in a bit produces the indicated function, a 0 has no effect).



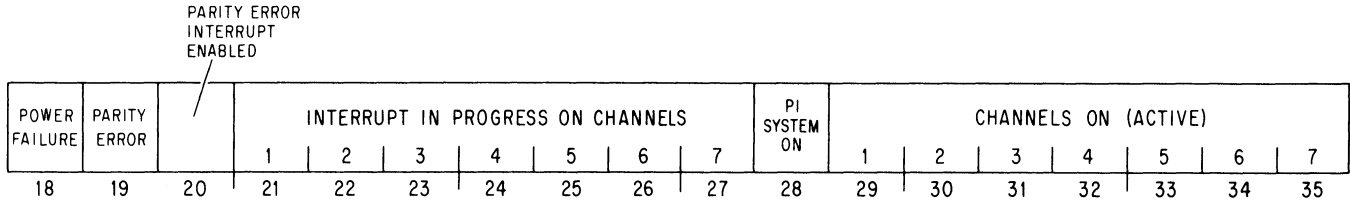
Bits 18–21 are actually for processor conditions [§2.14].

- 20 Prevent the setting of the Parity Error flag from requesting an interrupt on the channel assigned to the processor.
- 21 Enable the setting of the Parity Error flag to request an interrupt on the channel assigned to the processor.
- 23 Deactivate the priority interrupt system, turn off all channels, eliminate all interrupt requests that have already been accepted but are still waiting, and dismiss all interrupts that are currently being held.
- 24 Request interrupts on channels selected by 1s in bits 29–35, and force the processor to accept them even on channels that are off. There is at most one interrupt on a given channel, and a request is lost if it is made by this means to a channel on which an interrupt is already being held.
- 25 Turn on the channels selected by 1s in bits 29–35 so interrupt requests can be accepted on them.
- 26 Turn off the channels selected by 1s in bits 29–35, so interrupt requests cannot be accepted on them unless made by a CONO PI, with a 1 in bit 24.
- 27 Deactivate the priority interrupt system. The processor can then still accept requests, but it can neither start nor dismiss an interrupt.
- 28 Activate the priority interrupt system so the processor can accept requests and can start, hold and dismiss interrupts.

**CONI PI, Conditions In, Priority Interrupt**



Read the status of the priority interrupt (and several bits of processor conditions) into location *E* as shown.



Channels that are on are indicated by 1s in bits 29–35; 1s in bits 21–27 indicate channels on which interrupts are currently being held. A 1 in bit 28 means the interrupt system is on.

The remaining conditions read by this instruction have nothing to do with the interrupt. Bits 18–20 actually read processor status condition [§2.14] as follows.

- 18 Ac power has failed. The program should save PC, the flags and fast memory in core, and halt the processor.  
The setting of this flag requests an interrupt on the channel assigned to the processor. If the flag remains set for 5 ms, the processor is cleared.
- 19 A word with even parity has been read from core memory. If bit 20 is set, the setting of the Parity Error flag requests an interrupt on the channel assigned to the processor, at which time PC points to the instruction being performed or to the one following it.

Note that PC may point to an interrupt service routine rather than the main program.

**Timing.** The time a device must wait for an interrupt to start depends on the number of channels in use, and how long the service routines are for devices on higher priority channels. If only one device is using interrupts, it need never wait longer than the time required for the processor to finish the instruction that is being performed when the request is made. The maximum time can be considered to be about 15  $\mu$ s for FDVL, but a ridiculously long shift could take over 35  $\mu$ s.

**Special Considerations and Programming Suggestions.** If the interrupt routine uses a UWO it must first save the contents of the pair of locations that will be changed by it in case the interrupted program was in the process of handling a UWO. Hence the routine must save unrellocated location 40 and the location used by the UWO handler instruction to store the PC word. In all other respects, the special considerations and programming suggestions given at the end of the section on the KI10 interrupt hold for the KA10.

2.14 PROCESSOR CONDITIONS

There are a number of internal conditions that can signal the program by requesting an interrupt on a channel assigned to the processor. Condition IO instructions are used to control the appropriate flags and to inspect other internal conditions of interest to the program.

KI10 Processor Conditions

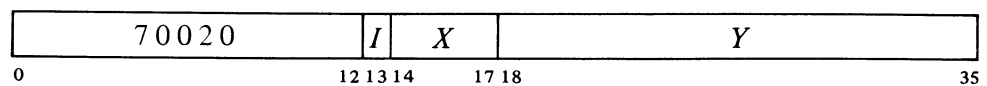
In the KI10, page failures and overflow are handled by trapping, but other internal conditions use the interrupt system. The program can actually assign two channels to the processor – one for error conditions and one specifically for the clock. Control over the Power Failure and Parity Error flags is exercised by a CONO that addresses the priority interrupt system [§2.13]. Control over other conditions and inspection of all are handled by condition IO instructions that address the processor; the CONI also reads some console switches and maintenance functions. The processor also has a data-out instruction through which the program can perform margin checking of the system in both speed and voltage.

The error conditions are generally regarded as important enough to be assigned to the highest priority channel. However for conditions that may be associated with user instructions (a parity error or unanswered memory reference), the common practice is for the error interrupt to switch over to the lowest priority channel by means of a program-set request. Then the time taken to handle the situation, which may well be considerable, cannot interfere with high priority events.

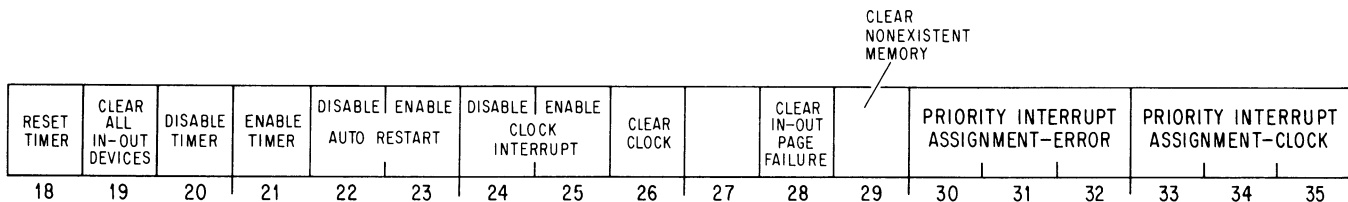
One of the features controlled by the CONO for the processor is the automatic restart after power failure. This restart applies only when the levels on the power mains go below specification while the processor is running, and the power switch is on when power is restored – the machine never begins operation by itself when the operator turns the power switch on or off. Inadequate power, over temperature, etc are indicated by the Power Failure flag. In order for the processor to restart itself, the program must respond in a particular way to the setting of Power Failure. If the program fails to respond properly, there is no restart.

The processor device code is 000, mnemonic APR.

CONO APR, Conditions Out, Arithmetic Processor



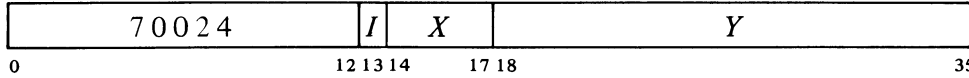
Assign the interrupt channels specified by bits 30–35 of the effective conditions *E* and perform the functions specified by bits 18–29 as shown (a 1 in a bit produces the indicated function, a 0 has no effect).



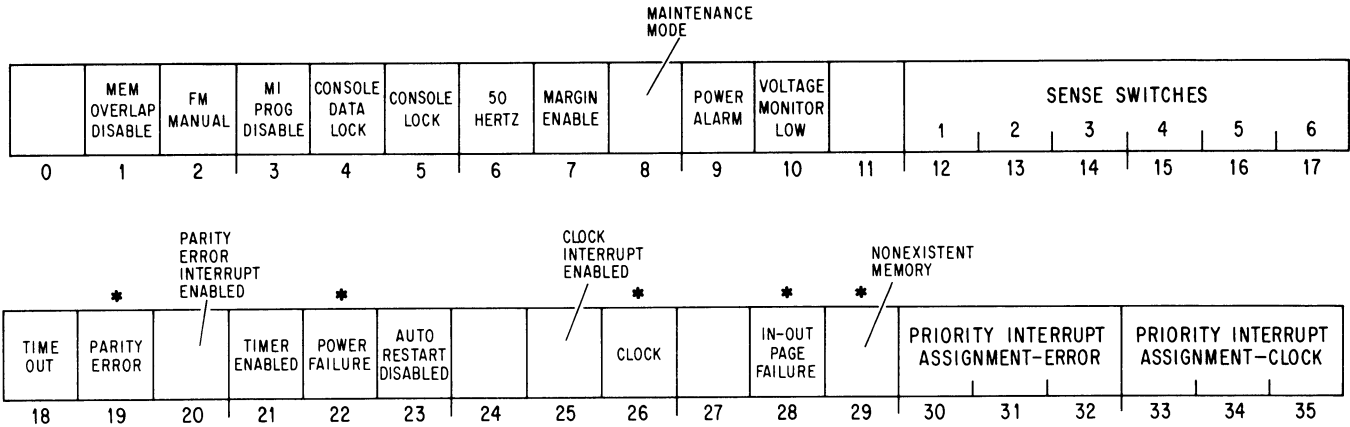
A 1 in bit 19 produces the IO reset signal, which clears the control logic

in all of the peripheral equipment (but affects neither the priority interrupt system nor the processor conditions).

**CONI APR, Conditions In, Arithmetic Processor**



Read the status of the processor (as well as various console switches and maintenance functions) into location *E* as shown.



Interrupts are requested on the error channel (assigned by bits 30–32 of the CONO) by the setting of Power Failure, In-out Page Failure, Nonexistent Memory, and if enabled, Parity Error. The setting of Clock Flag, if enabled, requests an interrupt on the clock channel (assigned by bits 33–35 of the CONO).

Bits 12–17 reflect the states of the console sense switches, which are specifically for operator communication with the program. Bits 1–5 reflect the settings of various console operating switches; for information on these switches refer to Appendix F1. Bits 7–10 are maintenance functions for which the reader should refer to Chapter 10 of the maintenance manual.

6 The system is operating on 50 Hz line power. This is important to the program, not only because some IO devices run slower on 50 Hz, but because the program must compensate for the time difference when using the line frequency clock (bit 26).

18 Bit 21 is 1 and the program has not reset the timer (CONO APR, bit 18) during the last 1.2 seconds (the period of the timer may vary from 1.2 to 1.5 seconds). The setting of this flag clears the processor and the peripheral equipment, and restarts the processor in kernel mode at location 70.

19 A word with even parity has been read from core memory. If bit 20 is 1, the setting of Parity Error requests an interrupt on the error channel [see cautions below].

\*These bits cause interrupts.

The processor does not actually have a maintenance mode — the bit is simply the OR function of a number of console switches, any one of which being on implies that the processor is being operated for maintenance purposes.

The timer provides a restart similar to that following power failure. Running the machine under margins may result in significant logical errors. If the timer is enabled, failure of the program to reset it about every second allows it to time out. The restart instruction should set up PC, which would otherwise be clear.

- 22 Ac power has failed. The program should save PC, the flags, mode information and fast memory in core, and halt the processor. Note that PC may point to an interrupt service routine rather than the main program.

The setting of this flag requests an interrupt on the error channel. After 4 ms the processor is cleared. But at that time, if the power switch is on and the program has cleared Power Failure (CONO PI,400000) and enabled the auto restart (CONO APR, 010000), then when adequate power levels are restored, the processor will resume normal operation by executing the instruction in location 70 in kernel mode.

The restart instruction should set up PC, which would otherwise be clear.

- 26 This flag is set at the ac power line frequency and can thus be used for low resolution timing (the clock has high long term accuracy). If bit 25 is 1, the setting of the Clock flag requests an interrupt on the clock channel.

An interrupt page failure caused by the console address break switch sets this flag instead of producing an address failure [§2.15].

- 28 A page failure has occurred in an interrupt instruction. The setting of this flag requests an interrupt on the error channel.

Note: A page failure in an interrupt instruction is regarded as a fatal error, and it causes an interrupt instead of a page failure trap. The kernel mode program is expected to set up the interrupt instructions so that a page failure simply cannot occur.

PC bears no relation to the unanswered reference if the attempted access originated from a console key function.

- 29 The processor attempted to access a memory that did not respond within 100  $\mu$ s. The setting of this flag requests an interrupt on the error channel [*see cautions below*].

*Programming Cautions.* When handling parity error or nonexistent memory interrupts, the programmer should beware of the following.

Remember that during the grant procedure, the interrupt system is otherwise static and the program continues. Moreover the processor is effectively at the far end of the bus.

◆ Should an error flag be set during an interrupt grant, the processor would handle a lower priority interrupt before getting to the processor interrupt. This means PC may be pointing to a lower level interrupt service routine rather than the program level at which the error occurred.

◆ Even without inadvertent interference from another channel, it is quite likely the processor will perform one or perhaps two more instructions between the time the error flag sets and its interrupt starts. Hence even though PC is at the correct program level, it may well be pointing to the first or second instruction following the one in which the error occurred.

◆ A processor error interrupt that switches over to a lower priority channel should not return to the interrupted program, as the error may simply recur, producing a second processor interrupt before the error-handling interrupt for the first. This could happen because PC is actually pointing to the offending instruction, but beyond that, one error often begets another — consider the case of PC counting into a nonexistent memory.

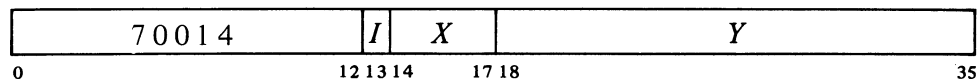
In any event, it is generally not worthwhile to return to any program without first finding out what has gone wrong.

◆ The error may have originated from a console key function, and thus be hidden from any investigation by the program.

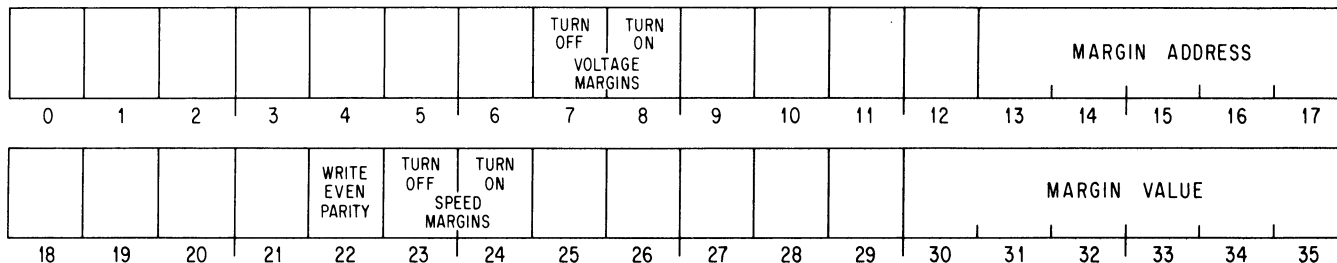


**DATA0 APR, Maintenance Data Out, Arithmetic Processor**

This instruction is primarily for maintenance, for which further information is given in Chapter 10 of the KI10 Maintenance Manual.



Supply diagnostic information and perform diagnostic functions according to the contents of location *E* as shown.



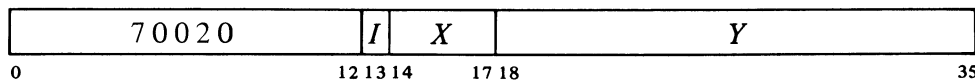
The margin value supplied by bits 30–35 of the output word is translated to a voltage in the range 0–10 volts by a D-A converter, whose output is available at pin 2S02V2. Running margins requires a slowdown capacitor in the converter. But turning off the margin enable switch cuts out the capacitor, making the converter output suitable for external use, such as for operating audio equipment to play Bach or rock or Bacharach.

**KA10 Processor Conditions**

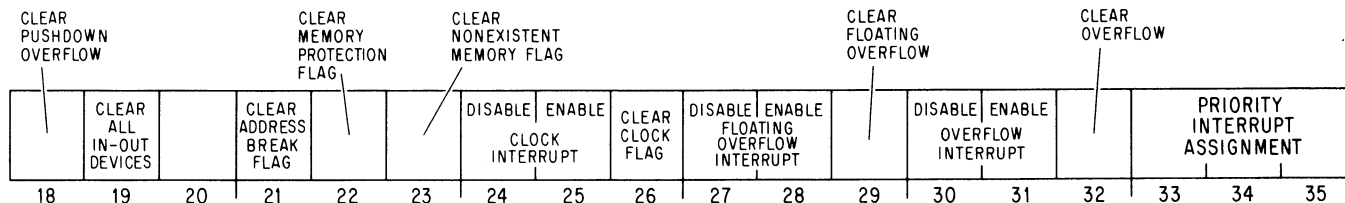
All KA10 processor conditions act through the interrupt on a single channel assigned to the processor. Flags for power failure and parity error are handled by the condition IO instructions that address the priority interrupt system [§2.13]. The remaining flags are handled by condition instructions that address the processor. Its device code is 000, mnemonic APR.

Most of these conditions are generally regarded as important enough to be assigned to the highest priority channel. Except in the case of a power failure however, the common practice is for the processor interrupt to switch over to the lowest priority channel by means of a program-set request. Then the time taken to handle the situation, which may well be considerable, cannot interfere with high priority events.

**CONO APR, Conditions Out, Arithmetic Processor**



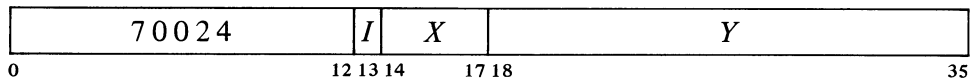
Assign the interrupt channel specified by bits 33–35 of the effective conditions *E* and perform the functions specified by bits 18–32 as shown (a 1 in a bit produces the indicated function, a 0 has no effect).



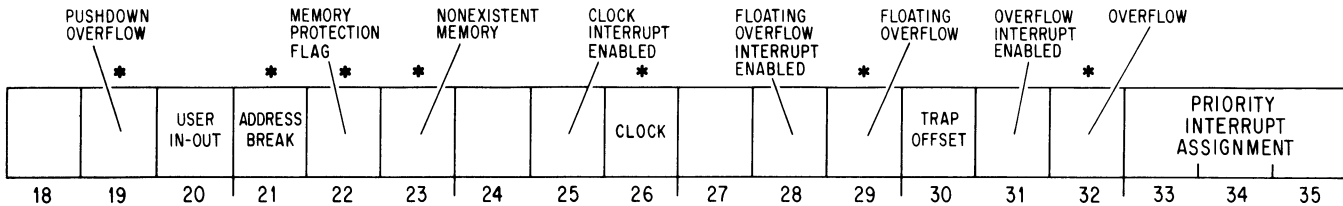
Enabling a particular flag to interrupt means that henceforth the setting of the flag will request an interrupt on the channel assigned (by bits 33–35) to the processor. Disabling prevents the flag from triggering a request.

A 1 in bit 19 produces the IO reset signal, which clears the control logic in all of the peripheral equipment (but affects neither the priority interrupt system, nor the processor flags cleared by this instruction or CONO PI.).

**CONI APR, Conditions In, Arithmetic Processor**



Read the status of the processor into the right half of location E as shown (all interrupt requests are made on the channel assigned to the processor).



\*These bits cause interrupts on the channel assigned to the processor, as do Power Failure and Parity Error, bits 18 and 19 read by a CONI PI.

With the possible exception of an illegal memory reference on an instruction fetch, if the highest priority active channel is assigned to the processor, then the occurrence of any processor interrupt condition is guaranteed to produce a processor interrupt with no lower priority interrupt intervening between it and the program level at which the processor condition occurred. The actual relationship between PC and the instruction associated with a given condition is as stated in its description.

- 19 Pushdown Overflow – in a PUSH or PUSHJ the count in AC left reached zero; or in a POP or POPJ the count reached -1. The setting of this flag requests an interrupt, at which time PC points to the instruction following that in which the overflow occurred. The location of the offending instruction is implied by PC for PUSH or POP, is indicated by the last item in the stack for PUSHJ, but is indeterminate for POPJ.
- 20 User In-out – even if the processor is in user mode, there are no instruction restrictions (but memory restrictions still apply) [ §2.16 ].
- 21 Address Break – while the console address break switch was on, the processor requested access to the memory location specified by the address switches and the memory reference was for the purpose selected by the address condition switches as follows:

The instruction switch was on and access was for retrieval of an instruction (including an instruction executed by an XCT or contained in an interrupt location or a trap for an unimplemented operation) or an address word in an effective address calculation.

PC bears no relation to the break if the access was requested for a console key function.

The data fetch switch was on and access was for retrieval of an operand (other than in an XCT).

The write switch was on and access was for writing a word in memory, other than in a read-modify-write.

The setting of this flag requests an interrupt, at which time PC points to the instruction that was being executed or to the one following it.

- 22 Memory Protection – a user program attempted to access a memory location outside of its area or to write in a write-protected part of its area and the user instruction was terminated at that time. The setting of this flag requests an interrupt, at which time PC points either to the instruction that caused the violation or to the one following it, unless the illegal reference was for fetching an instruction. In this exceptional case it is possible for a lower level interrupt to occur between the violation and its interrupt, even with the processor assigned to the highest priority active channel.
- 23 Nonexistent Memory – the processor attempted to access a memory that did not respond within 100  $\mu$ s. The setting of this flag requests an interrupt, at which time PC points either to the instruction containing the unanswered reference or to the one following it.
- 26 Clock – this flag is set at the ac power line frequency and can thus be used for low resolution timing (the clock has high long term accuracy). If bit 25 is set, the setting of the Clock flag requests an interrupt.
- 29 Floating Overflow – this is one of the flags saved in a PC word, and the conditions that set it are given at the beginning of §2.9. If bit 28 is set, the setting of Floating Overflow requests an interrupt, at which time PC points to the instruction following that in which the overflow occurred.
- 30 Trap Offset – the processor is using locations 140–161 for unimplemented operation traps and interrupt locations.
- 32 Overflow – this is one of the flags saved in a PC word, and the conditions that set it are given at the beginning of §2.9. If bit 31 is set, the setting of Overflow requests an interrupt, at which time PC points to the instruction following that in which the overflow occurred.

This flag can also be set by an instruction executed from the console while the USER MODE light is on, in which case PC bears no relation to the violation.

PC bears no relation to the unanswered reference if the attempted access originated from a console key function.

#### CAUTION

For an address break, a memory protection violation, a parity error, or a nonexistent memory, a processor error interrupt that switches over to a lower priority channel should not return to the interrupted program, as the processor will fetch the next user instruction before it accepts the program-set interrupt request. This makes it very likely that the same error will recur, producing a loop between the processor interrupt and the interrupted program.

## 2.15 KI10 PROGRAM AND MEMORY MANAGEMENT

General information about the machine modes and paging procedures is given in Chapter 1, in particular at the end of the introductory remarks and at the end of §1.3. Here we are concerned principally with the special instructions the Monitor uses to operate the system, the special effects that ordinary instructions have in executive mode, and certain hardware procedures, in particular paging and page failures, that are necessary for an understanding of executive programming.

**User Programming.** As far as user programming is concerned, all of the necessary information has already been presented. For convenience however we list here the rules the user must observe. [*Refer to the Monitor manual for further information including use of the Monitor for input-output.*]

- ◆ If possible, limit your memory needs to 32K, using addresses 0–37777 and 400000–437777, to gain the savings afforded by having the status of a “small user”. There are no restrictions of any kind on addresses 0–17 as these are in fast memory and are available to all users (even though page 0 may otherwise be inaccessible).
- ◆ If an area of memory is write-protected, *eg* for a reentrant program shared by several users, do not attempt to store anything in it. In particular do not execute a JSR or JSA into a write-protected page.
- ◆ Use the MUUO codes 040–077 only in the manner prescribed in the Monitor manual. In general, unless they are prescribed for special circumstances, code 000 and the unassigned codes should not be used.
- ◆ Do not use HALT (JRST 4,) or JEN (JRST 12, (specifically JRST 10,)).
- ◆ Unless User In-out is set do not give any IO instruction with device code less than 740. The program can determine if User In-out is set by examining bit 6 of the PC word stored by JSR, JSP or PUSHJ.
- ◆ If your public program has the use of concealed programs, do not reference a location in a concealed page for any purpose except to fetch an instruction from a valid entry point, *ie* a location containing a JRST 1,.

The user can give a JRSTF (JRST 2,) but a 0 in bit 5 of the PC word does not clear User (a program cannot leave user mode this way); and a 1 in bit 6 does not set User In-out, so the user cannot void any of the instruction restrictions himself. Note that a 0 in bit 6 will clear User In-out, so a user can discard his own special privileges. Similarly a 1 in bit 7 sets Public, but a 0 does not clear it, so a public program cannot enter concealed mode this way.

The above rules are the result of KI10 hardware characteristics. But in a real sense many of these rules are actually transparent to the user, in particular the whole paging setup is invisible. Although the hardware allows for user virtual address spaces that are scattered and/or very large (*eg* larger than available physical core), the actual constraints will be dictated by the particular Monitor and the system manager. It may be desirable (for compatible operation with KA10 systems) to enforce a two-segment virtual address space that mimics the one imposed by the KA10 hardware. In any case the user must write a sensible program, which can be handled easily and cheaply by the system; if he uses addresses a few to a page all over memory, his program can be run but will require a much larger amount of core than necessary or cause excessive page swapping.

### Paging

All of memory both virtual and physical is divided into pages of 512 words each. The virtual memory space addressable by a program is 512 pages; the locations in virtual memory are specified by 18-bit addresses, where the left nine bits specify the page number and the right nine the location within the page. Physical memory can contain 8192 pages and requires 22-bit addresses, where the left thirteen bits specify the page number. The hardware maps the virtual address space into a part of the physical address space by transforming the 18-bit addresses into 22-bit addresses. In this mapping the right nine bits of the virtual address are not altered; in other words a given location in a virtual page is the same location in the corresponding physical page. The transformation maps a virtual page into a physical page by substituting a 13-bit physical page number for the 9-bit virtual page number. The mapping procedure is carried out automatically by the hardware, but the page map that supplies the necessary substitutions is set up by the kernel mode program. Each word in the map provides information for mapping two consecutive pages with the substitution for the even numbered page in the left half, the odd numbered page in the right half.

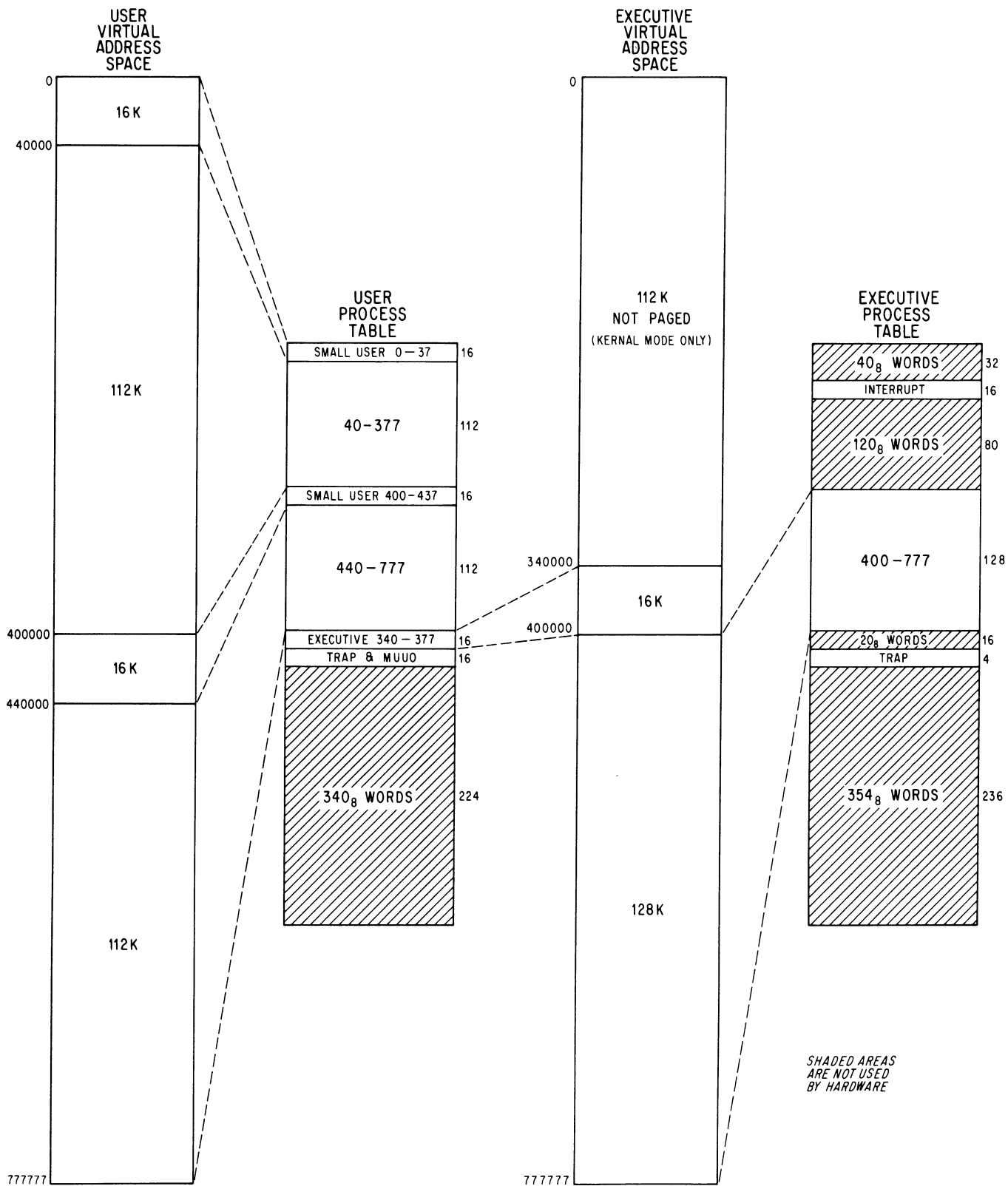
The paging hardware contains two 13-bit registers that the Monitor loads to specify the physical page numbers of the user and executive process tables. To retrieve a map word from a process table, the hardware uses the appropriate base page number as the left thirteen bits of the physical address and some function of the virtual page number as the right nine bits. *Eg* the entire user space of 512 virtual pages at two mappings per word requires a page map of just half a page, and this is the first half page in the user process table. Thus locations 0–377 in the table hold the mappings for pages 0 and 1 to 776 and 777. To find the desired substitution from the 9-bit virtual page number, the hardware uses the left eight bits to address the location and the right bit to select the half word (0 for left, 1 for right). If the Monitor specifies a program as being a small user, that program is limited to two 16K blocks with addresses 0–37777 and 400000–437777. This is pages 0–37 and 400–437, and the mappings are in locations 0–17 and 200–217 in the page map.

The executive virtual address space is also 256K but the first 112K are not paged — in other words any address under 340000 given in kernel mode addresses one of the first 112K locations in physical memory directly. The other 144K is paged for supervisor or kernel mode anywhere into physical memory. For this there are two maps. The map for the second half of the virtual address space uses the same locations in the executive process table as are used in the user process table for the user map (locations 200–377 for pages 400–777). The map for the remaining 16K in the first half of the executive virtual address space is in the *user* process table, the mappings for pages 340–377 being in locations 400–417. Thus the Monitor can assign a different set of thirty-two physical pages (the per-process area) for its own use relative to each user.

The illustrations on the next two pages show the organization of the virtual address spaces, the process tables and the mappings for both user and executive. The first illustration gives the correspondence between the various parts of each address space and the corresponding parts of the page

Actually page 0 has only 496 locations using addresses 20–777, as addresses 0–17 reference fast memory, which is unrestricted and available to all programs. (In general a user cannot reference the first sixteen core locations in his virtual page 0.) Throughout this discussion it is assumed that all references are to core and are not made by an instruction executed by an executive XCT [see below].

Thus when switching from one user to another, the Monitor need change only the user process table. This single substitution can make whatever change is necessary in the executive address space for a particular user.



VIRTUAL ADDRESS SPACE AND PAGE MAP LAYOUT

USER PROCESS TABLE

0	USER PAGE 0	USER PAGE 1
17	USER PAGE 36	USER PAGE 37
20	USER PAGE 40	USER PAGE 41
<i>AVAILABLE TO SOFTWARE IF SMALL USER</i>		
177	USER PAGE 376	USER PAGE 377
200	USER PAGE 400	USER PAGE 401
217	USER PAGE 436	USER PAGE 437
220	USER PAGE 440	USER PAGE 441
<i>AVAILABLE TO SOFTWARE IF SMALL USER</i>		
377	USER PAGE 776	USER PAGE 777
400	EXECUTIVE PAGE 340	EXECUTIVE PAGE 341
417	EXECUTIVE PAGE 376	EXECUTIVE PAGE 377
420	USER PAGE FAILURE TRAP INSTRUCTION	
421	USER ARITHMETIC OVERFLOW TRAP INSTRUCTION	
422	USER PUSHDOWN OVERFLOW TRAP INSTRUCTION	
423	USER TRAP 3 TRAP INSTRUCTION	
424	MUUO STORED HERE	
425	PC WORD OF MUUO STORED HERE	
426	EXECUTIVE PAGE FAILURE WORD	
427	USER PAGE FAILURE WORD	
430	KERNEL NO TRAP NEW MUUO PC WORD	
431	KERNEL TRAP NEW MUUO PC WORD	
432	SUPERVISOR NO TRAP NEW MUUO PC WORD	
433	SUPERVISOR TRAP NEW MUUO PC WORD	
434	CONCEALED NO TRAP NEW MUUO PC WORD	
435	CONCEALED TRAP NEW MUUO PC WORD	
436	PUBLIC NO TRAP NEW MUUO PC WORD	
437	PUBLIC TRAP NEW MUUO PC WORD	
440	<i>AVAILABLE TO SOFTWARE</i>	
777		

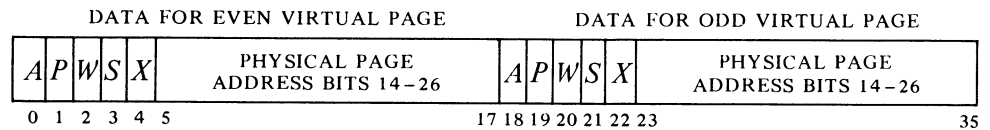
EXECUTIVE PROCESS TABLE

0	<i>AVAILABLE TO SOFTWARE</i>	
37		
40	EXECUTIVE LUUO STORED HERE	
41	LUUO HANDLER INSTRUCTION	
42		
57	STANDARD PRIORITY INTERRUPT INSTRUCTIONS	
60	<i>AVAILABLE TO SOFTWARE</i>	
177		
200	EXECUTIVE PAGE 400	EXECUTIVE PAGE 401
377	EXECUTIVE PAGE 776	EXECUTIVE PAGE 777
400	<i>AVAILABLE TO SOFTWARE</i>	
417		
420	EXECUTIVE PAGE FAILURE TRAP INSTRUCTION	
421	EXECUTIVE ARITHMETIC OVERFLOW TRAP INSTRUCTION	
422	EXECUTIVE PUSHDOWN OVERFLOW TRAP INSTRUCTION	
423	EXECUTIVE TRAP 3 TRAP INSTRUCTION	
424	<i>AVAILABLE TO SOFTWARE</i>	
777		

PROCESS TABLE CONFIGURATION

map for it. The second illustration lists the detailed configuration of the process tables. Any table locations not used by the hardware can be used by the Monitor for software functions. Note that the numbers in the half locations in the page map are the virtual pages for which the half words give the physical substitutions. Hence location 217 in the user page map contains the physical page numbers for virtual pages 436 and 437.

Although the virtual space is always 256K by virtue of the addressing capability of the instruction format, the Monitor usually limits the actual address space for a given program by defining only certain pages as accessible. The Monitor also specifies whether each page is public or not and writeable or not. Each word in the page map has this format to supply the necessary information for two virtual pages.



Bits 5-17 and 23-35 contain the physical page numbers for the even and odd numbered virtual pages corresponding to the map location that holds the word. The properties represented by 1s in the remaining bits are as follows.

<i>Bit</i>	<i>Meaning of a 1 in the Bit</i>
<i>A</i>	Access allowed
<i>P</i>	Public
<i>W</i>	Writeable (not write-protected)
<i>S</i>	Software (not interpreted by the hardware)
<i>X</i>	Reserved for future use by DEC (do not use)

**Associative Memory.** If the complete mapping procedure described above were actually carried out in every instance, the processor would require two memory references for every reference by the program. To avoid this the paging hardware contains a 32-word associative memory, in which it keeps the more recently used mappings for both the executive and the current user. Each word is divided into two parts with one part containing a virtual page number specified by the program and the other containing the corresponding physical page number as determined from the page map. Hence the associative memory is a page table made up of a list of virtual pages and a list of physical pages, each with thirty-two corresponding locations. In the virtual list, each entry contains a 9-bit virtual page number, a single bit that indicates whether the specified page is in the user or executive address space, and a bit that indicates whether the entry is valid or not (it is not suitable to clear a location as 0 is a perfectly valid page number). Each corresponding entry in the physical list contains a 13-bit physical page number and the *P*, *W* and *S* bits from the map half word for that page. The *A* bit is not needed in the table as the mapping is not entered into the table at all if the page is not accessible.

At each reference the hardware compares the page number supplied by

There is no requirement that the accessible space be continuous — it can be scattered pages. The convention however is for the accessible space to be in two continuous virtual areas, low and high, beginning respectively at locations 0 and 400000. The low part is generally unique to a given user and can be used in any way he wishes. The (perhaps null) high part is a reentrant area, which is shared by several users and is therefore write-protected. The small user configuration is consistent with this arrangement.

The program can inspect the contents of the page table by using the MAP instruction and IO instructions that address the paging hardware [see below].



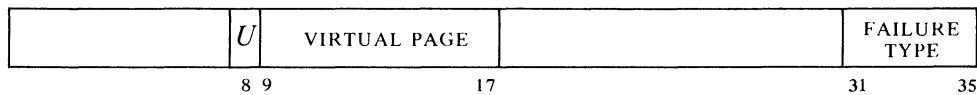
the program with those in the virtual part of the page table. If there is a match for the appropriate address space, the corresponding entry in the physical list is used as the left thirteen bits in the physical address (provided of course that the reference is allowable according to the *P* and *W* bits). If there is no match, the hardware makes a memory reference to get the necessary information from the page map and enters it into the page table at the location specified by a reload counter. This counter is incremented whenever it is used to reload the table, and also whenever the location to which it points is used for a mapping. Hence the counter tends to stay away from locations containing the page numbers most frequently referenced.

This memory reference is referred to as a “page refill cycle.”

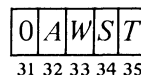
### Page Failure

A page failure that occurs during an interrupt instruction terminates the instruction and sets the In-out Page Failure flag, requesting an interrupt on the error channel assigned to the processor. In all other circumstances, if the paging hardware cannot make the desired memory reference, it terminates the instruction immediately without disturbing memory, the accumulators or PC, places a page fail word in the user process table, and causes a page failure trap. If the attempted reference is in user virtual address space, the page fail word is placed in location 427 of the user process table, and the processor executes the trap instruction in location 420 of the same table. If the attempted reference is in executive virtual address space, the page fail word is placed in location 426 of the *user* process table, and the processor executes the trap instruction in location 420 of the *executive* process table. The trap instruction is executed in the same address space in which the failure occurred. The page fail word supplies this information.

When a page failure trap instruction is performed, PC points to the instruction that failed (or to an XCT that executed it), unless the failure occurred in an overflow trap instruction, in which case PC points to the instruction that overflowed. After taking care of the failure, the processor can always return to the interrupted instruction. Either the instruction did not change anything, or the failure was in the second part of a two-part instruction, where First Part Done being set prevents the processor from repeating any unwanted operations in the first part.



IF BIT 31 IS 0, BITS 31-35 HAVE THIS FORMAT



Whether the violation occurred in user or executive virtual address space is indicated by a 1 or a 0 in bit 8. If bit 31 is 1, the number in bits 31-35 ( $\geq 20$ ) indicates the type of “hard” failure as follows.

- 23 Address failure – this is a simulated page failure caused by the satisfaction of an address condition selected from the console. It indicates that while the console address break switch was on and the Address Failure Inhibit flag was clear (bit 8 of the PC word), the processor initiated a page check for access to the memory location that was specified by the paging and address switches and for which a comparison was enabled, and the intended memory reference was for the purpose selected by the address condition switches as follows:

The instruction fetch switch was on and the requested access was for retrieval of an ordinary instruction, including an instruction executed by an XCT or an LUUO (address 41).

Since a user page failure trap instruction is executed in user address space, the Monitor should be careful not to have the trap instruction do indirect addressing that might cause another page failure.

Whether or not a comparison can be made is a function of the settings of the paging switches [Appendix F1] and the state of the User Address Compare Enable flag [see below].

Virtual addresses are supplied to the paging hardware via the address bus. An inadvertent failure occurs when the bus is not used for an access, but it accidentally contains the number set into the address switches. The data fetch switch also catches the attempt to retrieve a dispatch interrupt instruction or inadvertently a standard interrupt instruction, but the page failure sets the In-out Page Failure flag instead of resulting in a trap for an address failure.

Using this flag, the Monitor can return to a user instruction that caused an address failure and “set by it.”

Tests for hard page failures are actually made in the order given here.

The type of reference implies nothing about the cause of failure — it indicates only the reason the failed reference was being made.

In a soft page failure, the mapping entry for the page is removed from the page table on the assumption that the Monitor will change it. When the instruction is restarted, the hardware must go to the page map to get a new entry for the page table.

The data fetch switch was on and the requested access was for retrieval of an address word in an effective address calculation or read-only retrieval of an operand (other than in an XCT). This switch can also cause a failure inadvertently on the retrieval of a trap instruction or a PC word in an MUUO.

The write switch was on and the requested access was for writing, either write-only or read-modify-write, including writing by an LUUO (address 40). This switch also causes a failure on the first write in an MUUO if the address switches contain the effective address of the MUUO (even though that address is not used for the access), and can cause a failure inadvertently on the second write.

The Address Failure Inhibit flag, which can be set only by a JRSTF or MUUO, prevents an address failure during the next instruction — the completion of the next instruction automatically clears it. If an interrupt or trap intervenes, the flag has no effect and it is saved and cleared if the PC word is saved. If it is not saved, it affects the instruction following the interrupt or trap. Otherwise it affects the instruction following a return in which it is restored with the PC word.

- 22 Page refill failure — this is a hardware malfunction. The paging hardware did not find the virtual page listed in the page table, so it loaded paging information from the page map into the table but still could not find it.
- 20 Small user violation — a small user has attempted to reference a location outside of the limited small user address space.
- 21 Proprietary violation — an instruction in a public page has attempted to reference a concealed page or transfer control into a concealed page at an invalid entry point (one not containing a JRST 1.).

If the violation is not one of these, then bits 31–35 have the format shown above where *A*, *W* and *S* are simply the corresponding bits taken from the map half word for the page, and *T* indicates the type of reference in which the failure occurred — 0 for a read reference, 1 for a write or read-modify-write reference.

The page fail trap instruction is set by the Monitor to transfer control to kernel mode. After rectifying the situation, the Monitor returns to the interrupted instruction, which starts over again from the beginning. Even a two-part instruction that has been stopped by a failure in the second part is redone properly, provided the Monitor restores the First Part Done flag.

Note that a failure does not necessarily imply that anything is “wrong”. The virtual address space of even a small user is 32K words, which may well be more than is needed in a given run. Hence the Monitor may have only ten or twenty pages of the user program in core at any given time, and these would be the virtual pages indicated as accessible. When the user attempts to gain access to a page that is not there (a virtual page indicated in the page map as inaccessible), the Monitor would respond to the page failure by

bringing in the needed page from the drum or disk, either adding to the user space or swapping out a page the user no longer needs.

The same situation exists for writeability. When bringing in a user program, the Monitor would ordinarily indicate as writeable only the buffer area and other pages that will definitely be altered. Then in response to a write failure, the Monitor makes the page writeable and indicates to itself (perhaps by means of the software bit in the page map) that that page has in fact been altered. When the user is done, the Monitor need write only the altered pages back onto the drum.

### Monitor Programming

The kernel mode program is responsible for the overall control of the system. It is the only program that has access to any of physical core unpaged and that has no instruction restrictions. The kernel program handles all in-out for the system and must set up the page maps, trap locations, interrupt locations and the like. The supervisor program labors under the same instruction restrictions as the user but has no way of bypassing them — they always apply. Supervisor mode is limited to the 144K paged part of the executive address space, although within that space it can read but not alter concealed pages (the kernel program supplies data tables of all kinds to the supervisor program, and the latter cannot affect them). The supervisor can give a JRSTF that clears Public provided it is also setting User; in other words the supervisor can return control to a concealed program but cannot enter kernel mode by manipulating the flags. The PC words supplied by MUUOs can manipulate the flags in any way, switching arbitrarily from one mode to another, but these are in the process table and assumed to be under control solely of kernel mode.

For accumulator, index register and fast memory references, the Monitor automatically uses fast memory block 0. For each user, the kernel mode program must assign a block. The usual procedure is to assign blocks 2 and 3 to individual user programs on a semipermanent basis for special applications and to assign block 1 to all other users. In this way the Monitor need not store blocks 2 and 3 when the special users are not running, and it need not store block 1 when it takes over control from an ordinary user temporarily. When switching from one user to another, the Monitor usually stores the first user's accumulators in his shadow area — this is locations 0–17 in user virtual page 0, an area not generally accessible to the user at all — and loads the new user's accumulators from his shadow area, where they were stored after the last time the new user ran.

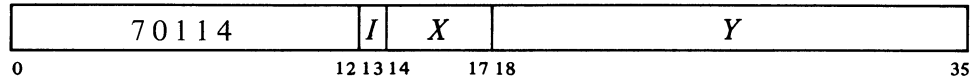
Even while User is set, the interrupt instructions are not part of the user program and are thus subject only to executive restrictions. As interrupt instructions, JSR, JSP and PUSHJ automatically take the processor out of user mode to jump to an executive service routine. An MUUO can also be used.

The paging hardware has one non-IO instruction and two condition IO instructions primarily for diagnostic purposes. Otherwise control over the system is exercised by data IO instructions. The device code for the paging hardware is 010, mnemonic PAG.

If the Monitor shared block 0 with any users, it would have to store the user accumulators even when taking control only temporarily.

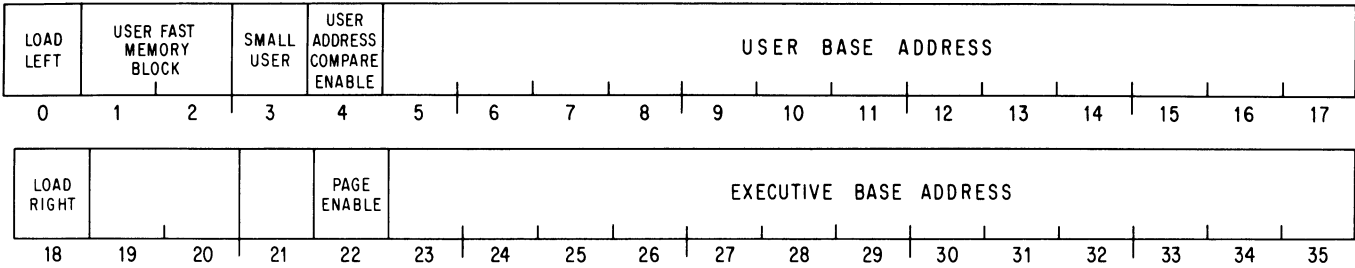
The page failure and overflow trap instructions are executed in the user address space if caused by the user.

**DATAO PAG, Data Out, Paging**



Invalidating all data in the associative memory means setting the Word Empty bit in each location to indicate that the rest of the word is meaningless and should not be used.

Invalidate all data in the associative memory, and set up the paging hardware according to the contents of location *E* as shown.



Bits 0 and 18 are change bits. If bit 0 is 0, ignore the rest of the left half word. But if bit 0 is 1, load bits 5–17 into the user base register to select the user process table, select the fast memory block specified by bits 1 and 2 for the user, limit the address space to that of a small user if bit 3 is 1, and enable address comparison if bit 4 is 1.

Similarly if bit 18 is 0, ignore the rest of the right half word. Otherwise load bits 23–35 into the executive base register to select the executive process table, and enable executive paging if bit 22 is 1. For normal operation of the system, bit 22 must be 1. A 0 in this bit disables overflow traps, and disables executive paging so there is no supervisor mode and no executive virtual addressing – in other words an executive mode program automatically runs in kernel mode with all access in the first 256K of physical memory unpagged.

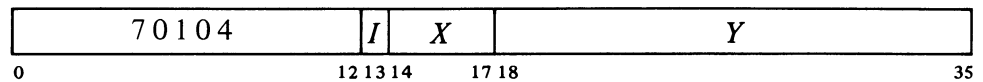
The Address Compare Enable bit functions in conjunction with the console paging switches, as explained in Appendix F1.

An executive mode program that does not set bit 22 and avoids other special KI10 features will run on a KA10 as well. This is useful for hardware diagnostics and bootstrap loaders [see *readin mode*, §2.12].

**NOTE**

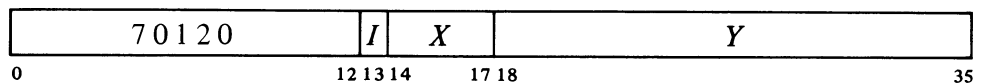
Neither turning on power nor pressing the reset switch invalidates the data in the associative memory. Therefore, after power has been off, the starting kernel mode program must do a DATAO PAG, to clear the associative memory of random data before entering executive or user paged address space.

**DATAI PAG, Data In, Paging**

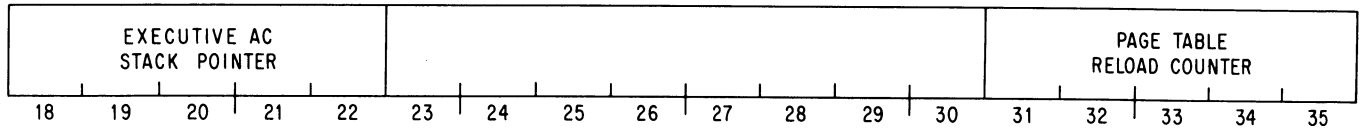


Read the status of the paging hardware into location *E*. The information read is the same as that supplied by a DATAO (bits 0 and 18 are 0).

**CONO PAG, Conditions Out, Paging**

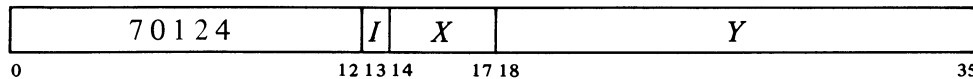


Load the executive stack pointer from bits 18–22 and the page table reload counter from bits 31–35 of the effective conditions *E* as shown.

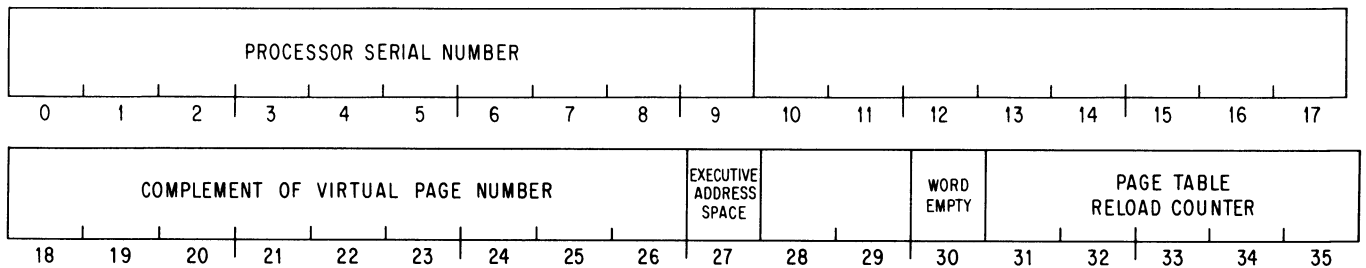


The executive stack pointer specifies a block of sixteen locations in the user process table by supplying the left five bits for a 9-bit address that references a location in the table; this function is used only for accessing stacked fast memory blocks in an instruction executed by an executive XCT [see below]. Loading the reload counter causes it to point to the specified location in the page table.

**CONI PAG, Conditions In, Paging**



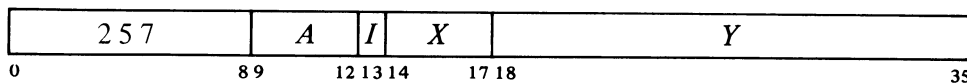
Read the processor serial number, the page table reload counter, and the contents of the location in the virtual page table specified by the counter into the right half of location *E* as shown.



Note that bits 18–26 contain the complement of the virtual page number in the selected location. A 1 in bit 27 indicates the page is in the executive address space; a 1 in bit 30 means the information in bits 18–27 is invalid.

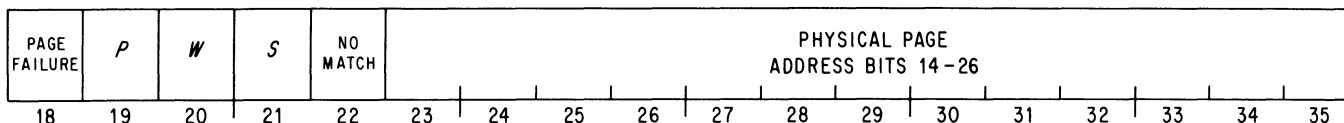
It is possible for the reload counter to change between the CONI and the CONO, so the CONI might read a different location than was selected by the CONO.

**MAP Map an Address**



Note that unlike all other instructions since §2.10, this is not an IO instruction.

Map the virtual effective address *E* and place the resulting map data in AC right in the same format as it is in the page map, *ie* bits *P*, *W* and *S* in bits 19–21 and the physical page number in bits 23–35. Clear AC left.



These three instructions can be used to inspect the contents of the associative memory. The CONO selects a location, the CONI reads the contents of the virtual-page part of that location, and an MAP that addresses the specified virtual page reads the contents of the physical-page part of that location.

This instruction cannot produce a page failure, but if a page failure would have resulted had an ordinary instruction in the same mode attempted to write in location *E*, place a 1 in AC bit 18. If no match can be made by the paging hardware, place a 1 in bit 22. This results in four possible situations as a function of the states of bits 18 and 22.

<i>Bit 18</i>	<i>Bit 22</i>	<i>Meaning</i>
0	0	AC right contains valid map data.
0	1	There is no page failure but also no match, so the instruction must have made an unmapped reference – perhaps to fast memory or to the unpagged area in kernel mode.
1	0	There is a page failure but the map data is correct as a match exists.
1	1	There is a page failure, and since there is no match, the failure must have resulted from the instruction referencing an inaccessible page or from some prior failure (such as a page refill malfunction). Hence AC right contains invalid information.

### Executive XCT

Ordinarily an instruction in a user program is performed entirely in user address space and an instruction in the executive program is performed entirely in executive address space. In order to facilitate communication between Monitor and users, the XCT instruction allows the executive to execute instructions whose memory operand references can cross over the boundary between user and executive address spaces.

It is very important to note that the only difference between an instruction executed by an executive XCT and an instruction performed in normal circumstances is in the way the memory operand references are made. There is no difference in the XCT itself. Everything in the XCT is done in executive address space, and the instruction fetched by the XCT is fetched in executive space. Moreover, in the executed instruction all effective address calculation and accumulator references are in executive space. If the instruction makes no memory operand references, as in a jump, shift or immediate mode instruction, its execution differs in no way from the normal case. The only difference is in *memory operand references*.

Control over the special effects of the executed instructions is determined by the User In-out flag (whose implied meaning is confined to user mode) and bits 11 and 12 of the *A* portion of the XCT instruction word (in user mode *A* is ignored). If the *A* bits are both 0, the XCT acts as described in §2.9, and the executed instruction differs in no way from the normal case.

Read the next four paragraphs very carefully (reading them two or three times is highly recommended).

But if these bits are not both 0, then some memory operand references are made to *user* virtual address space, where the type of reference is determined by the *A* bits and the type of memory is selected by User In-out. With this flag set, the *A* bits affect both core memory and fast memory references, whereas with User In-out clear, the *A* bits affect only fast memory references. For the memory operand references selected by User In-out, the effect of 1s in bits 11 and 12 is as follows: a 1 in bit 12 causes the executed instruction to perform all selected read and read-modify-write memory operand references to be performed in user virtual address space; a 1 in bit 11 causes all selected memory operand write references to be performed in user space; and 1s in both bits cause all types of selected memory operand references in the executed instruction to be performed in user space.

The meaning of user space is obvious in terms of core memory references, but not so for fast memory. When User In-out is set, the user space for fast memory references depends on which fast memory block is currently selected for the user. If block 0 is selected, fast memory operand references of the types specified by bits 11 and 12 are made to the user shadow area. If some other block is selected, the specified fast memory references are made to the selected block.

If User In-out is clear, all core memory references are in executive address space. Fast memory references of the types specified by bits 11 and 12 are made to the user process table, in particular to that set of sixteen locations specified by the executive stack pointer. The pointer is given by a CONO PAG,.

#### *User Space Fast Memory References*

<i>User In-out</i>	<i>User Fast Memory Block Selected</i>	
	<i>Zero</i>	<i>Nonzero</i>
1	Shadow area	Selected block
0	AC stack	AC stack

There is another flag that plays a role in the execution of instructions by an executive XCT. This is Disable Bypass, bit 0 of the PC word. When Disable Bypass is clear, a bypass in the logic allows an executed instruction to access the concealed user area from supervisor mode. With the flag set, an attempt to do this results in a page failure. Generally the new MUUO PC word would set this flag when the Monitor is being called from public mode, so the concealed area can be accessed only when such access is requested by the concealed program.

**Individual Instruction Effects.** The effects of execution by an executive XCT on different types of instructions is as follows.

◆ Instructions without memory operand references are not affected. This includes shifts, jumps, immediate mode instructions, CONSO, CONO, and even an XCT. In fact not only is an executive XCT not affected when executed by an executive XCT, but the first destroys any effect the second would otherwise have on a third instruction (in other words, a pair of executive XCTs is equivalent to a pair of ordinary XCTs).

- ◆ Instructions that refer to one memory location for reading only or reading and writing are controlled by the read bit (MOVE, MOVES, ADDM, AOS). The read bit controls writing when the write is done to the same location as the read, whether the memory references are done as a single cycle including both read and write or as separate read and write cycles.
- ◆ Instructions that refer to one memory location for writing only are controlled by the write bit (MOVEM, MAP, HRLZM).
- ◆ Instructions that refer to two different memory locations are controlled by the read bit in the read part of the instruction and by the write bit in the write part (BLT, PUSH).
- ◆ BLKI and BLKO are controlled by the write bit and the read bit respectively. The pointer reference is done in the same address space as the data transfer.
- ◆ In byte instructions all pointer calculations are done in executive address space. The read and write bits affect only the second part, *ie* the load or deposit.

**Philosophy.** The purpose of the executive XCT is to facilitate the handling of user requirements by the Monitor, but the selection made by User In-out of the references affected by the read and write bits is to allow the Monitor to make recursive calls to itself, *ie* to perform MUUOs in the process of carrying out an MUUO given by the user. Specifically the state of User In-out differentiates between the Monitor response directly to the user MUUO and its response to its own MUUOs.

The new PC word of an MUUO from the user would set User In-out so that core memory references can be made across the user-executive boundary, and fast memory references can be made to the user AC block. The point in choosing between the shadow area and the selected block if not block 0 is to reference the information that was held in the user AC block before the Monitor took over. If the user shared block 0 with other users and the Monitor, the Monitor will have saved his ACs in the shadow area of his address space. The other AC blocks are not disturbed when the Monitor takes over temporarily, so the Monitor need not save them and they will still hold the user information.

If in the course of carrying out a user MUUO, the Monitor should itself give an MUUO, the new PC word would clear User In-out. Thus at this level all core memory references are in the executive address space and fast memory references are to an AC block in the user process table as specified by the executive stack pointer. MUUO calls by the Monitor to itself can be nested to a number of levels, but in all cases User In-out is left clear. The particular AC block used at any level is specified by the stack pointer. Hence the AC stack in the user process table is effectively a pushdown list kept by the stack pointer; at each level the program must change the pointer to specify the appropriate block. Each user process table would contain the blocks needed for carrying out MUUOs for that user.

**EXAMPLE.** Suppose that the Monitor has been called by an MUUO from the user (hence User In-out is set) and wishes to save the user's ACs in the shadow area. Assume that every user runs with AC block 1, 2 or 3, and that the Monitor always sets up executive virtual page 342 to point to the same

This makes a different set of sixteen words available at each level using the same addresses.



physical page as user page 0. Using accumulator T in block 0, the Monitor saves the user ACs by giving these two instructions,

```
MOVEI T,342000 ;Initialize pointer: from 0 to 342000
XCT 1,[BLT T,342017]
```

and restores them with these two.

```
MOVSI T,342000 ;From 342000 to 0
XCT 2,[BLT T,17]
```

## 2.16 KA10 PROGRAM AND MEMORY MANAGEMENT

The KA10 has only user and executive modes and uses protection and relocation hardware.

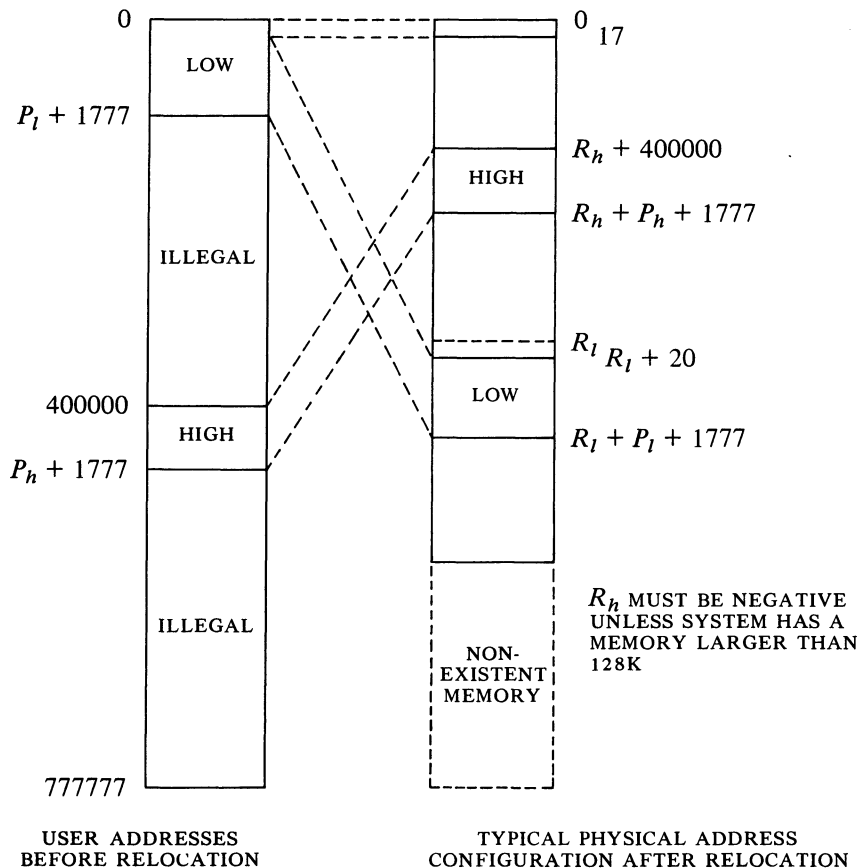
Every user is assigned a core area and the rest of core is protected from him — he cannot gain access to the protected area for either storage or retrieval of information. The assigned area is divided into two parts. The low part is unique to a given user and can be used for any purpose. The high part may be for a single user, or it may be shared by several users. The Monitor can write-protect the high part so that the user cannot alter its contents, *ie* he cannot write anything in it. The Monitor would do this when the high part is to be a pure procedure to be used reentrantly by several users. One high pure segment may be used with any number of low impure segments. The user can request that the Monitor write-protect the high part of a single program, *eg* in order to debug a reentrant program. All users write programs beginning at address 0 for the low part, and beginning usually at 400000 for the high part. The programmed addresses are retained in the object program but are relocated by the hardware to the physical area assigned to the user as each access is made while the program is running.

The size and position of the user area are defined by specifying protection and relocation addresses for the low and high blocks. The protection address determines the maximum address the user can give; any address larger than the maximum is illegal. The relocation address is the address, as seen by the Monitor and the hardware, of the first location in the block. The Monitor defines these addresses by loading four 8-bit registers, each of which corresponds to the left eight bits (18–25) of an address whose right ten bits are all 0.

To determine whether an address is legal its left eight bits are compared with the appropriate protection register, so the maximum user address consists of the register contents in its left eight bits, 1777 in its right ten bits (*ie* it is equal to the protection address plus 1777). Since the set of all addresses begins at zero, a block is always an integral multiple of  $1024_{10}$  ( $2000_8$ ) locations. Relocation is accomplished simply by adding the contents of the appropriate relocation register to the user address, so the first address in a block is a multiple of 2000. The relative user and relocated address configurations are therefore as illustrated here, where  $P_l$ ,  $R_l$ ,  $P_h$  and  $R_h$  are respectively the protection and relocation addresses for the low and high

Note that the relocated low part is actually in two sections with the larger beginning at  $R_l + 20$ . This is because addresses 0-17 are not relocated, all users having access to the accumulators. The Monitor uses the first sixteen locations in the low user block to store the user's accumulators when his program is not running.

Some systems have only the low pair of protection and relocation registers. In this case the user program is always nonreentrant and the assigned area comprises only the low part.



parts as derived from the 8-bit registers loaded by the Monitor. If the low part is larger than 128K locations, *ie* more than half the maximum memory capacity ( $P_l \geq 400000$ ), the high part starts at the first location after the low part (at location  $P_l + 2000$ ). The high part is limited to 128K. If the Monitor defines two parts but does not write-protect the high part, the user has a two-part nonreentrant program.

If the user attempts to access a location outside of his assigned area, or if the high part is write-protected and he attempts to alter its contents, the current instruction terminates immediately, the Memory Protection flag is set (status bit 22 read by CONI APR.), and an interrupt is requested on the channel assigned to the processor [§2.14].

*Addressing Summary.* Let  $A_u$  be the address supplied by the user, and let  $A_p$  be the physical core address generated from it by the relocation hardware.

If  $A_u \leq 17$ , then  $A_p = A_u$  (fast memory, no relocation).

If  $20 \leq A_u \leq P_l + 1777$ , then  $A_p = (A_u + R_l) \bmod 2^{18}$ .

If the greater of  $\left\{ \begin{matrix} 400000 \\ P_l + 2000 \end{matrix} \right\} \leq A_u \leq P_h + 1777$ ,

then  $A_p = (A_u + R_h) \bmod 2^{18}$ .

Any other value of  $A_u$  is illegal. These are  $A_u > P_l + 1777$  if either  $A_u < 400000$  or  $A_u > P_h + 1777$ .

If a relocated address is in the range 0-17, the reference is to core rather than fast memory.

**User Programming.** The user must observe the following rules when programming on a time shared basis. [*Refer to the Monitor manual for further information including use of the Monitor for input-output.*]

- ◆ Use addresses only within the assigned blocks for all purposes – retrieval of instructions, retrieval of addresses, storage or retrieval of operands. The low part contains locations with addresses from 0 to the maximum; the high part contains from the greater of 400000 or  $P_l + 2000$  to the maximum. Either part can address the other.
- ◆ If the high part is write-protected, do not attempt to store anything in it. In particular do not execute a JSR or JSA into the high part.
- ◆ Use instruction codes 000 and 040–127 only in the manner prescribed in the Monitor manual.
- ◆ Unless User In-out is set do not give any IO instruction, HALT (JRST 4,) or JEN (JRST 12, (specifically JRST 10,)). The program can determine if User In-out is set by examining bit 6 of the PC word stored by JSR, JSP or PUSHJ.

The user can give a JRSTF (JRST 2,) but a 0 in bit 5 of the PC word does not clear User (a program cannot leave user mode this way); and a 1 in bit 6 does not set User In-out, so the user cannot void any of the restrictions himself. Note that a 0 in bit 6 will clear User In-out, so a user can discard his own special privileges.

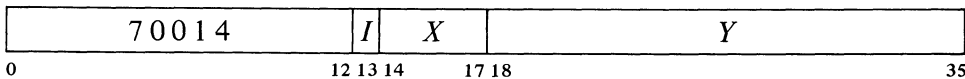
LUUOs (001–037) function normally and are relocated to addresses 40 and 41 in the low block [§2.10].

**Monitor Programming.** The Monitor must assign the core area for each user program, set up trap and interrupt locations, specify whether the user can give IO instructions, transfer control to the user program, and respond appropriately when an interrupt occurs or an instruction is executed in unrelocated 41 or 61. Core assignment is made by this instruction.

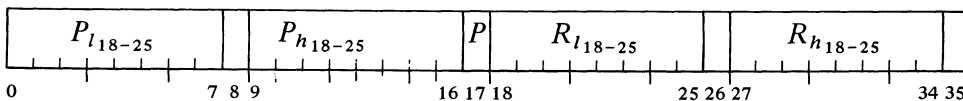
The user can actually write any size program: the Monitor will assign enough core for his needs. Basically the user must write a sensible program; if he uses absolute addresses scattered all over memory his program cannot be run on a time shared basis with others.

These instructions are illegal unless User In-out is set.

**DATA0 APR, Data Out, Arithmetic Processor**



Load the protection and relocation registers from the contents of location  $E$  as shown, where  $P_l$ ,  $P_h$ ,  $R_l$  and  $R_h$  are the protection and relocation



addresses defined above. If write-protect bit  $P$  (bit 17) is 1, do not allow the user to write in the high part of his area.

For a two part nonreentrant program, set  $P = 0$ . For a one-part nonreentrant program, make  $P_h \leq P_l$ . If the hardware has only one set of protection and relocation registers, the user area is defined by  $P_l$  and  $R_l$ , the rest of the word is ignored.

Giving a JRSTF with a 1 in bit 6 of the PC word allows the user to handle his own input-output. The Monitor can also transfer control to the user with this instruction by programming a 1 in bit 5 of the PC word, or it may jump to the user program with a JRST 1, which automatically sets User. The set state of this flag implements the user restrictions.

While User is set, certain instructions are not part of the user program and are therefore completely unrestricted, namely those executed in the interrupt locations (which are not relocated) and in unrelocated trap locations 41 and 61. Illegal instructions and UWO codes 000 and 040-077 are trapped in unrelocated 40; codes 100-127 are trapped in unrelocated 60. BLKI and BLKO can be used in the even interrupt locations, and if there is no overflow, the processor returns to the interrupted user program. JSR should ordinarily be used in the remaining even interrupt locations, in odd interrupt locations following block IO instructions, and in 41 and 61. The JSR clears User and should jump to the Monitor. JSP, PUSHJ, JSA and JRST are acceptable in that they clear User, but the first two require an accumulator (all accumulators should be available to the user) and the latter two do not save the flags.

After taking appropriate action, the Monitor can return to the user program with a JRSTF or JEN that restores the flags including User and User In-out.

The trap locations are 140-141 and 160-161 in a second KA10 processor.

## 2.17 REAL TIME CLOCK DK10

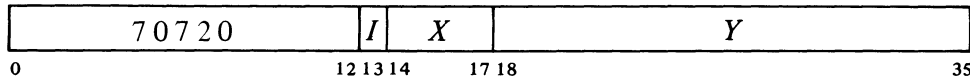
The clock referred to throughout this section is the DK10 real time clock and should not be confused with the line frequency clock whose flag is one of the processor conditions [§2.14].

This processor option can be used to signal the end of a specified real time interval or to measure the real time taken by an event. With appropriate software the DK10 can easily be used to keep the time of day. The basic element in the clock is an 18-bit binary counter that is incremented repeatedly by a clock source; a 100 kHz  $\pm$  .01% crystal-controlled source is available internally, or a source of any frequency up to 400 kHz can be provided externally. Operation is synchronized so that the program can read the counter at any time without missing a count. Associated with the counter is an 18-bit interval register, which can be loaded by the program. Each time the count reaches the number held in the register, the clock requests an interrupt while the counter clears and begins a new count. With the internal clock source, whose period is 10  $\mu$ s, the total count is about 2.6 seconds.

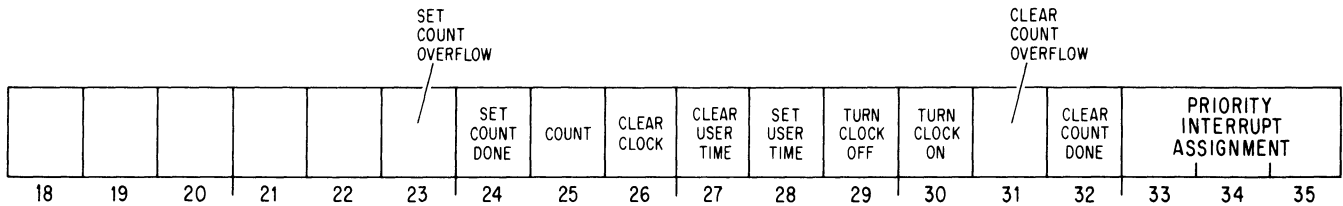
The program turns the clock on and off by enabling and disabling the counter. The clock has two modes of operation: with the User Time flag clear, the counter operates continuously; with User Time set, the counter stops while the processor is handling interrupts. Hence in the latter mode the clock discounts interrupt time and can be used to time user programs. In a system that contains two clocks, one can be used by the Monitor to time user programs while the other is used to keep the time of day.

**Instructions.** The clock device code is 070, mnemonic CLK. A second clock would have device code 074.

**CONO CLK, Conditions Out, Clock**



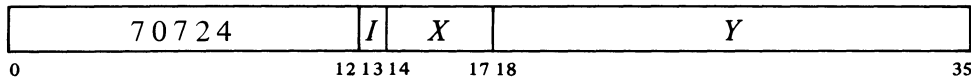
Assign the interrupt channel specified by bits 33–35 of the effective conditions *E* and perform the functions specified by bits 23–32 as shown (a 1 in a bit produces the indicated function, a 0 has no effect).



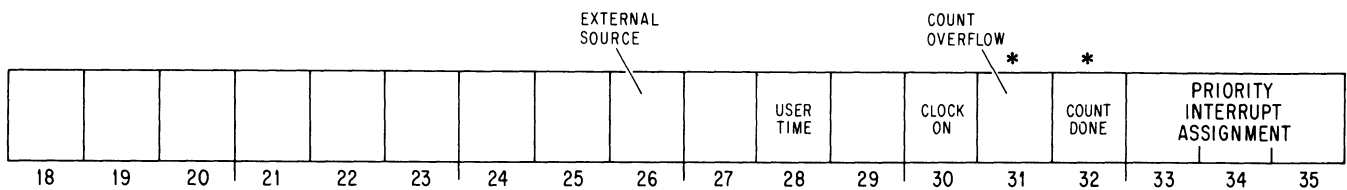
A 1 in bit 26 clears the clock counter and the Count Done, Count Overflow and User Time flags, turns off the clock, and dismisses the PI assignment (assigns zero). The effect of giving conflicting conditions is indeterminate.

A 1 in bit 25 increments the counter provided the clock is off (this is for maintenance only).

**CONI CLK, Conditions In, Clock**



Read the contents of the interval register into the left half of location *E* and read the status of the clock into bits 26–35 as shown.

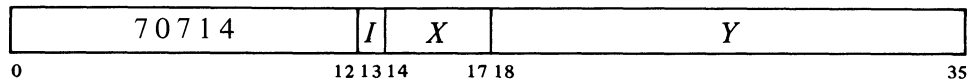


Interrupts are requested on the assigned channel by the setting of Count Overflow and Count Done.

\*These bits cause interrupts.

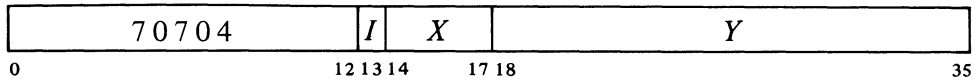
- 26 The counter is connected to an external source (0 indicates the internal source is connected).
- 28 The counter cannot be incremented while an interrupt is being held or a request has been accepted and the channel is waiting for an interrupt to start.

Note that to time a user properly, the Monitor must also compensate for any noninterrupt time taken from the user.

**DATAO CLK, Data Out, Clock**

The comparison of the counter against the interval register that follows every count is inhibited while this instruction is loading the register.

Load the contents of the right half of location *E* into the interval register.

**DATAI CLK, Data In, Clock**

The counter is always stable while being read, and any count held back is picked up immediately afterward.

Read the current contents of the clock counter into the right half of location *E*.

Following turnon the first count may occur at any time up to the full period of the source.

Remember that although a CONO need not affect the mode or the clock state, every CONO must renew the PI assignment.

Initially the program should give a CONO CLK,1000 to clear the clock, and then give a DATAO to select the interval and a CONO to turn on the clock, select the mode, and assign the interrupt channel. When the count reaches the specified interval, Count Done sets, requesting an interrupt on the assigned channel. At the same time, the counter clears and a new count begins with the next pulse. The program should respond with a CONO to clear Count Done.

The interval can be changed at any time simply by giving a DATAO. However, if the program does not clear the counter at the same time, then it should make sure that the count has not yet reached the value of the new interval. If the count is already beyond that point, the counter will continue until it overflows. When the counter overflows, either because the count started too high, the program specified the maximum count ( $2^{18}$  is selected by loading zero), or there is a malfunction of some sort, Count Overflow sets, requesting an interrupt, and a new count begins.

To use the clock to time some operation, turn it on with the counter at zero. For a counter reading of *C*, the elapsed time is

$$T(C + nI)$$

where *T* is the period of the source, *n* is the number of clock interrupts since the clock was started, and *I* is the interval selected by the program. To cause the clock to request an interrupt after  $T \times n \mu\text{s}$ , where  $n \leq 2^{18}$  and *T* is the period of the source in microseconds, load the interval register with *n* expressed in binary. There is an average indeterminacy of half a count every time the counter starts and stops. Therefore, when the clock is keeping user time, there is an average indeterminacy of one count for every *group* of overlapping interrupts and requests (not for every interrupt, as the counter is inhibited while there is any request or interrupt being held).

For keeping the time of day, the program can use a memory location to maintain a count of the clock interrupts. The location should be cleared

at midnight, and the time can be determined by combining its contents with the current contents of the clock counter. If the location itself is to be used as a low resolution clock kept in hours, minutes and seconds, it is better to use a more convenient interval than the full count. Using the internal source, an interval of  $2\frac{1}{2}$  seconds, which is octal 750220, is the most straightforward interval with the fewest interrupts. To interrupt every second the interval would be 303240.

Note that an error of .01% amounts to 8.64 seconds in 24 hours.





# 3

## Console In-out Equipment

The PDP-10 contains three in-out devices as standard equipment: tape reader, tape punch, and a console terminal. The punch supplies output in the form of 8-channel perforated paper tape in either of two modes. In alphanumeric mode, 8-bit characters are processed; in binary mode, 6-bit characters. Information punched in the tape can be brought into memory by the tape reader, which handles characters in the same two modes. Reader and punch are generally used at the most basic program level, such as for program loading and assembler output.

The console invariably has a full duplex terminal, combining a keyboard with a printer or a display screen. The program can type out characters on the printer or screen and can read characters that have been typed in at the keyboard. The terminal is a slow device, but it provides the most convenient means of communication between man and machine.

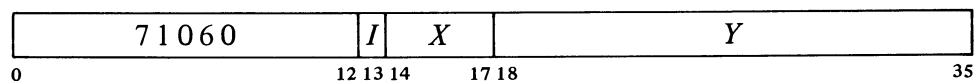
The choice of alphanumeric code used with paper tape is up to the programmer, but it is generally best to use the ASCII code given in Appendix B.

Maintenance procedures require that the console have a hardcopy terminal when certain long diagnostic and reliability tests are run. However a terminal with a display screen can be used for normal operations, and a hardcopy terminal substituted only for the tests that require it.

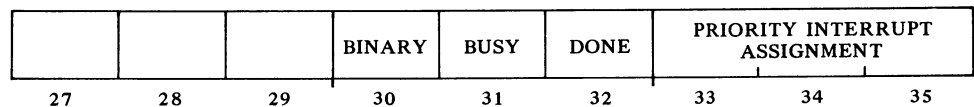
### 3.1 PAPER TAPE READER

The reader processes 8-channel perforated paper tape photoelectrically at a speed of 300 lines per second. The device can operate in alphanumeric or binary mode, as specified by the 0 or 1 state respectively of the Binary flag. In alphanumeric a single tape-moving command reads all eight channels from the first line encountered. In binary the device reads six channels from the first six lines in which hole 8 is punched and assembles the information into a 36-bit word. The interface contains a 36-bit buffer from which all data is retrieved by the processor. The reader device code is 104, mnemonic PTR.

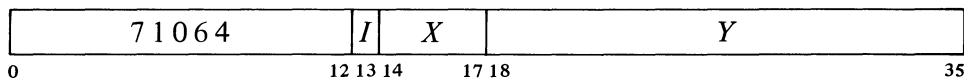
#### CONO PTR, Conditions Out, Paper Tape Reader



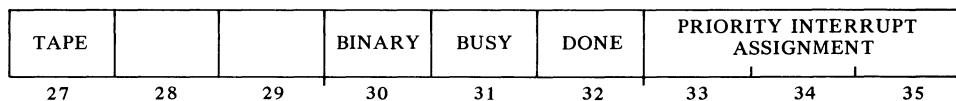
Set up the reader control register according to bits 30-35 of the effective conditions *E* as shown (a 1 in a flag bit sets the flag, a 0 clears it).



**CONI PTR, Conditions In, Paper Tape Reader**

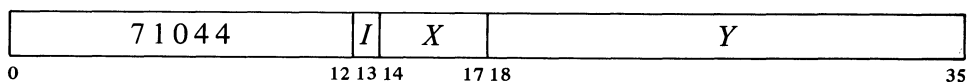


Read the status of the reader into bits 27 and 30-35 of location *E* as shown.



Placing the tape in motion sets the Tape flag and it remains set so long as the tape is in the reader. If tape motion is caused by the operator pressing the tape feed switch, the on transition of the flag sets Done, requesting an interrupt on the assigned channel. A 0 in bit 27 indicates that the last time an attempt was made to read, the reader was out of tape.

**DATAI PTR, Data In, Paper Tape Reader**



Transfer the contents of the reader buffer into location *E*. Clear Done and set Busy.

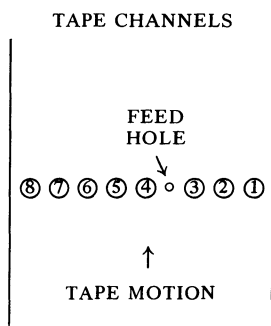
The absence of tape is detected by the absence of tape material between feed holes. Hence damage that results in the joining of two feed holes will most likely be mistaken for the end of tape. Conversely, if the tape is torn diagonally in an area where the program expects to retrieve data, the end may well not be detected until after the program has read a few garbage characters.

Setting Busy clears the reader buffer, sets the Tape flag (if it is not already set) and places the reader in operation. If Binary is clear, all eight channels from the first line on tape are read into bits 28-35 of the buffer with channel 1 corresponding to bit 35 (the presence of a hole produces a 1 in the buffer). If Binary is set, the device reads only channels 1-6, but it reads the first six lines encountered in which channel 8 is punched (lines without a hole in channel 8 are skipped) and assembles them into a full word in the buffer. The first line is at the left in the word and channel 1 corresponds to the rightmost bit in each 6-bit byte.

After the specified number of lines has been read, the reader clears Busy and sets Done, requesting an interrupt on the assigned channel. A DATAI brings the data into memory and also causes the reader to continue in operation. The programmer must give a CONO to clear Busy if he does not want the reader to move the tape after the final DATAI is given.

The operator can signal the program that a tape has been loaded by feeding a few frames and thus setting the Tape flag; this in turn sets Done to request an interrupt. The Tape flag is cleared if the tape runs out or malfunctions while a read operation is in progress, or is deliberately run out by the operator.

**Timing.** At 300 lines per second the reader takes 3.33 ms per alphanumeric character, 20 ms per binary word if the binary characters are contiguous. After Done is set, the program has 1.6 ms to give a DATAI and



keep the tape in continuous motion. Waiting longer causes the reader to shut down for 40 ms. Thus start-stop operation is limited to 21 lines per second in alphanumeric, 15½ reads in binary.

EXAMPLES. This program reads ten binary words (60 lines) from paper tape and stores them in memory beginning at location 4000. The block pointer is kept in accumulator PNT.

```

                MOVE   PNT,[IOWD 12,4000]    ;Put pointer in PNT
                CONO   PTR,60                ;Set up reader
NEXT:           CONSO  PTR,10                ;Watch Done
                JRST   .-1
                BLKI   PTR,PNT               ;Word ready, get it
                JRST   .+2                   ;Got all data
                JRST   NEXT                  ;Go back for next word
                :
                :

```

If instead of just waiting we wish to continue our program while the data is coming in, we can use the priority interrupt. The following uses channel 4 and signals the main program that the data is ready by setting bit 35 of accumulator F.

```

                MOVE   17,[BLKI PTR,[IOWD 12,4000]]
                MOVEM  17,50                 ;Set up 50 and 51 for channel 4
                MOVE   17,[JSR  DONE]
                MOVEM  17,51
                CONO   PTR,64                ;Set up reader on channel 4
                CONO   PI,12210              ;Clear PI, then activate it and turn on
                                                ;channel 4
                :                             ;Continue program
                :
                TRZN   F,1                   ;Check if data ready when needed
                JRST   .-1                   ;Wait if necessary
                :
                :
DONE:           0                             ;Interrupt routine, block done
                CONO   PTR,0                 ;Stop tape
                TRO    F,1                   ;Set F bit 35
                JEN    @DONE                  ;Dismiss and restore flags

```

### Readin Mode

The only requirement (beyond those given in §2.12) for readin mode with paper tape is that the data must be in binary (hole 8 punched). To select the reader in the readin device switches, turn on the third from the left and the last on the right (104).

The program below is the RIM10B Loader, which is brought into the accumulators in readin mode, and then continues to read any number of blocks of binary data from the same tape. The tape is formatted as a series

of blocks separated by a half-dozen lines of blank tape (tape with only feed holes punched). The first block is the loader in readin format. The rest of the tape contains any number of data blocks and ends with a transfer block. Each data block contains any number of words of program data, preceded by a standard IO block pointer for the data only, and followed by a checksum, which is the sum of all the data words and the pointer. It is recommended that the number of data words per block be limited to twenty for ease in repositioning the tape in case of error. The transfer block is a JRST to the starting location of the program, followed by a throw-away word to stop the reader.

This loader is written for minimum size and is quite complex. Do not approach it as a simple programming example.

```

XWD      -16,0          ;1410 words starting at location 1
ST:      CONO  PTR,60   ;Set up reader binary
ST1:     HRRI  A,RD+1   ;Put RD+1 in Y part of A
RD:      CONSO PTR,10   ;Watch Done
JRST     .-1
DATAI   PTR,@TBL1-RD+1(A) ;First and last words in
                        ;ADR, data in block
XCT     TBL1-RD+1(A)   ;TBL1+2 first word, +1 data,
                        ;+0 checksum
XCT     TBL2-RD+1(A)   ;TBL2+2 JRST, +1 data, +0
                        ;bad checksum
A:      SOJA  A,        ;RD+1 first word, RD data, RD-1
                        ;last word
TBL1:   CAME  CKSM,ADR  ;Compare computed checksum with
                        ;one read
        ADD  CKSM,1(ADR) ;Add word read to checksum
        SKIPL CKSM,ADR  ;Put first word in CKSM, skip if
                        ;pointer
TBL2:   JRST  4,ST      ;Halt if checksum bad
        AOBJN ADR,RD    ;If data done, go to A; otherwise wait
                        ;for next word
ADR:    JRST  ST1      ;Read in executes this. First and last
CKSM=ADR+1 ;word of each block also put here

```

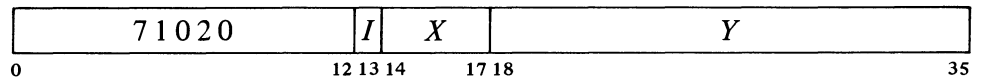
The processor halts if a computed checksum does not agree with the tape. To reread a block, move the tape back to the preceding blank area and press the continue key. A halt following the transfer block is not an error — many programs begin by halting.

### 3.2 PAPER TAPE PUNCH

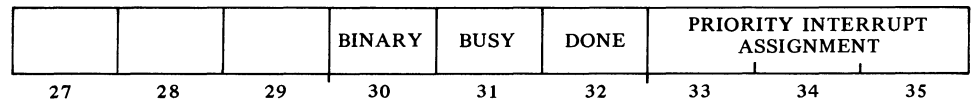
The punch perforates 8-channel tape at speeds up to 50 lines per second. It can operate in alphanumeric or binary mode, as specified by the 0 or 1 state respectively of the Binary flag; but in either mode a single tape-moving command punches only one line. Alphanumeric mode punches an 8-bit character supplied by the program; binary mode always punches channel 8,

never punches channel 7, and punches a 6-bit character in the remaining channels. The interface contains an 8-bit buffer that receives data from the processor. The punch device code is 100, mnemonic PTP.

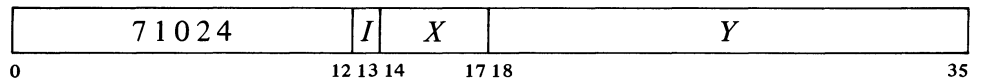
**CONO PTP, Conditions Out, Paper Tape Punch**



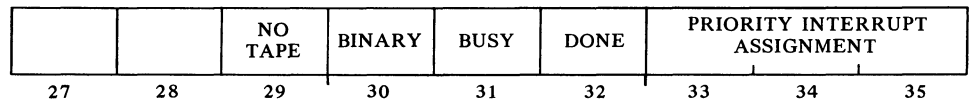
Set up the punch control register according to bits 30–35 of the effective conditions *E* as shown (a 1 in a flag bit sets the flag, a 0 clears it).



**CONI PTP, Conditions In, Paper Tape Punch**

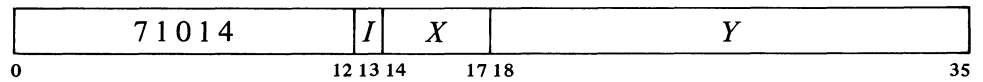


Read the status of the punch into bits 29–35 of location *E* as shown.



A 1 in bit 29 indicates that the punch is out of tape.

**DATAO PTP, Data Out, Paper Tape Punch**



Load the contents of bits 28–35 of location *E* into the punch buffer. Clear Done and set Busy.

A CONO need be given only to change Binary or the PI assignment; DATAO sets Busy while loading the buffer. Setting Busy places the punch in operation. If Binary is clear, one line is punched in tape from bits 28–35 of the buffer with bit 35 corresponding to channel 1 (a 1 in the buffer produces a hole in the tape). If Binary is set, channel 8 is punched, channel 7 is not

punched, and the remaining channels are punched from bits 30–35 of the buffer with bit 35 corresponding to channel 1. After punching is complete, the device clears Busy and sets Done, requesting an interrupt on the assigned channel.

**Timing.** If Busy is set when the punch motor is off, punching is automatically delayed 1 second while the motor gets up to speed. While the motor is on, punching is synchronized to a punch cycle of 20 ms. After Done sets, the program has 10 ms within which to give a new DATAO to keep punching at the maximum rate; after 10 ms punching is delayed until the next cycle. If Busy remains clear for 5 seconds the motor turns off.

**EXAMPLE.** Suppose we wish to punch out the same information we read from tape in the examples of the previous section. We cannot use a BLKO as an interrupt instruction unless we first spread the 6-bit characters over sixty memory locations. The example uses channel 5 and assumes that other channels are already in use.

```

                MOVE  A,[JSR PUNCH]
                MOVEM A,52          ;Set up channel 5
                CONO  PTP,55        ;Request interrupt for first word
                CONO  PI,2004       ;Turn on channel 5
                .                   ;Continue program
                .
                .
PUNCH:         0                   ;Interrupt routine
                ILDB  A,BYPPNT      ;Put byte in A
                AOSL  CNT           ;Got all bytes?
                CONO  PTP,40        ;Yes, prevent interrupt after last word
                DATAO PTP,A        ;Punch byte
                JEN   @PUNCH
BYPPNT:       XWD   440600,4000    ;Generate pointer here
CNT:          ↑D-60                ;Initialize count

```

### 3.3 CONSOLE TERMINAL

Mounted inside the console bay of every KI10 and KA10 is an asynchronous serial interface for connection to a terminal to be used by the operator for direct communication with the computer software. Through this interface the program can type out characters and can read in the characters produced when keys are struck at the keyboard. The interface is designed for full duplex operation, *ie* it separates its input and output functions so the terminal in effect acts as though it were two devices with a single device code. Each device has its own Busy and Done flags, but the two share a common interrupt channel. Placing the code for a character in the output buffer causes the terminal to print or display the character or perform the designated control function. Striking a key places the code for the associated character in the input buffer where it can be retrieved by the program, but it does nothing at the terminal unless the program sends the code back as

output (*ie* echoes the input).

The interface handles terminals that operate either in current mode or with EIA standard levels. Terminals differ one from another in various programming characteristics, particularly the character set, timing, and fill requirements between certain characters. Transmission rates available for the KI10 terminal are 110, 150, 300, 600, 1200 and 2400 baud; these rates are switch selectable for input and output separately. The KA10 interface is set up for a particular baud rate (regularly 110 or 150), which can be changed only through modification by Field Service. Characters always contain eight data bits. At 110 baud the interface handles 11-unit characters with two stop bits; at all other rates characters are ten units with a single stop bit. Most terminals use the ASCII character set given in Appendix B, or some subset of it, and the control characters that affect a given terminal are generally used in a manner that is relatively consistent with their ASCII definitions. For information on terminal-dependent characteristics, refer to the description of the appropriate terminal in Chapter 9.

Any terminal is, however, connected to the same interface, is controlled by the same instructions, and is in general programmed in the manner described here. The terminal device code is 120, mnemonic TTY, which stands for telétypewriter.

**CONO TTY, Conditions Out, Console Terminal**

7 1 2 2 0	I	X	Y
0	12 13 14	17 18	35

Set up the terminal control register according to bits 24–35 of the effective conditions *E* as shown (a 1 in bit 24 sets Test, a 0 clears it; all other flag functions are produced by 1s, 0s have no effect).

TEST	CLEAR INPUT BUSY	CLEAR INPUT DONE	CLEAR OUTPUT BUSY	CLEAR OUTPUT DONE	SET INPUT BUSY	SET INPUT DONE	SET OUTPUT BUSY	SET OUTPUT DONE	PRIORITY INTERRUPT ASSIGNMENT
24	25	26	27	28	29	30	31	32	33 34 35

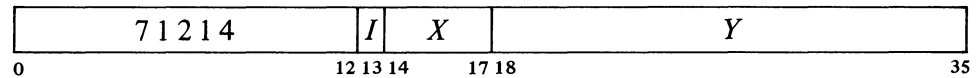
Setting Test connects the output buffer directly to the input buffer, allowing the program to check out the interface logic without the line and the device.

**CONI TTY, Conditions In, Console Terminal**

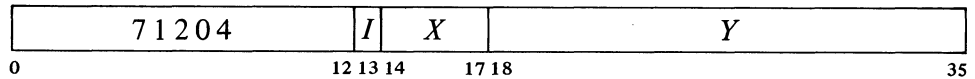
7 1 2 2 4	I	X	Y
0	12 13 14	17 18	35

Read the status of the terminal into bits 24 and 29–35 of location *E* as shown.

TEST					INPUT BUSY	INPUT DONE	OUTPUT BUSY	OUTPUT DONE	PRIORITY INTERRUPT ASSIGNMENT
24	25	26	27	28	29	30	31	32	33 34 35

**DATAO TTY, Data Out, Console Terminal**

Load the contents of bits 28–35 of location *E* into the output buffer. Clear Output Done, set Output Busy, and enable the transmitter.

**DATAI TTY, Data In, Console Terminal**

Transfer the contents of the input buffer into bits 28–35 of location *E*. Clear Input Done.

**Output.** A CONO need be given only to change the PI assignment; DATAO sets Output Busy and enables the transmitter while loading the buffer. Enabling the transmitter causes it to send the contents of the output buffer serially to the terminal. Completion of transmission clears Output Busy and sets Output Done, requesting an interrupt on the assigned channel.

**Input.** Reception requires no initiating action by the program except to supply a PI assignment. Striking a key transmits the code for the character serially to the input buffer. The beginning of reception sets Input Busy; completion clears Input Busy and sets Input Done, requesting an interrupt on the assigned channel. A DATAI brings the character into memory and clears Input Done.

**Timing.** After Output Done is set, the time the program has to give a DATAO to keep typing at the maximum rate is a half bit time in the KI10, one bit time in the KA10. After Input Done is set, the character is available for retrieval by a DATAI for a half bit time plus the stop bit times before another key strike can destroy it. Generally speaking, keyboard transfer rates that are suitable for a console terminal are 110, 150 and 300 baud (ten, fifteen and thirty characters per second). The available times for the various rates are as follows.

	<i>Baud rate</i>	<i>Bit time in ms</i>	<i>KI10 DATAO time in ms</i>	<i>KA10 DATAO time in ms</i>	<i>DATAI time in ms</i>
*Two stop bits.	110	9.09	4.54	9.09	22.7*
	150	6.67	3.33	6.67	10
	300	3.33	1.67		5
†Not generally recommended.	600	1.67	.833		2.5†
	1200	.833	.417		1.25†
	2400	.417	.208		.625†



The only other timing considerations are the fill character requirements. These vary greatly from one terminal to another and are treated with the terminals in Chapter 9.

2

3

4

5

# 4

## Hardcopy Equipment

This chapter discusses the line printer, *XY* plotter, card reader, and card punch. Like the basic in-out equipment, these devices are primarily for communication between computer and operator using a paper medium: form paper, graph paper or cards.

The line printer provides text output at a relatively high rate. The program must effectively typeset each line; upon command the printer then prints the entire line. With the plotter, the program can produce ink drawings by controlling the incremental motion of pen on paper in a cartesian coordinate system. Curves and figures of any shape can be generated by proper combinations of motion in *x* and *y*.

The card equipment processes standard 12-row 80-column cards. Many programmers find cards a convenient medium for source program input and for supplying data that varies from one program run to another. Cards are convenient to prepare manually, input is much faster than paper tape, and simple changes are easy to make: individual cards can be repunched, and cards can be added or removed from the deck. The card reader cannot be used in readin mode, but a standard card-reading program in readin format can be kept on paper tape or DECTape. A possible consideration in using cards is that many installations do not include an online card punch.

These four devices are all run by the BA10 Hardcopy Control. Interface logic for a plotter can also be mounted in the TD10A DECTape Control.

### 4.1 LINE PRINTER LP10

The line printer outputs hardcopy composed of lines 132 characters long at rates of 300 to 1250 lines per minute. Character sets available on the various models are listed in tables of the line printer code in Appendix B. Besides accepting printing characters, the printer responds to ten control characters, HT, CR, LF, VT, FF, DLE and DC1-4. Only these control codes affect the printer, but the interface recognizes two others: NUL, which is ignored, and delete, which allows expansion of the character set by providing a means for distinguishing between control characters and printing characters with the same codes. All other codes are ignored.

Printers LP10A, B, C and F have a 64-character print drum, whose print positions are selected by the figure and upper case codes, 040-137. But lower case codes (140-176) are also valid for these printers: when a lower

Virtually any character set can be had on any printer by special order.

Although most control codes are ignored, it is recommended that NUL be used for fill, as DEC guarantees it to be suitable for that purpose.

The feature for selecting the hidden character is optional. The standard version therefore has a 95-character set even though it has a 96-character drum.

case code is given, the corresponding upper case code is loaded into the buffer. Models D and H have a 96-character drum whose print positions are selected by the figure, upper case and lower case codes, 040–176, and the delete code. A single delete code is ignored, but two consecutive 177s cause the code 177 to be loaded into the buffer. When a code for a printing character is the same as one for a nonprinting character and is loaded by giving it immediately after a delete, the printing character is said to be “hidden” under the nonprinting one. Model E has a 128-character drum and uses the entire set of 7-bit codes for printing characters, with characters hidden under the ten control characters and also under null and delete.

The character sets in Models A to E are fixed, but Models F and H have removable drums. Two standard versions (designated respectively by E and F appended to the model number) of these drums are available, with EDP and scientific character sets; in the latter, zero and Z are crossed.

The printer has a 132-character buffer that holds the image of a single line; the program must first load the buffer up to five characters at a time, and then give a control character to print the entire line. The buffer is loaded from left to right, and only the portion filled produces a printout. Hence for each line the program need send out characters (including spaces) only as far as the rightmost nonspace character. The characters are printed in the order that they pass the print hammers, and a given character is printed simultaneously in all positions that require it. In other words the drum has a row of 132 *Ms*, a row of *Ns*, etc; all *Ms* are printed together, all *Ns* together, and so forth. The first character printed depends only upon the position of the drum when the print command is given.

**Output Format.** Paper motion is controlled by a format tape loop in the printer. The tape has eight columns and the loop corresponds to an integral number of pages of the fanfold form paper. With the exception of CR, every control character that prints a line from the contents of the buffer produces a different spacing by selecting a particular tape column. The paper then advances until a hole is encountered in the selected column.

The standard paper has 11-inch pages of sixty-six lines, and the standard tape for these generates the formats listed below. The fourth column gives the hole positions in terms of the numbered lines *on the tape*. The tape is usually installed at random and then positioned by pressing the top-of-form button on the printer. Then the paper is adjusted so that the desired line on the paper corresponds to line 0 on the tape. Ordinarily the paper is set with the print hammers at the fourth line, so all but one of these formats leaves a three-line margin at the top and a margin of at least three lines at the bottom of each page.

<i>Character</i>	<i>Column</i>	<i>Normal meaning</i>	<i>Hole positions</i>
FF (014)	1	Top of form	Line 0
CR (015)	<i>None</i>	No spacing (paper motion inhibited)	
LF (012)	8	Single space with automatic top of form after every 60 impressions	Every line from 0 to 59

Spacing other than the standard can be produced by using a different format tape. The length of the loop should correspond to one or more pages of the printer form used, with holes punched at the lines where paper spacing is to stop. Models F and H can be set for eight lines per inch.

Programmers generally treat the data for the line printer and teletype identically, using the combination CR plus LF for printing and spacing. This way a given character string can be outputted on either device. CR is used alone only when the next print command

DC1 (021)	3	Double space with automatic top of form after every 30 impressions	Every even numbered line from 0 to 58
DC2 (022)	4	Triple space with automatic top of form after every 20 impressions	Every third line from 0 to 57
DC3 (023)	5	Single space	Every line
DC4 (024)	6	Space one sixth of a page	Lines 0, 10, 20, 30, 40, 50
VT (013)	7	Space one third of a page	Lines 0, 20, 40
DLE (020)	2	Space half a page	Lines 0, 30

will overprint, *ie* will print another character in a column position already printed. With this technique the program can produce a character such as “≠” by overprinting a slash on an equal sign (or vice versa).

The actual printer action of advancing the paper to the next hole in the tape produces the “normal” format only if the program consistently selects the same tape column. Always using DC1 to print produces double spaced text from line 4 to line 62 on every page. But if the last print command spaced to an odd numbered line, DC1 moves the paper only one line.

**Printing Speed.** The printer is available in seven models with differing printing speeds. On models with two speeds, the slower one gives higher printing quality.

<i>Printer</i>	<i>Number of Characters</i>	<i>Nominal printing speed in lines per minute</i>	<i>Drum rotation in rpm</i>	<i>Time per revolution in ms</i>
LP10A	64	300	333	180
LP10B	64	600	750	80
LP10C	64	1000	1250	48
LP10D	95/96	600	750	80
LP10E	128	500	550	109
LP10F	64	925/1250	1200/1800	50/33
LP10H	95/96	675/925	800/1200	75/50

Printing begins as soon as a print command is given and terminates when the last required character is printed, *ie* without necessarily waiting for a complete drum revolution. Therefore print time depends on the initial drum position and the number of characters that must pass the print head before the last is printed. No time is required for spaces: the printer produces spaces in a line by not printing anything in the columns corresponding to the buffer positions that hold space characters. As a given character is printed, space codes replace the codes for the character in all buffer positions that hold it, and printing ceases when the buffer is filled with spaces.

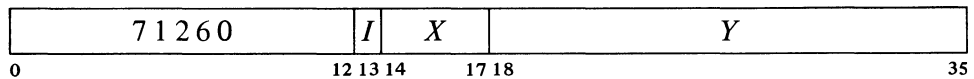
A complete print cycle consists of the print time plus the time required for advancing the paper; paper spacing begins immediately after printing terminates, and further printing is inhibited while the paper is moving. It takes about 12 ms to advance the paper one line, about 6–8 ms for each additional

line. If the buffer is loaded only with spaces, the print cycle consists entirely of paper spacing.

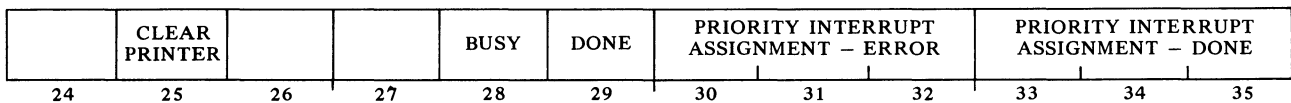
Using an ordinary distribution of characters results in printing at or slightly above the nominal speed. Printing is faster however if paper spacing occurs while unused characters are passing the print head. *Eg* text that uses only the alphabet can be printed at the full drum rotation speed.

**Instructions.** The printer has the usual instructions for sending and reading conditions, but after initial setup it can be controlled entirely by the characters sent by a string of DATAOs. The program supplies five characters at a time to a 35-bit character buffer in the printer interface. The interface processes the characters from left to right loading valid data characters into the line buffer, ignoring invalid characters, and sending control signals to the printer when a control character is encountered. The printer device code is 124, mnemonic LPT. A second printer would have device code 234, and a third would have code 230.

#### CONO LPT, Conditions Out, Line Printer



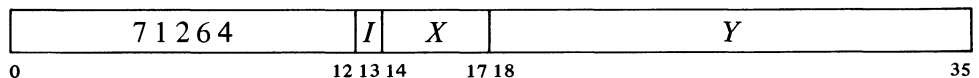
Perform the function given below if specified by a 1 in bit 25 and set up the printer control register according to bits 28–35 of the effective conditions *E* as shown (a 1 in a flag bit sets the flag, a 0 clears it).



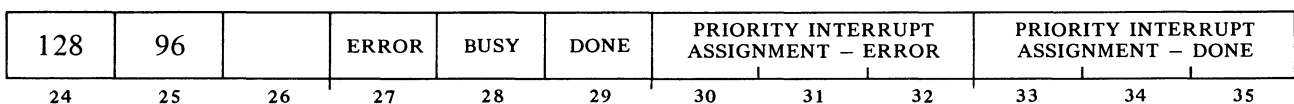
Power turnon and the IO reset signal generated by CONO APR,200000 duplicate this clear function.

If bit 25 is 1, clear Done, set Busy, clear the interface logic, and trigger a print cycle to clear the line buffer. The cycle clears the buffer by replacing the characters in it with spaces, and the time required is the same as would be required to print whatever is in it. Completion of the cycle clears Busy and sets Done, requesting an interrupt on the channel assigned by bits 33–35.

#### CONI LPT, Conditions In, Line Printer

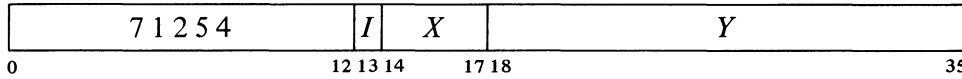


Read the status of the printer into bits 24–35 of location *E* as shown.

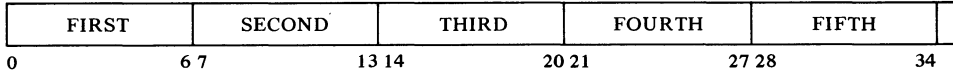


A 1 in bit 24 indicates that the printer has a 128-character drum; a 1 in bit 25 indicates that at least 95 characters are available to the program.

**DATAO LPT, Data Out, Line Printer**



Load the contents of bits 0–34 of location *E* into the character buffer, clear Done, set Busy, and trigger the interface processing cycle. The format of the data word and the order in which the characters are processed is as shown.



Characters are assembled into words in this manner by an IDPB loop or an ASCII or ASCIZ pseudoinstruction.

Following power turnon, the Error flag (CONI bit 27) is set if the printer cable is not connected or any other condition exists that makes the printer unavailable to the program [*these other conditions are given in the discussion of printer operation in Appendix H2*]. If Error is set when a CONO gives an error PI assignment (with bits 30–32 of *E*), there is an immediate interrupt request on the error channel. Barring accident or hardware malfunction, an error interrupt is likely to occur during a printout run only when the printer is about to run out of paper or the operator stops it (in either case Error sets and the printer stops when the buffer is empty following the printing of a line).

Models F and H always complete the final form before shutting down when the paper runs out. Models A to E shut down after completing the current line when a low paper alert occurs. A program intended to print to the end of form should, upon operator direction, provide data only to the end of the present form. Then the operator can make the printer print one line at a time but only until the data is exhausted [*refer to §H2.1*]. Upon completing the form, the program should refuse to supply anymore data – even should the operator enable printing – until the operator has given some indication (as by a message via the console terminal) that new paper has been loaded.

At the beginning of a print run the program should give a CONO to clear the line buffer and assign the PI channels. After that a CONO need be given only to change the PI assignments; each DATAO starts the character-processing operations of the interface while loading the character buffer. The interface processes the characters from left to right, starting each character cycle when the line buffer is ready. Printing characters are simply sent to the buffer, with lower case codes translated to upper case for a 64-character printer. Unused codes are ignored. The interface responds as follows when a control character is encountered.

◆ A horizontal tab (HT) is simulated by sending a string of spaces to the line buffer. Tab stops are every eight columns (9, 17, . . .). The interface always sends at least one space, and then sends as many more as are necessary for the next character to be at a tab stop. Thus if a DATAO gives the sequence

A HT B

These tabs are the same as the ones ordinarily used on a terminal.

where *A* is placed in column 7, *B* will go into column 9. But if *A* goes into column 8, *B* will go into column 17.

◆ Upon encountering any other printer control character, the interface signals the printer to print the contents of the line buffer, and unless the character is CR, it also selects a format tape column to space the paper as listed in the format discussion at the beginning of this section. When the buffer again becomes available, subsequent characters will be loaded starting in column 1. If printing is caused by a CR, the next line will overprint unless the paper is advanced before any nonspace characters are loaded into the buffer.

If the buffer is filled with 132 characters and the next character does not cause printing, the interface simulates a line feed to print and advance the paper, and then loads the next character at column 1 for the new line. If the program tabs to the end of a line, the interface simulates a line feed and also tabs at the beginning of the next line. In other words a printing character following the tab will be loaded at column 9 for the new line.

When the interface finishes processing the five characters supplied by a DATAO, it clears Busy and sets Done, requesting an interrupt on the channel assigned by bits 33–35 of the conditions out.

**Timing.** The time from one DATAO to the next while the program is loading the buffer is simply the time required by the interface to process five characters. Loading each printing character, including each space in a horizontal tab, takes 10  $\mu$ s. Skipping an illegal character takes 8  $\mu$ s.

If the fifth character causes printing, Done is set immediately and the program can give a DATAO to send the first set of characters for the next line. However, the interface does not begin processing the new characters until the buffer becomes available after the printer finishes printing the previous line. If printing is produced by any character before the last, the print time elapses before the interface processes the next character in the current set.

The overall time required for a print run is the total printing and spacing time for all lines as given above in the discussion of the printing speed. The time required to process individual characters is a consideration in programming the DATAOs that load the buffer, but buffer loading time is not a factor in total printer operating time except when loading characters for overprinting (following a CR). This is because the buffer becomes available while the paper is moving, in plenty of time for the program to load it before the paper stops.

**EXAMPLES.** In the first example, which uses the line printer without the interrupt, we have simply filled in the missing part of the print subroutine given in §2.9 (it prints the characters that accompany the calling sequence given just before the subroutine).

```
PRINT:  HRLI    T,440700
        ILDB   CH,T
        JUMPE  CH,1(T)
        CONSZ  LPT,200      ;Skip when printer not busy
        JRST   .-1         ;Wait for Busy to clear
        LSH    CH,1        ;Shift character to bits 28-34
```



```

DATAO LPT,CH      ;Send character to printer
JRST  PRINT+1

```

The same program could be used for output on the console terminal by changing

```

CONSZ LPT,200 to CONSZ TTY,20
DATAO LPT,CH to DATAO TTY,CH

```

and deleting the LSH CH, 1.

The above is perhaps an overly simple example. It assumes the line buffer is clear initially and the printer is available. Moreover the processor spends most of its time waiting. Characters are processed individually in order to detect the null, but if the processor has anything else to do, it would be much more efficient to use the interrupt and send five characters at a time.

In the following example the main program sets up each print run by giving a JSR SETUP. The number of words printed and the starting location of the block containing them are determined by the contents of PNTR1. Once a run is set up, the program can change the contents of PNTR1 for the next one.

```

SETUP:  0
        SKIPGE PNTR
        JRST  .-1          ;Wait for current IO to finish
        MOVE  T,[JSR  ERROR]
        MOVEM T,42          ;Channel 1 for error
        MOVE  T,[JSR  DATA]
        MOVEM T,44          ;Channel 2 for data
        MOVE  T,PNTR1
        MOVEM T,PNTR        ;Set up new IO block pointer
        CONO  LPT,2012      ;Clear printer, assign channels
        CONO  PI,2340       ;Turn on PI and channels
        JRST  @SETUP

```

End of clear function sets  
Done, requesting a data  
interrupt.

```

PNTR1:  0
PNTR:   0

ERROR:  0
        CONO  LPT,2          ;Drop error request by dropping error
                                ;PI assignment
        .
        .
        JEN   @ERROR

DATA:   0
        CONO  LPT,12        ;Reassign error channel
        BLKO  LPT,PNTR      ;Send out word
        CONO  LPT,0         ;Turn off printer
        JEN   @DATA

```

## 4.2 PLOTTER XY10

The XY10 plotter control interfaces the PDP-10 central processor to various plotters that use cartesian coordinates. The models most frequently used are manufactured by Calcomp, but others can be accommodated. The following lists the type and paper size of the most commonly supplied Calcomp models.

<i>Calcomp model</i>	<i>Type</i>	<i>Paper size in inches</i>
502, 602	Bed	31 × 34
518, 618	Bed	54 × 72
563, 663	Drum	29½ × 1440
565, 665	Drum	11 × 1440

These are high accuracy, incremental digital plotters that produce fine quality ink plots of computer-generated data. Bidirectional stepping motors provide individual increments of motion in either coordinate or both at once. The program draws a continuous sequence of line segments by controlling the relative motion of pen and paper with the pen lowered, and it can raise the pen for repositioning.

Motion in  $y$  is movement of the pen carriage along a pair of rods. Motion in  $x$  is movement of the entire carriage-and-rod mechanism on a bed plotter, movement of the paper underneath the carriage on the drum type. On a bed plotter the coordinate directions are the standard ones when viewing the device from the front: positive  $x$  to the right, positive  $y$  to the back. The coordinate system on a drum is in the standard orientation when the viewer is standing at the right side, unrolling the paper from the drum with his left hand. In other words positive  $y$  is movement of the pen from right to left across the drum, positive  $x$  is drum rotation downward at the front (drawing a line toward the paper supply roll at the back).

The step sizes and plotting speeds available with the various Calcomp models are the following.

Two of these are standard models with the plotter control: the XY10A has the 565, the XY10B has the 563.

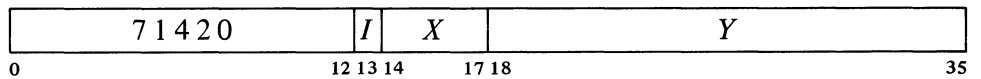
<i>Model</i>	<i>Step size</i>	<i>Plotting speed in steps per second</i>
502	<i>All sizes</i>	300
	.005 inch	200
518	.002 inch	450
	.1 mm	200
	.05 mm	400
	.010 inch	200
563	.005 inch	300
	.1 mm	300
565	<i>All sizes</i>	300
602	<i>All sizes</i>	450/900

Calcomp plotters in the 600 series have two step sizes and two plotting speeds: a switch at the back selects the step size, delay settings in the plotter control determine the speed.

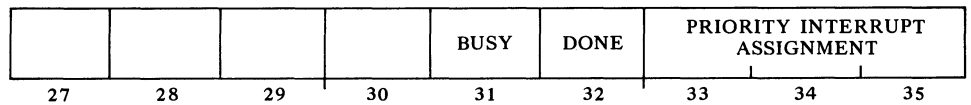
	.005/.0025 inch	200/400
618	.002/.001 inch	450/900
	.1/.05 mm	200/400
	.05/.025 mm	450/900
	.010/.005 inch	350/700
663	.005/.0025 inch	450/900
	.0025/.00125 inch	450/900
	<i>All sizes</i>	450/900

The program can draw any complete figure by giving a string of DATAOs, each of which supplies the information for one step. The plotter device code is 140, mnemonic PLT. A second plotter would have device code 144.

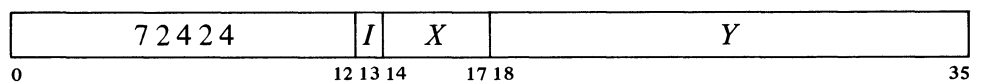
**CONO PLT, Conditions Out, Plotter**



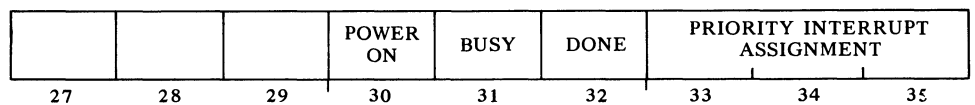
Set up the plotter control register according to bits 31–35 of the effective conditions *E* as shown (a 1 in a flag bit sets the flag, a 0 clears it).



**CONI PLT, Conditions In, Plotter**

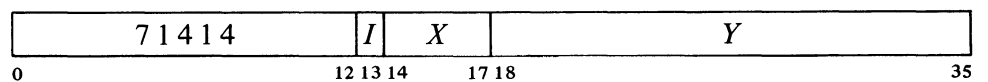


Read the status of the plotter into bits 30–35 of location *E* as shown.



Power On is not available on all plotters.

**DATAO PLT, Data Out, Plotter**



Clear Done, set Busy, and move the pen as specified by bits 30–35 of the

contents of location *E* as shown (a 1 in a bit produces the indicated motion, a 0 has no effect).

RAISE PEN	LOWER PEN	- $\Delta X$ (DRUM UP)	+ $\Delta X$ (DRUM DOWN)	+ $\Delta Y$ (CARRIAGE LEFT)	- $\Delta Y$ (CARRIAGE RIGHT)
30	31	32	33	34	35

A CONO need be given only to change the PI assignment; DATAO places the plotter in operation by supplying plotting data. After sufficient time has elapsed for the device to carry out the specified action, the control clears Busy and sets Done, requesting an interrupt on the assigned channel.

To avoid drawing line segments shorter than one step, do not raise or lower the pen in the same DATAO that calls for *xy* motion. The consequences of specifying contradictory movements cannot be predicted.

**Timing.** Lowering the pen takes 60 ms, raising it takes 10 ms. The time required to move one step in either or both coordinates depends on the plotting speed as follows.

<i>Plotting speed in steps per second</i>	<i>Time per step in ms</i>
200	5.00
300	3.33
350	2.86
400	2.50
450	2.22
700	1.43
900	1.11

**EXAMPLE.** The plotting commands sent out by this program are contained six to a word in WC words beginning at location BUFFER. The interrupt routine uses one accumulator which is shared with the main program and other channels.

```

CONSZ  PLT,7           ;Wait until previous run finished as
JRST   .-1            ;indicated by no PI assignment
MOVE   T,[JSR DATA]
MOVEM  T,50           ;Set up channel 4
MOVEI  T,WC*6         ;Set up count for plotting commands
MOVEM  T,COUNT
MOVE   T,[POINT 6,BUFFER] ;Initiate byte pointer
MOVEM  T,CHARP
CONO   PLT,4          ;Assign channel
CONO   PI,2210        ;Turn on PI and channel
DATAO  PLT,PUP        ;Raise pen to trigger first interrupt
:
:

```

The asterisk is the sign for multiplication in MACRO.

POINT is a pseudoinstruction that causes MACRO to generate a byte pointer from the three arguments that follow it. In order these arguments are the byte length in decimal, the address of the location containing the byte, and

```

DATA:      0
           SOSGE  COUNT      ;Is plot finished?
           JRST   DATA1     ;Yes
           MOVEM  T, TSAVE    ;Save T
           ILDB   T, CHARP    ;Get next plotting command
           DATAO PLT, T      ;Plot point
           MOVE   T, TSAVE    ;Restore T
           JEN    @DATA

PUP:       40
TSAVE:     0
COUNT:    0
CHARP:     0

DATA1:     CONO   PLT, 0      ;Disconnect plotter from interrupt
           DATAO PLT, PUP    ;Raise pen
           JEN    @DATA
    
```

the position of the rightmost bit of the byte as the decimal number of the bit in the word. If the last argument is omitted, MACRO takes it as -1; in other words, after being incremented the pointer will point to the first byte. The left half of the pointer generated here is 440600.

### 4.3 CARD READER CR10

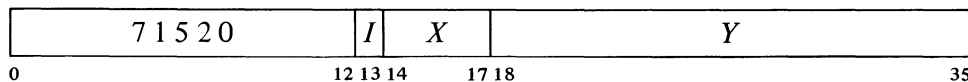
The card reader handles standard 12-row 80-column cards at maximum speeds from 300 to 1200 cards per minute depending on the model. Once started, an entire card is read column by column. The reader supplies each column to the processor as twelve bits corresponding to the column punch and also in a more compact form. The program can translate the column data in any way it wishes, but the standard DEC character representations and the translation to 7-bit ASCII made by the Monitor are given in Appendix B. Of course the data can simply be in binary at three columns per word (a 7 and 9 punch in the first column is the traditional indication that the rest of the card contains binary data).

The reader is available in four models with differing speeds and capacities.

<i>Reader</i>	<i>Cards per minute</i>	<i>Hopper/stacker capacity</i>	<i>Model type</i>
CR10A/B	A: 1000 B: 833	1000	Table
CR10D	1000	950	Table
CR10E	1200	2200	Floor
CR10F	300	550	Table

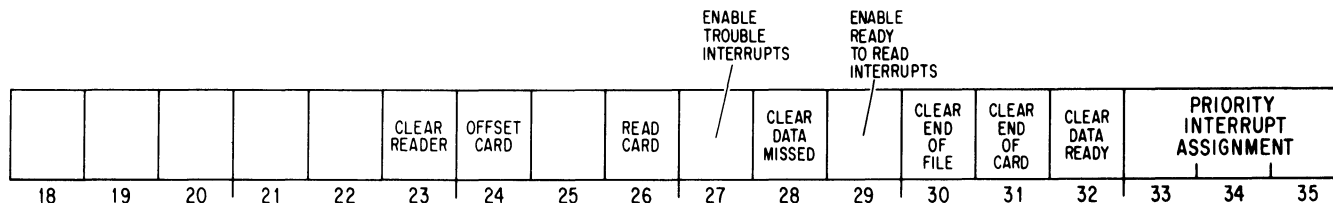
Models A and B are actually the same machine: the A version runs on 60 Hz power, the B on 50 Hz – and the speed depends on the line frequency. For the other models the 60 or 50 Hz version is indicated respectively by A or B appended to the model number, and line frequency has no effect on timing.

#### CONO CR, Conditions Out, Card Reader



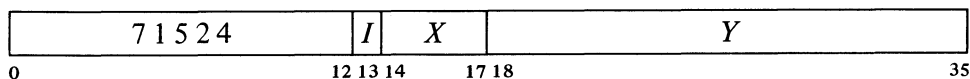
Assign the interrupt channel specified by bits 33–35 of *E* and perform the

functions specified by bits 23–32 as shown (in bits 27 and 29 a 1 enables the given flag to interrupt, a 0 disables it; in all other bits a 1 produces the indicated function, a 0 has no effect).

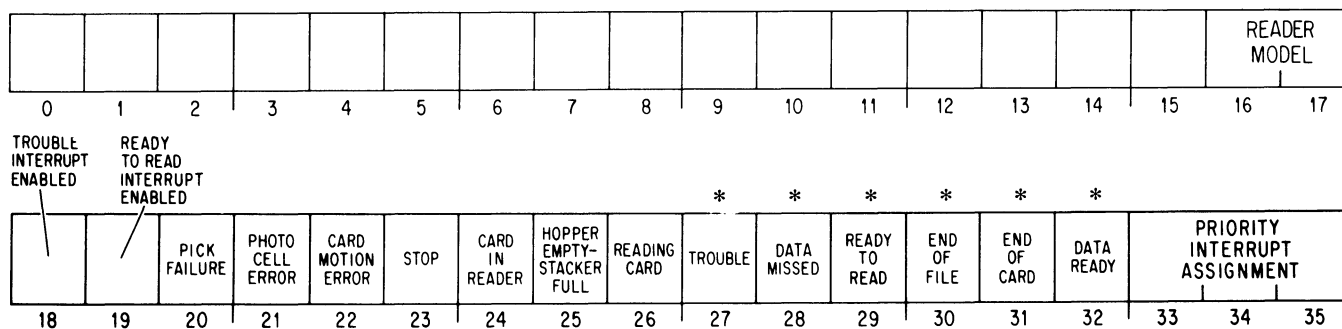


- 23 Dismiss the PI assignment (assign zero); clear flags Reading Card, Data Missed, End of File, End of Card, Data Ready, Trouble Interrupt Enabled, Ready to Read Interrupt Enabled; clear the card column buffer; and disable any read command given by a CONO if the reader has not yet started the card. If any action specified by the rest of the CONO bits conflicts with these actions, the clear function has precedence.
- 24 *CR10A/B only*: If a card is currently being processed in the reader (Card In Reader, CONI bit 24, is 1), offset it when it is placed in the stacker. The card will actually stick out about a half inch from the rest of the stacked deck.

**CONI CR, Conditions In, Card Reader**



Read the status of the reader into the right half of location *E* as shown.



\*These bits cause interrupts.

Interrupts are requested on the assigned channel by the setting of Data Ready, Data Missed, End of Card, End of File, and if enabled, Trouble and Ready to Read.

16-17 These bits identify the type of reader connected to the hardcopy control.

0	CR10A/B
1	CR10D
2	CR10E
3	CR10F

Note that CONSZ and CONSO can test only bits 18-35. To test bits 16 and 17 the program must give a CONI CR, AC and then use a test instruction [§2.8].

20 The reader has received a read command but has failed to bring in a card from the hopper.

21 The reader has failed to read a card properly and maintenance is probably required. The program should be dubious of any data taken from the card being read when the error occurred.

22 A card has failed to move properly through the reader (it has probably slipped). The program should be dubious of any data taken from the card being read when the error occurred.

23 Reader power is on but the reader is or soon will be unavailable to the program either because the operator has pressed the stop button or there is a trouble condition (bit 27). If Stop is set while a card is being read, the reader usually finishes it; only a power failure can stop the reader in the middle of a card.

24 The reader has brought a card in from the hopper and has not yet finished reading it. The program can give a CONO offset command while this bit is 1.

26 The reader has accepted a read command and has not yet finished reading the card.

27 Bit 20, 21, 22 or 25 is 1. If bit 18 is also 1, the setting of Trouble requests an interrupt on the assigned channel.

Any condition that sets Trouble also sets Stop (bit 23) and the reader will stop at the end of the current card (of course a pick failure prevents the reader from even starting a card). Although a 1 in bit 27 does not necessarily imply an error or malfunction, it always requires operator intervention. If bit 25 is 1 it is very likely that the only trouble is the hopper is empty or the stacker is full.

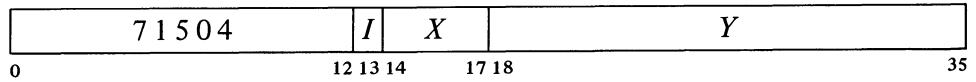
28 The program failed to retrieve a column of data before the next column was loaded into the buffer by the reader.

29 The reader is ready to accept a read command. If bit 19 is 1, the setting of Ready to Read requests an interrupt on the assigned channel.

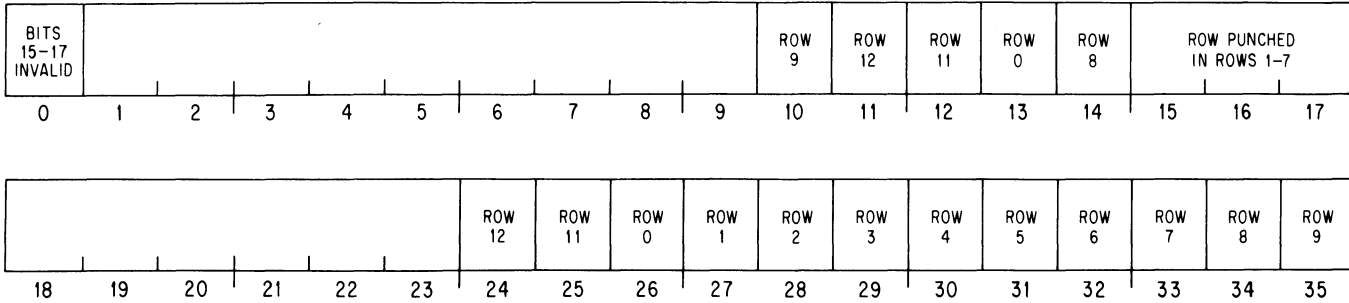
30 The reader has stopped (probably because the hopper is empty) and the operator has pressed the end-of-file button.

The usual procedure is to put an end-of-file card at the end of the deck rather than use the button. Actually the button can be used to signal the program for any purpose provided the reader is off line (stopped).

**DATAI CR, Data In, Card Reader**



Clear Data Ready and transfer two versions of the contents of the card column buffer into bits 10–17 and 24–35 of location *E* where the correspondence of card rows to bit positions is as shown.



The arrangement in bits 10–17 simplifies many standard card decoding procedures. *Eg* having the card character represented in eight bits allows conversion to ASCII with a table containing only 256 entries (as against 4096 entries for 12-bit characters). With a deck containing integers, bits 14–17 are the BCD representation of the numbers 0–8.

If bit 0 is 0, bits 15–17 hold the octal row number of the single punch in rows 1–7 (zero indicates no punch). However, if bit 0 is 1, there is more than one punch in rows 1–7 and bits 15–17 are meaningless.

If the program does not retrieve the final column and a CONO that starts a new card does not clear Data Ready, Data Missed will be set by the first column in the new card.

The program must give a CONO with a 1 in bit 26 to start every card. This read card command waits until the reader is ready, at which time Reading Card sets and the reader card cycle begins. Movement of a card in from the hopper sets Card in Reader. As each column is loaded into the buffer, Data Ready sets, requesting an interrupt on the assigned channel. The program must respond with a DATAI to transfer the column to memory and clear Data Ready. If Data Ready is still set when the next column is loaded into the buffer, Data Missed is set, requesting a second interrupt.

After all eighty columns have been read, Card in Reader goes off, clearing Reading Card and setting End of Card, which requests an interrupt. The card then moves out to the stacker, and when the device is ready to begin a new card cycle, Ready to Read goes on, but only if no new read card command has been given. If a read card command is already waiting when the reader becomes ready, it simply accepts the command and Ready to Read remains off. If no new command is waiting, Ready to Read goes on, requesting an interrupt if enabled (CONI bit 19 is 1), and it goes off automatically when a new command is given.

It is generally most convenient to use data interrupts only to handle data, and to give a CONO to read the next card after an End of Card interrupt. With the CR10D, E and F, however, Ready to Read comes on at the same time as End of Card. To save time, the program can give a new CONO read



card command right after retrieving the data from the final column. Then End of Card can be ignored, and there will be no Ready to Read interrupt.

**Timing.** Of interest to the programmer at the beginning of each card are the time from the turnon of Reading Card to the turnon of Card in Reader, assuming the first pick attempt is successful, and then the time to the first Data Ready. Subsequent columns are ready at fixed intervals, which determine the time after each setting of Data Ready within which the program must give a DATAI in order not to miss the data from the column. When End of Card sets after the final Data Ready, Card in Reader and Reading Card both clear. The bottom number in each column of the table below is the time after End of Card the program has to give a new CONO read card command to keep the reader going at the nominal maximum rate. With the CR10A/B, Ready to Read goes on at the end of this period if no new command appears; with the other readers, Ready to Read goes on with End of Card if no new command has already appeared.

	<i>CR10A</i>	<i>CR10B</i>	<i>CR10D</i>	<i>CR10E</i>	<i>CR10F</i>
Time per card in ms	60	72	60	50	200
Reading Card → Card in Reader in ms	18	21.6	15	14	24
Card in Reader → first Data Ready in ms	1.8	2.2	1.9	1.1	2.6
Each Data Ready to next in $\mu$ s	370	444	478	405	2014
DATAI window after Data Ready in $\mu$ s (otherwise Data Missed)	350	420	450	385	1900
Total first to last Data Ready in ms	29.2	35	37.7	32.1	158
Final Data Ready → End of Card in ms	1.8	2.2	1.9	1.6	8.0
Time for CONO after End of Card for nominal maximum rate in ms	9.2	11	0	0	0

When the last card in a deck is read, the hopper empty signal goes on well after the final Data Ready but before End of Card, except in the CR10A/B where it is simultaneous with End of Card.

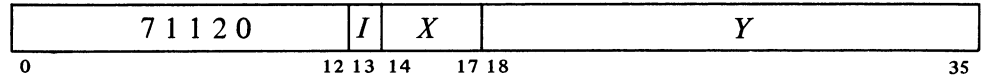
#### 4.4 CARD PUNCH CP10

The card punch handles standard 12-row 80-column cards at speeds up to 200 cards per minute if all eighty columns are punched, 365 cards per minute if only the first sixteen columns are punched. The processor must supply each column to the punch as twelve bits, and the program can generate this data by any procedure it wishes; the standard DEC character representations and the translation from ASCII made by the Monitor are given in Appendix B. Of course the data can simply be in binary at three columns per word (punching rows 7 and 9 in the first column is the standard procedure for indicating that the rest of the card contains binary data).

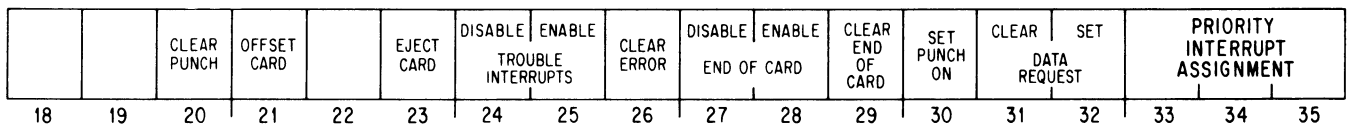
A card is taken from the hopper only when the program supplies data for the first column. In the interface is a 12-bit buffer to which the processor

sends each column, but the punch has a 48-bit buffer, and it punches four columns at a time from each set of four 12-bit bytes sent through the interface. The program can send a card to the stacker after punching any number of columns. The punch device code is 110, mnemonic CDP.

**CONO CDP, Conditions Out, Card Punch**

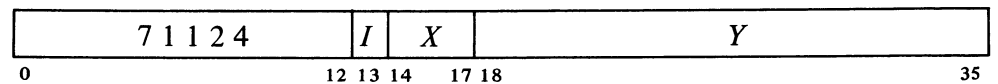


Assign the interrupt channel specified by bits 33–35 of the effective conditions *E* and perform the functions specified by bits 20–32 as shown (a 1 in a bit produces the indicated function, a 0 has no effect).



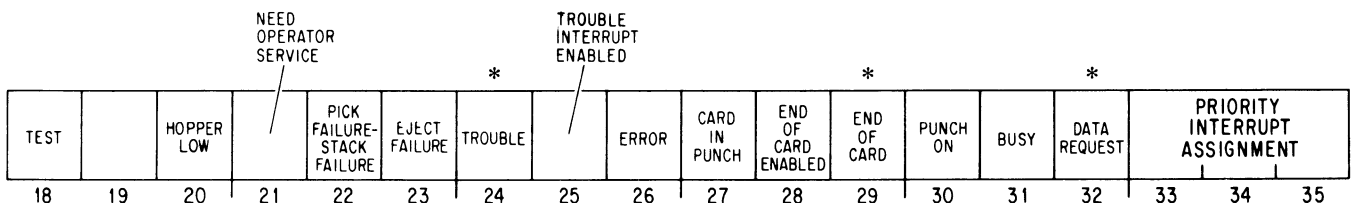
- 20 Clear flags Trouble Interrupt Enabled, Error, End of Card Enabled, End of Card, Punch On, Busy, Data Request; clear the card column buffer. If any action specified by the rest of the CONO bits conflicts with these actions, the other bits have precedence.
- 21 If a card is currently being processed in the punch (Card in Punch, CONI bit 27, is 1) or was ejected less than 3 ms ago, offset it when it is placed in the stacker. The card will actually stick out about a half inch from the rest of the stacked deck.
- 23 If a card is currently being processed (Card in Punch, CONI bit 27, is 1), punch whatever data is in the 4-column buffer and then eject the card. Ejection moves a card through the punch head assembly four times as fast as punching blank columns.

**CONI CDP, Conditions In, Card Punch**



\*These bits cause interrupts.

Read the status of the punch into the right half of location *E* as shown.





Setting Punch On turns on the punch motor, but only a DATAO can pick a card. Since DATAO also sets Punch On, the program can initiate punch operations while supplying data, but the usual procedure is to set Punch On while giving other initial conditions.

When the punch is ready to take a card from the hopper it sends a ready signal to the interface. This sets Data Request, which requests an interrupt on the assigned channel. To pick a card the program must respond with a DATAO, which supplies the first column, clears Data Request, and sets Punch On and Busy. The interface then sends the column to the 4-column punch buffer and clears Busy. While the punch is picking a card it also makes three more data requests to each of which the program must respond with a DATAO. When the card is properly registered in the punch head assembly, Card in Punch sets. When this flag has set and the program has supplied the first four columns, the device punches the four columns simultaneously (a 1 sent to the column buffer produces a hole in the card). The punch then continues in this fashion making four data requests for each set of four columns.

Punch On clears when the program gives an eject command. This causes the device to punch whatever is in its 4-column buffer and the card then moves out to the stacker. If the punch has already sent a ready signal, the CONO that ejects should also clear Data Request. If End of Card has been enabled by a 1 in CONO bit 28, the eject command sets it, requesting an interrupt. If no eject command has been given by the time data is supplied for column 80, End of Card sets anyway if it is enabled (producing an interrupt request), but the card remains in the punch head assembly until an eject command is given. The actual ejection of a card clears Card in Punch.

**Timing.** If Punch On is set when the punch motor is off, the first ready signal is delayed about 120 ms while the motor gets up to speed. When the motor is on, Card in Punch sets about 60 ms after the DATAO that sends the first column for a card. While the card is in the head assembly, punching is synchronized to a punch cycle of 11.1 ms. About 30  $\mu$ s elapse from each DATAO to the next Data Request, but after the first request the program has the full punch cycle time to supply all four columns and keep punching at the maximum rate; after that punching is delayed until the next cycle.

Giving an eject command clears Punch On and sets End of Card after 5  $\mu$ s, but Card in Punch does not clear until the card leaves the head assembly; this takes about 25 ms plus 2.8 ms for each set of four columns skipped over. After Card in Punch clears, about 30  $\mu$ s elapse before the punch indicates that it is ready to pick another card from the hopper, at which time the program should give a DATAO to pick another card at the maximum rate. (Of course the first DATAO can be given right after the eject command, and the punch will then pick another card automatically without setting Data Request for the first column.) When the final card is punched, the hopper empty signal is simultaneous with End of Card. If Punch On remains clear for about 30 seconds, the motor turns off.

If the program gives a DATAO to turn on the motor, the initial ready from the punch takes the first column from the column buffer but does not set Data Request. When that flag does set, the punch is ready for the second column.

If the program does not eject before the punch starts punching columns 77-80, it makes another data request. The program can then supply two more columns, which will be punched in the margin of the card.

#### CAUTION

Any data that is given but not punched (*eg* the first column(s) when there is a pick failure) is usually lost when the punch goes off line. Hence the program should always start with the first column of a card when the punch is restarted.

# 5

## Data Interfaces

The equipment described in this chapter is used for transferring data directly between memory and an in-out device or between the DECsystem-10 and some other computer.

A data channel can be used with high speed IO equipment. The channel moves data words between the IO device control and memory, thus bypassing the central processor. A disk or drum must use the data channel unless it has an equivalent data interface built into its own control. Use of the channel with the standard magnetic tape is optional; but the DECTape and all devices already described do not use the data channel — all of their transfers must be made individually under program control over the IO bus.

An interface from one computer to another not only handles the transfer of a block of words but also compensates for the difference in word size. The DA10, which interfaces the PDP-10 to a 12- or 18-bit computer, uses the IO bus and therefore depends on the program for each transfer.

### 5.1 DATA CHANNEL

The maximum rate for data transfers between external devices and core memory would be no greater than 150,000 words per second if the transfers were executed under program control. This rate would tie up the central processor completely. To allow much greater transfer rates and to free the central processor for more useful programs while in-out operations are in progress, a data channel can be installed in parallel with the central processor. When the data channel and the central processor are both accessing the same memory, they must compete for memory cycles, but when they are accessing different memories the data channel can execute transfers at the full memory cycle rate (a million words per second with the MA10). An efficient procedure is to use the data channel to bring information for the next program into a memory (*eg* memory 1) while the processor is running a program in another memory (*eg* memory 0); then while the processor executes the program in memory 1, the channel can be used to output the results of the previous program from memory 0 and bring in the next program.

The data channel is essentially a small processor that runs on its own very limited program taken from memory. This program is a set of control words. Most of these control words are simply addresses and word counts for controlling data transfers, but some are equivalent to jump and halt instructions.

The data channel is connected to memory either by its own memory bus or through a multiplexer sharing a common bus with other processors; a channel bus connects the data channel to those IO devices that use it.

The central processor program cannot affect the data channel directly because the channel is not connected to the IO bus and therefore has no instructions; instead the program sets up the device to use the channel. Some of the conditions that the program can supply via a CONO to the device are actually for the data channel; similarly status conditions from the channel are supplied to the program through the device. Moreover a device connected to the data channel requires no data transfers over the IO bus, so the DATAO and DATAI for the device are often used for control and status information.

Although several devices can be connected to a single channel bus, the data channel operates with only one device at a time — once the channel begins a series of transfers with a given device it services only that device until the entire series is complete. The priority among devices for gaining use of the channel is determined simply by physical location on the channel bus: the closer a device, the higher its priority. To place a device in operation with the channel, the processor sends all of the necessary control information for both the device and the channel to the device via the IO bus. When the channel is free and the device has priority, it automatically connects to the channel and remains connected until the channel operation is terminated by either a control word taken from memory, an error in the channel or a signal sent by the device.

The illustration on the next page shows the organization of the data channel. All transfers to, from, through and within the channel except for the transfer of addresses over the memory bus address lines are actually made through a memory buffer, which is not shown. Controlling the operation of the data channel are three 18-bit counters that count data words, data addresses and control word addresses. The first of these counters determines the number of words transferred in a single block, the second supplies consecutive memory addresses for the data transfers in a block, and the last supplies the address for retrieval of the next control word when a block is complete. Following each memory access, the counter from which the address for the access was taken is incremented by one. Thus, the data transfers in a block are made to consecutive locations and control words are also taken from consecutive locations.

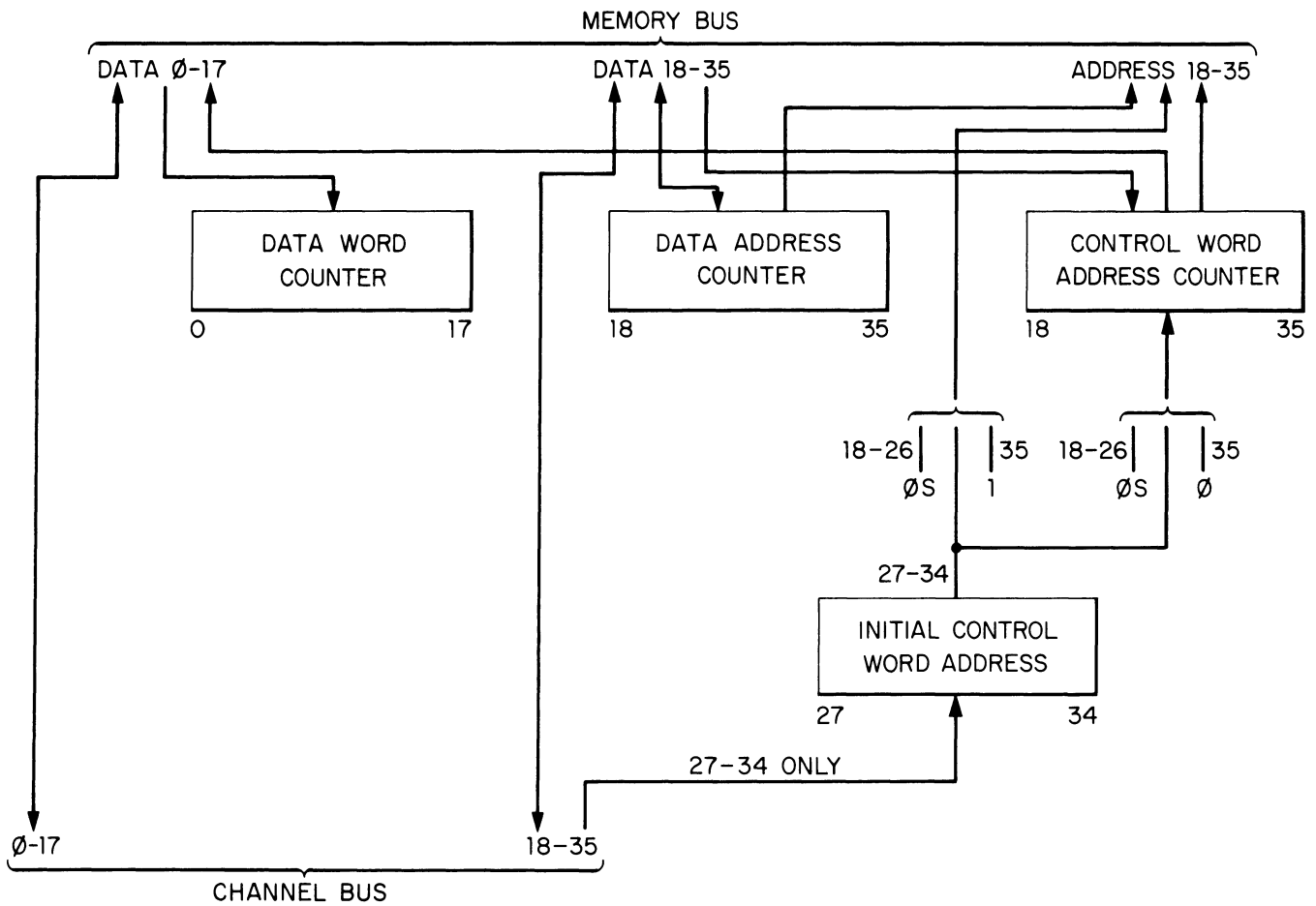
When a device connects to the data channel, it supplies an initial control word address, which must be an even number between 2 and 776 (in other words the address occupies bits 27–34 of a word sent by the program to the device). This address is saved in a register in the channel and is also loaded into the control word address counter from which it is sent to memory for the retrieval of the first control word. When a control word is brought into the memory buffer, its left and right halves are first inspected to determine what the channel must do. The format of these control words is as follows.

The word counter can also be used for other purposes, such as counting records spaced on tape, but this is always done by counting throwaway words.

<i>Left half</i>	<i>Right half</i>	<i>Meaning</i>
Nonzero	Nonzero	Normal block transfer. Left half is negative of word count, right half is one less than initial data address.
Nonzero	Zero	Normal block transfer for output, but for input the words received from the device are not sent to memory – they are discarded.
Zero	Nonzero	Jump. Right half is sent to control word address counter to get another control word.
Zero	Zero	Halt. The channel terminates operation and disconnects from the device.

The control word for a block transfer is the same as the pointer for a pushdown or block IO instruction [see page G3] and can be generated by an IOWD pseudoinstruction.

Note that the initial data address would usually be 20 or greater as the central processor cannot normally access locations below 20 if the accumulators are in a fast memory. Location 0 can be used only by counting around from high addresses.



CHANNEL DATA FLOW

If the left half is nonzero, the word supplies information for a block transfer and its left and right halves are sent to the data word counter and the data address counter respectively. The channel logic then increments both address counters so that the data address counter points to the first location in the block and the control word address counter points to the next control word. Data channel operation then begins, and as each word is transferred to or from the location specified by the data address counter, the logic increments both data counters. When the word counter overflows, the channel gets another control word from the location specified by the control word address counter and continues operation as specified by the new control word. A control word with a nonzero left half and a zero right half causes a normal block transfer for output; however, for input it functions in the normal way with the device but simply throws away the data received without accessing memory. This feature can be used for skipping blocks on a tape or disk without requiring program intervention at the device.

When the channel receives a zero control word, it terminates operation and disconnects from the device. At the same time the data channel stores a control word in the memory location whose address is one greater than the initial control word address, *ie* the next odd location following the even address initially given by the program to the device. However, this control word is not the same as the control words retrieved by the channel; the word stored has the present contents of the control word address counter in its left half, and the present contents of the data address counter in its right half. The channel stores a control word in this manner upon termination for whatever reason.

**Output Conditions.** The program can supply two control signals via the device.

*Write Control Word.* Upon receipt of this signal the channel waits until the next data transfer is complete and then writes a control word of the form described above in the location following that of the initial control word.

*Write Even Parity.* This signal is usually supplied as bit 35 of the word containing the initial control word address and is for maintenance only. The signal causes the channel to generate an even parity bit for each data word stored in memory, but the channel still checks all words retrieved from memory for odd parity and uses odd parity when storing control words.

**Input Conditions.** The channel supplies four conditions via status flags in the device.

*Control Word Parity Error.* If the data channel receives a control word with even parity from memory, it sends this signal to the device, terminates operation and disconnects from the device.

*No Such Memory.* If a memory addressed by the data channel does not respond within 100  $\mu$ s, the channel sends this signal to the device, terminates operation and disconnects from the device.

*Data Parity Error.* If the channel receives a data word with even parity from memory, it sends this signal to the device but continues normal operation.

Ordinarily the program would do this to determine how near to completion an operation is.



*Control Word Written.* When the data channel finishes storing a control word at the request of the device, it sends this signal to the device.

EXAMPLES. Let  $A$  be the initial control word address. This control word list is used for an output operation; it causes the channel to take 1024 ( $2000_8$ ) words from memory beginning at location 5000 and send them to the device.

<i>Location</i>	<i>Contents</i>
$A$	776000 004777
$A+1$	0 0

Note that this example uses the location following the initial control word so the program cannot signal the channel to write a control word during its execution.

It is usually better to leave location  $A+1$  free and use a jump as in this example.

<i>Location</i>	<i>Contents</i>
$A$	0 001000
1000	777700 001777
1001	0 0

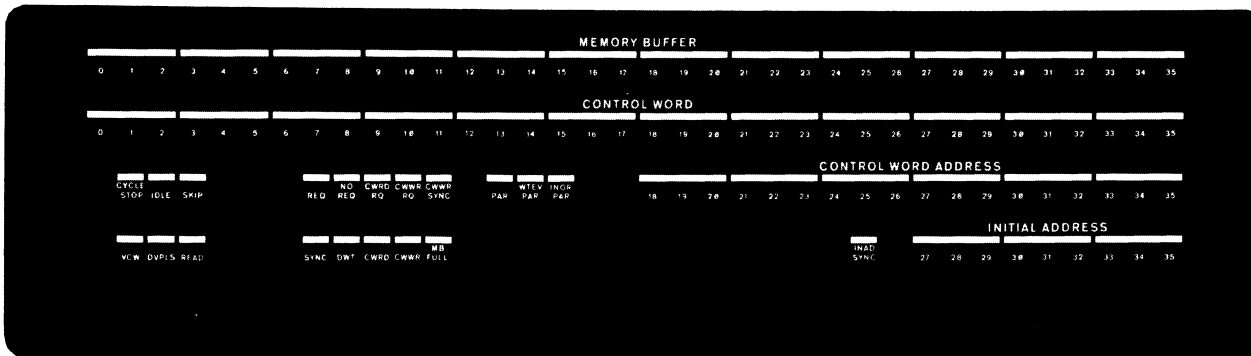
For an input operation this control word list causes the channel to accept sixty-four words from the device and store them in locations 2000–2077.

Suppose the following control word list is used for an input operation.

<i>Location</i>	<i>Contents</i>
$A$	0 000500
500	777700 000777
501	777000 004777
502	777300 0
503	777000 005777
504	0 0

This sequence causes the channel to store the first sixty-four words supplied by the device in locations 1000–1077, the next 512 words go to locations 5000–5777, the next 320 words are thrown away, and the final 512 words are stored in locations 6000–6777. Hence four blocks of data are involved, but the channel skips the third block. Using this list for output would involve equivalent transfers in the opposite direction, but the third block of 320 words would be taken from *core* locations 1–500.

**Operation.** Registers and a number of control signals for the data channel are displayed on an indicator panel. The lights for the left and right halves of the control word display the current contents of the word counter and the data address counter respectively. The other register lights display the memory buffer, the address of the next control word, and the initial address (the bit 35 light is not used).



Indicator Panel,  
Data Channel

The three lights at the left end of the third row show the status of the channel: the left light indicates the channel is ready to terminate operations, the second light indicates the channel is not in use, and the third light indicates that the control word for the current block transfer had a zero right half, *ie* if the current operation is for input, the channel is skipping a block on the device. The next set of five lights in the third row and the five lights directly below them indicate the type of memory cycle being requested and the specific type of individual transfer being made. The lights in the middle of the third row display parity information: PAR is the parity bit generated for a word to be written in memory; the second light is on when the channel is generating even parity for data words; the third light indicates when a switch at the memory is causing the channel to ignore parity, and hence the channel can discover no parity errors in words taken from memory. Mounted in the logic is a toggle switch that allows the operator to select odd parity for the channel regardless of the conditions given by the program via the device. The switch is located above panel D; it should be set to the left to lock out even parity.

The lights at the left in the bottom row indicate the type of data channel operation. VCW indicates a block transfer (*ie* the control word is valid, being neither jump nor halt). The READ light indicates an input operation (the middle light is the device pulse, which indicates that the device is ready to receive output or has sent input data that the channel has not yet transferred to memory).

Located behind the doors on a bracket between panels C and D are a push-button that clears the data channel and the toggle switch for locking out even parity writing. On two brackets between panels A and B are maintenance switches for local control of the channel; the left switch on the left bracket must be set to the right for proper operation of the channel in the system (when set to the left this switch enables the other switches for test operation without a device).

### 5.2 TWELVE- AND EIGHTEEN-BIT COMPUTER INTERFACE DA10

This unit interfaces a PDP-10 processor to a variety of 12- and 18-bit DEC computers. It is connected to the PDP-10 IO bus and the IO bus of a small computer, and from either bus it appears as an interface to a device. The small computer must have a negative bus, which is always either standard or available as an option.

The interface contains two 36-bit buffers, From Ten and To Ten, so that data transfers can be performed simultaneously in both directions. Data is moved between the PDP-10 and the interface in 36-bit words, but the small computer operates with only part of a buffer at a time depending on its own word size. The unit also contains four status flags, two for each computer, to indicate when it has data for the computer to retrieve (*ie* the input buffer is full) and when it is ready to receive data (*ie* the output buffer is empty). The flags for the PDP-10 are From Ten Empty and To Ten Full; the flags for the small computer are To Ten Empty and From Ten Full. The setting of any of these flags requests a priority interrupt in the associated computer.

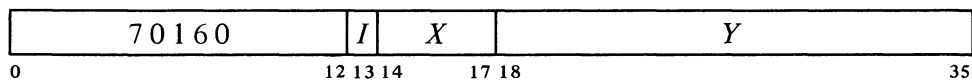
The interface must be set up to operate either with a 12-bit computer or with an 18-bit computer. For setup information refer to the *DA10 Maintenance Manual*.

The input buffer for the PDP-10 is the output buffer for the small computer, and vice versa.

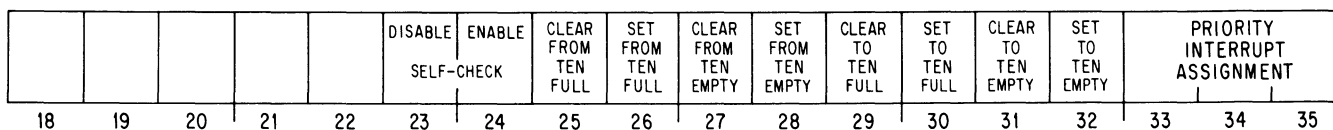
#### PDP-10 Instructions

The interface has the usual instructions for sending and reading conditions, but except for assigning a PI channel, it can be controlled entirely by DATAIs and DATAOs. The interface device code is 014, mnemonic CCI (computer-computer interface). A second interface would have device code 020.

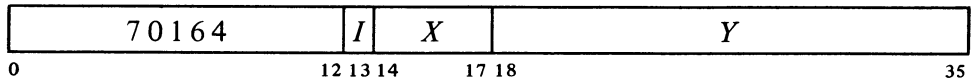
#### CONO CCI, Conditions Out, Computer-Computer Interface



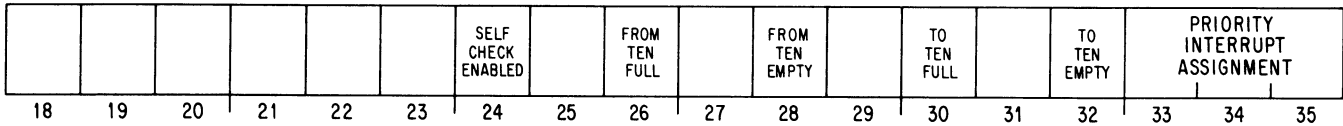
Assign the interrupt channel specified by bits 33–35 of the effective conditions *E* and perform the functions specified by bits 23–32 as shown (a 1 in a bit produces the indicated function, a 0 has no effect).



**CONI CCI, Conditions In, Computer-Computer Interface**

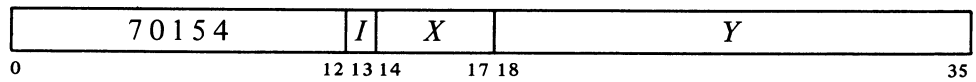


Read the status of the interface into bits 24–35 of location *E* as shown.



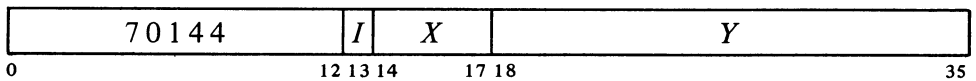
The setting of From Ten Empty or To Ten Full requests an interrupt on the assigned channel.

**DATAO CCI, Data Out, Computer-Computer Interface**



Load the contents of location *E* into the From Ten buffer. Clear From Ten Empty and set From Ten Full.

**DATAI CCI, Data In, Computer-Computer Interface**



Transfer the contents of the To Ten buffer into location *E*. Clear To Ten Full and set To Ten Empty.

**Twelve-bit Computer Instructions**

For operation with a 12-bit computer such as the PDP-8 or PDP-12, the data buffers are divided into three parts, 1, 2, and 3 (bits 0–11, 12–23, and 24–35 respectively). Because the 12-bit words must be handled individually, the interface uses three small-computer device codes, 35, 36, and 37. There are no standard mnemonics defined for instructions that use these codes, but because the DA10 is often a part of a DC68A communication system, the mnemonics defined in the *X680* software are given here.

In the following, the term “AC” refers to the 12-bit accumulator. The setting of To Ten Empty or From Ten Full requests an interrupt at the small computer.

<b>DAOSKP</b>	<b>DA10 Out Skip</b>	6361	
Skip the next instruction in sequence if To Ten Empty is set.			
<b>DAISKP</b>	<b>DA10 In Skip</b>	6371	
Skip the next instruction in sequence if From Ten Full is set.			
<b>DAOCLR</b>	<b>DA10 Out Clear</b>	6351	
Clear the To Ten buffer and To Ten Empty.			
<b>DALOD1</b>	<b>DA10 Load 1</b>	6354	
Load the contents of AC into bits 0–11 of the To Ten buffer.			
<b>DALOD2</b>	<b>DA10 Load 2</b>	6364	
Load the contents of AC into bits 12–23 of the To Ten buffer.			
<b>DALOD3</b>	<b>DA10 Load 3</b>	6374	
Load the contents of AC into bits 24–35 of the To Ten buffer, and set To Ten Full.			
<b>DARED1</b>	<b>DA10 Read 1</b>	6352	
Inclusive OR the contents of bits 0–11 of the From Ten buffer with the contents of AC in AC.			
<b>DARED2</b>	<b>DA10 Read 2</b>	6362	
Inclusive OR the contents of bits 12–23 of the From Ten buffer with the contents of AC in AC.			
<b>DARED3</b>	<b>DA10 Read 3</b>	6372	
Inclusive OR the contents of bits 24–35 of the From Ten buffer with the contents of AC in AC. Clear From Ten Full and set From Ten Empty.			

Note that these read instructions perform an inclusive OR. Hence, each complete 12-bit read operation actually requires an instruction pair: a CLA to clear AC, followed by the read.

### Eighteen-bit Computer Instructions

For operation with an 18-bit computer such as a PDP-9 or PDP-15, the data buffers are divided into two halves, whose contents are handled individually. The 18-bit computer device code for the interface is 22, and a 0 or 1 respectively in instruction bit 13 selects the left or right half of the buffer (bits 0–17 and 18–35). There are no standard mnemonics defined for instructions that use this device code.

In the following, the term “AC” refers to the 18-bit accumulator. The setting of To Ten Empty or From Ten Full requests an interrupt at the small computer.

**Skip if Ten Interface Empty** 702221

Skip the next instruction in sequence if To Ten Empty is set.

**Skip if Ten Interface Full** 702241

Skip the next instruction in sequence if From Ten Full is set.

**Clear Ten Interface** 702201

Clear the To Ten buffer and To Ten Empty.

**Load Ten Interface Left** 702204

Load the contents of AC into bits 0–17 of the To Ten buffer.

**Load Ten Interface Right** 702224

Load the contents of AC into bits 18–35 of the To Ten buffer, and set To Ten Full.

**Read Ten Interface Left** 702212

Clear AC and then transfer the contents of bits 0–17 of the From Ten buffer into AC.

**Read Ten Interface Right** 702232

Clear AC and then transfer the contents of bits 18–35 of the From Ten buffer into AC. Clear From Ten Full and set From Ten Empty.

A 1 in instruction bit 14 clears AC prior to the transfer. The instructions given are the forms that are most likely to be used. However,

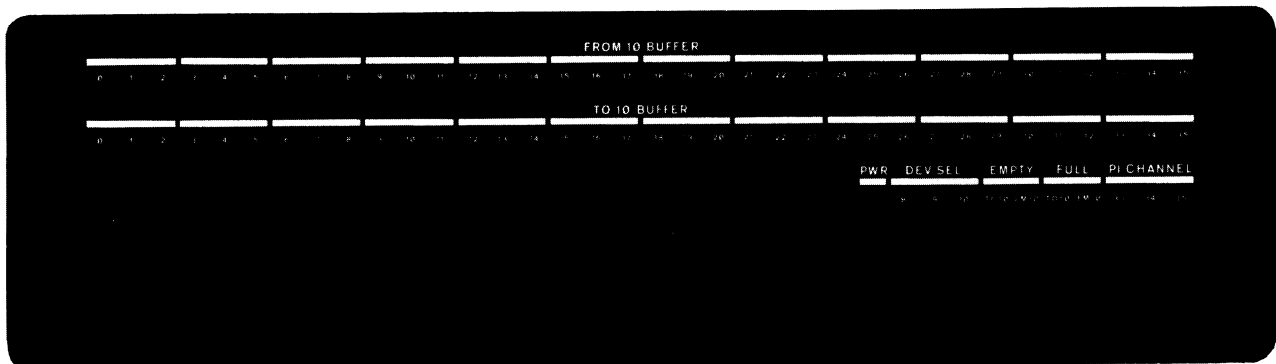
### Programming Considerations

Initially, a CONO need be given only to assign a PI channel. In the small computer, the interrupt flags are connected to the priority request line; the only requirement is that the interrupt be turned on.

For the transfer of data toward the small computer, the PDP-10 program must give a DATAO for each 36-bit word; this loads the word into the From Ten buffer, clears From Ten Empty and sets From Ten Full. The setting of the latter flag requests an interrupt in the small computer, which then uses its skip instructions to determine which device has requested service. To complete the transfer, the small computer retrieves its words from the buffer from left to right. The instruction that retrieves the right-hand word also clears From Ten Full and sets From Ten Empty, requesting an interrupt in the PDP-10. After the final word in a block is transferred, the program must give a CONO with a 1 in bit 27 to clear From Ten Empty and drop the interrupt request.

For transfers in the opposite direction, the small computer should first clear To Ten Empty and the To Ten buffer. The small words are then loaded into the buffer from left to right, where the instruction that loads the right-hand word also sets To Ten Full, requesting an interrupt in the PDP-10. The

Note that the small computer must always work from left to right. Hence, if single small words are being processed through the interface, they must be right-justified.



PDP-10 program responds with a DATAI that retrieves the 36-bit word from the To Ten buffer, clears To Ten Full, and sets To Ten Empty, requesting an interrupt in the small computer. After the final word in the block is transferred, the program must give a DAOCLR to clear To Ten Empty and drop the interrupt request.

The programmer must establish some sort of protocol to be used by the program in the two computers so the receiving program will know what is going on. *Eg* the first words of a block might be used to indicate the amount of data, where it should be stored, what it is for, and so on.

The top two rows of lights on the DA10 indicator panel display the contents of the 36-bit buffers. The device select lights in the bottom row

DA10 Indicator Panel

indicate which computers are currently selecting the interface (the 8 light represents any 12-bit computer, the 9 light any 18-bit computer). The remaining lights indicate when power is on and display the status flags and the PDP-10 PI channel assignment.

**Timing.** The DA10 can handle transfers at the maximum PDP-10 rate of 150,000 words per second. Whether or not this rate can be attained depends entirely on the speed of the small computer. Some small computers can actually operate faster than this, but others are much slower. No times are given with the instruction descriptions, as the time for each IOT type depends on the processor rather than the device. To determine the instruction times and the transfer rate that can be maintained, refer to the handbook for the appropriate small computer. The system designer must be careful to ensure that one computer cannot swamp the other with interrupt requests. This might occur particularly with a fast small computer, especially in an arrangement where it sends out data in its own word size rather than assembling PDP-10 words.

**Self-checking.** The DA10 contains circuitry that allows a PDP-10 maintenance program to check out the transfer logic with the interface disconnected from the small computer bus. Self-checking is enabled by giving a CONO with a 1 in bit 24. Then each DATAO not only loads the From Ten buffer but also triggers a sequence that moves the small computer words (left to right) through the bus logic to the To Ten buffer. The transfer of the right-hand word sets To Ten Full, requesting an interrupt, so the program can give a DATAI and check the word received against that originally sent.

The manner in which the data is moved (*ie* the word size) depends on the type of small computer with which the interface is set up to operate.



# 6

## Magnetic Tape

Two types of Magnetic tape equipment are regularly available for use in a DECsystem-10. One handles the large reels of half-inch tape that are standard throughout the industry; the other handles DECTape, of DEC's own creation. DECTape trades off some of the speed and storage capacity of standard magnetic tape for the convenience of paper tape. Its small reels are easy to handle and carry about, making DECTape desirable for program input. In addition, the redundancy used in recording helps to maintain the information intact despite heavy handling. The format of the tape allows single blocks to be replaced without affecting the rest of the tape, thus making it especially convenient for holding a library of commonly used routines.

The industry-standard magnetic tape may be used as backup storage for memory and is especially convenient for storing large amounts of data that need not be available to the system continuously. (A disk or drum can store information much faster, but physical storage of the information storing medium separate from its handling equipment is less convenient, and often impossible.)

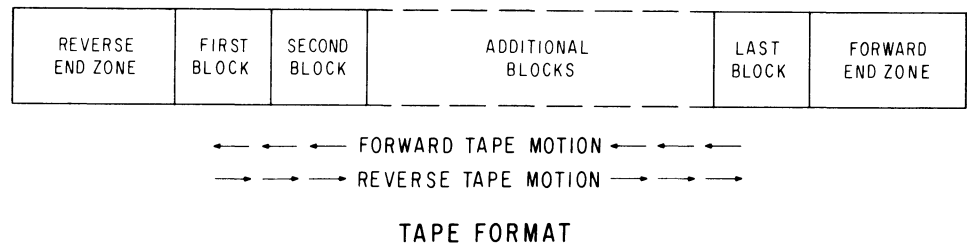
Both types of magnetic tape equipment automatically record error-checking information while recording data on tape. This provides a means of checking for possible data loss that results from using the tape as a storage medium.

### PART I DECTAPE

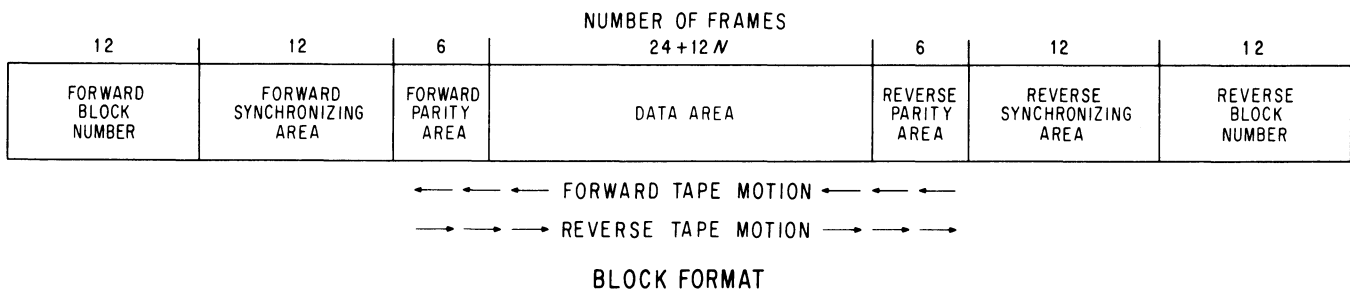
A DECTape system consists of a TD10 control and up to eight DECTape transports. Three of the tape transports can be mounted in the same cabinet with the control. Any number of tapes can be in motion simultaneously, but the control can monitor only one at a time. Both reading and writing can be done in either direction of tape motion, with an average data transfer rate between processor and tape of 400  $\mu$ s per 36-bit word. Each transport has two reels, which function as supply or takeup depending on the direction of tape motion. A full reel has 260 feet of  $\frac{3}{4}$ -inch, 1 mil magnetic tape and can store 2½ million data bits, three bits per frame (73,984 36-bit words in standard PDP-10 DECTape format).

### 6.1 TAPE FORMAT

From a programming point of view the tape has four tracks, one for format information – the “mark” track – and three for data. While handling the data tracks, the control uses the mark track internally to determine the position of the tape, *ie* which type of zone is present at the heads. The control expects to find the tape formatted as a series of contiguous blocks preceded and followed by an end zone.



Each block has seven zones. In the middle is a data area that has a minimum of twenty-four frames and can be expanded by any multiple of twelve frames; a block thus holds an integral number of 36-bit words. Blocks with different size data areas can be mixed on one tape. At either end of the data area are a block number (BN), a synchronizing area and a parity area. Twelve



frames at each end of every block are used for a block number; identifying blocks in this manner allows the data area of a block to be treated as addressable storage. Tape processing involves three types of functions: those that process only the block numbers, those that process only the data and parity areas, and those that process everything; these functions are respectively BN, Data and All functions. There are two functions of each type, for reading and writing, and every function can be executed in either direction.

A BN function processes only the block number at the beginning of each block; in other words it processes only the forward BN when the tape is moving forward, and the reverse BN when the tape is moving reverse. For convenience, the first end encountered in any block will be referred to as the leading end regardless of which direction the tape is moving – and similarly the other end is the trailing end. Hence a BN function processes only leading block numbers.

Data functions process the data areas, but when the control is processing a data area it also automatically processes the parity areas. The programmer need not be concerned with the parity areas as such: generation and checking of parity is entirely automatic. The programmer's only concern is a flag that indicates whether any errors have been detected in information read. Note that if only BN and Data functions are used, the synchronizing areas are not processed at all, but the program can take advantage of the time that elapses while the control is spacing past the synchronizing areas to change functions. The usual procedure is to search for a desired block by reading block numbers, and then switch to a read or write Data function to process the data area when the block is found.

With the All functions, the program can write or read all areas, except of course parity areas which are processed automatically just as in the Data functions. The All functions allow use of the synchronizing areas for special identification purposes or storage of large quantities of information without regard to the block structure, but often they are used only to write block numbers on a tape that has just been formatted.

The DECTape control processes data in terms of 36-bit words even though the tape has 3-bit frames. The twelve frames written from a given word are read as that same word regardless of the direction of tape motion. BN functions in different directions actually process different tape areas, *ie* they process numbers at opposite ends of the blocks. Data and All functions however are truly bidirectional; the only difference in processing a block in opposite directions is that the order of the words is reversed.

Besides the above functions there is a special function through which the program formats a blank tape by writing the mark track. As the mark track is written, a timing track is also recorded on tape. The programmer should not be concerned with this timing track since its generation and use are internal to the control. The timing track is, however, responsible for variations in the data transfer rate: when processing is in the same direction that the mark track was recorded, the transfer rate is a constant 400  $\mu$ s per 36-bit word; in the opposite direction the transfer rate varies from 480  $\mu$ s near the forward end zone to 320  $\mu$ s near the reverse end zone.

**Standard Format DECTape.** DEC supplies tapes with timing and mark tracks recorded on a transport whose write head has zero skew; these tapes may therefore be used on any transport where the skew is within specifications. A standard tape is formatted into 578 blocks numbered in octal from 0 to 1101. Each block has a data area of 128 words.

The forward BN for the first block and the reverse BN for the last block cannot be processed; but this is not important since in either case the block is being approached from the end zone.

When the control is writing a block, it writes 1s in the leading parity area, writes the data area, and writes parity information in the trailing parity area. When the tape is being read, the direction is of no significance; the control simply checks the parity of the two parity areas and the data area combined.

Internally the control handles 6-bit bytes, each of which occupies two tape frames. The frames thus contain octal digits, which are oriented so that the control encounters the most significant digit of each word first when tape is moving forward.

The Monitor actually reverses the order of the words handled in a transfer between memory and tape when the tape is going in reverse. Hence core data sent to tape and subsequently retrieved appears in the same orientation in core regardless of the direction in which the tape happens to be moving during writing and reading.

The mark track of a standard format tape has 200 BN space codes between each end zone and the block nearest it. With tape moving from the end zone, the control ignores these codes; with motion toward the end zone, the tape simply appears to have a string of leading block numbers for a block that is not there.

**Compatibility.** All DECTape controls except those used with the PDP-1, 4 and 7 generate and check parity automatically and use the same zone marks. Tapes are thus compatible from one of these computers to another (of course the number of words in a given block depends on the word length of the computer). In the PDP-4 and 7 the program handles parity and must therefore match the parity configuration of the PDP-10 or an error indication results when the tape is read. The TD10 generates even parity the length of a data area for corresponding bits in *6-bit* data bytes; the resulting 6-bit parity character is written in the first two frames of the trailing parity area, and the rest of the half word is filled with 0s. The PDP-4 is not fast enough to compute parity while processing the tape.

A discussion of the compatibility of PDP-1 DECTape and LINC tape with the PDP-10 is not included in this manual.

## 6.2 TAPE HANDLING CHARACTERISTICS

The program communicates with the DECTape control, which in turn governs all tape transports but communicates with only a single transport at a time. Once the program starts a tape function, the control operates automatically but transfers data only between itself and the selected tape; all data transfers between control and memory must be handled by the program. Initial conditions the program must supply include PI assignments, a tape function, the number of a tape unit, and the direction of tape motion. However, it is not enough simply to address a tape unit — the unit must also be selected. Once selected, a transport requires about 150 ms to bring the tape up to speed. This time is actually sufficient to stop a transport that is already running, but in the wrong direction, and to change direction and bring the tape up to a speed that allows data processing to take place in the opposite direction.

A 120 ms delay is built into the hardware to prevent the tape from being processed should it be already up to speed but in the wrong direction. Following the delay, processing can begin as soon as the tape is up to speed in the right direction.

Suppose transport *A* is running and we wish to use transport *B*. A CONO can be given that deselected *A* and selects *B*, but *A* will stay in motion and a deselected tape can run right off the reel. To stop *A* and select *B*, a CONO must be issued that selects *A* and stops it, followed by a second CONO that deselected *A* and selects *B*. The only way a deselected tape can be stopped (other than by selecting it) is for the program to stop all tapes simultaneously, including both the selected one (if any) and all deselected ones. The selected transport can be stopped individually by the program, and the transport always stops automatically when the end zone is encountered. To reselect a tape that is already moving in the right direction, the program can save time by inhibiting the start delay. Tape processing can then begin following a 1 ms delay to allow the selection relays to settle.

The time from a known to a desired position on the tape can be estimated with reasonable accuracy, so the program can leave one tape running while using another.

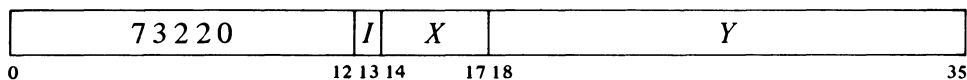
Suppose we have searched for and found a desired block number but wish to process the data area of the block in the opposite direction. To do this the program must give a CONO that turns the tape around. However, before the control begins the stop-turnaround-start sequence, it waits through a turnaround delay of 200 ms while the tape continues in the same direction with data processing inhibited. This delay ensures that when the tape is up to speed in the opposite direction it will still be beyond the desired block, *ie* the block will not have reached the tape head and hence can be processed in the new direction. In many cases this feature is not necessary. Suppose block 17 is required and block 32 has just been identified going forward. In this case, we simply want the tape to turn around as quickly as possible. By subterfuge, the program can eliminate the turnaround delay [§6.4].

If the tape is positioned at some arbitrary point and we wish to process block 0 forward, instead of checking block numbers, we can simply run the tape to the reverse end zone where it stops automatically. After the tape stops, do not simply turn the tape around or make it go forward, since it is unlikely that the tape will be in the correct position to process block 0 when the necessary speed is attained. Instead, the program should give two CONOs, where the first makes the tape go in reverse again, and the second turns it around. The program instructions are too quick for the system to respond to the end zone, so the tape starts up in reverse and continues in reverse through the turnaround delay. This makes the tape go far enough into the end zone so the first block can be processed.

### 6.3 INSTRUCTIONS

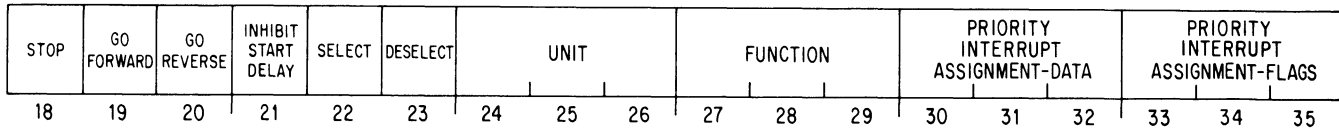
The DECTape control has two device codes, 320 and 324, mnemonics DTC and DTS. Device code 320 is used for transmitting data and command information; device code 324 is used for enabling interrupts and reading status. (There are also data instructions that use the status code, but they are for maintenance purposes only.) A second DECTape system would have device codes 330 and 334.

#### CONO DTC, Conditions Out, DECTape Command



Clear Active, Function Stop, Parity Error, Data Missed, Job Done, Illegal Operation, End Zone, Block Missed, Incomplete Block, Mark Track Error and Data Request. Set up the DECTape command register according to bits 24–35 of the effective conditions *E* and perform the functions specified by bits 18–23 as shown (a 1 in a bit produces the indicated function, a 0 has no effect).

Note that the data and flag interrupt assignments are reversed from their usual positions. For most devices, bits 33–35 assign a channel for data, and bits 30–32 are used for flags, error conditions, etc.



*Notes.*

Bits 18–20 all 0 affects a transport newly selected by this CONO only to the extent that its state differs from the present state of the motion command bits in the control.

**CAUTION**

Spurious data processing can occur if the programmer inhibits the delay but the tape is actually moving in the direction opposite the one selected.

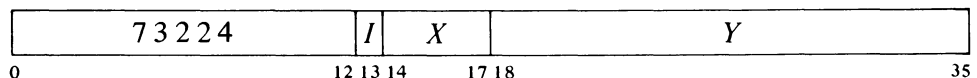
In a CONO that supplies control information for a transport already selected, bits 22–26 can all be 0.

- 18-20 Bits 19 and 20 both 1 changes the direction specified by the control and hence causes a tape already selected to turn around. If bit 18 is 1, bits 19 and 20 should both be 0. A transport already selected is unaffected if bits 18–20 are all 0.
- 21 If a tape is being newly selected by this CONO, begin processing it after only 1 ms provided the tape is up to speed. The program should inhibit the start delay only when selecting a tape that is already moving in the right direction.
- 22 Following the loading of the conditions supplied by this instruction, select the transport addressed by the unit register. If a transport is already selected and bit 23 is 0, bit 22 has no effect.
- 23 Clear the unit register and deselect the transport previously selected.
- 24-26 Numbers 1–7 address transports 1–7; 0 addresses transport 8. The contents of these bits are ored into the unit register. Hence to address a new transport the programmer must give the CONO with a 1 in bit 23; otherwise the control will address the transport whose number is the OR function of bits 24–26 and the previous number.
- 27-29 Perform the function specified by these bits on the selected tape.
 

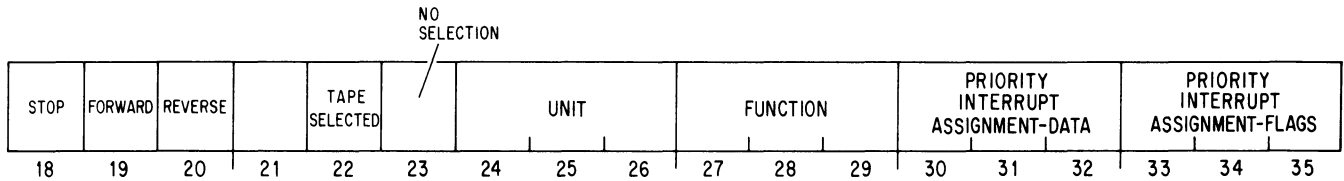
0	Do Nothing
1	Read All
2	Read BN
3	Read Data
4	Write Mark Track
5	Write All
6	Write BN
7	Write Data

Any write function sets the Data Request flag, requesting an interrupt on the channel assigned by bits 30–32.

**CONI DTC, Conditions In, DECTape Command**



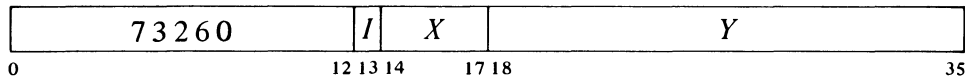
Read the contents of the DECTape command register into the right half of location *E* as shown.



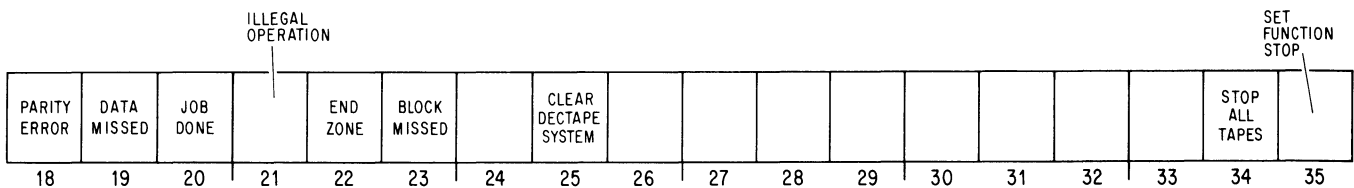
*Notes.*

- 18 The currently selected tape is not in motion or is stopping.
- 19 The currently selected tape is moving forward.
- 20 The currently selected tape is moving reverse.
- 22 The transport addressed by bits 24–26 is currently selected.
- 23 No transport is currently selected.

**CONO DTS, Conditions Out, DECTape Status**



Enable or disable flags as specified by bits 18–23 of the effective conditions *E* to interrupt on the channel assigned to the flags (a 1 in a bit enables the given flag, a 0 disables the flag) and perform the functions specified by bits 25, 34 and 35 as shown (a 1 produces the indicated function, a 0 has no effect).



*Notes.*

- 34 All transports will stop, but the selected transport (if any) will then resume the motion currently specified for it.
- 35 Setting Function Stop clears Data Request and inhibits all further data requests; it then terminates the function currently being executed, setting Job Done, as follows:

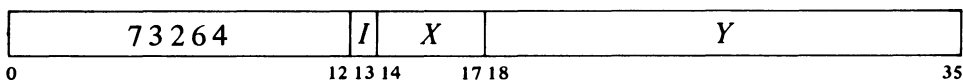
If the function is processing a block number or synchronizing area, it terminates at the end of the current word (hence Job Done sets within half a millisecond).

Note that terminating a function has no effect on tape motion.

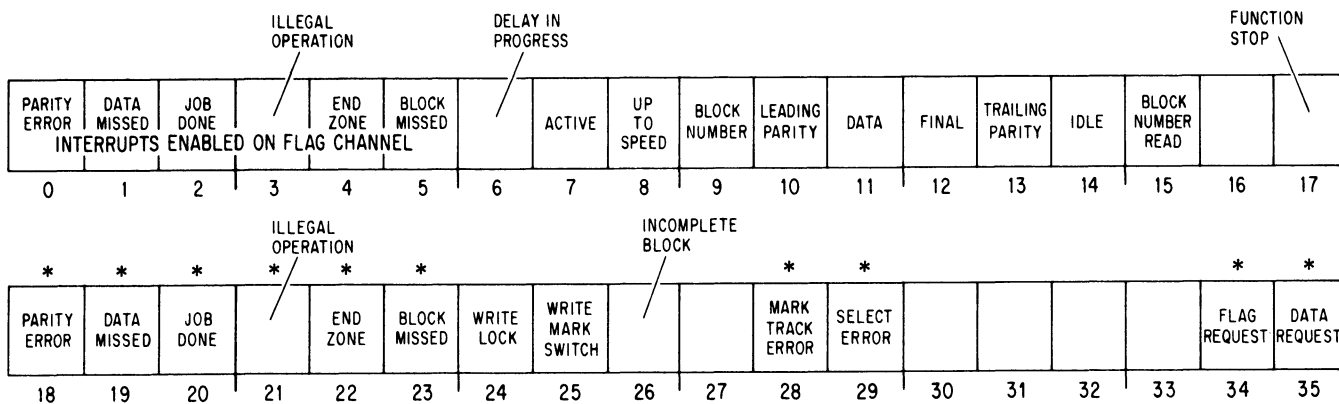
If the function is processing a data or parity area, it terminates following the trailing parity. However, there are no data requests. Therefore, during a read operation, the function simply reads and discards the rest of the data and checks parity; and during a write operation, the function writes the last word sent by the program over and over in the rest of the data area and writes parity.

In any other situation (a BN function spacing between block numbers; Write Mark Track) the function terminates immediately.

**CONI DTS, Conditions In, DECTape Status**



Read the status of the DECTape into location *E* as shown.



\*These bits cause interrupts.

*Notes.*

Note that this device has a full word of status but CONSZ and CONSO can test only the right half. To test the left half, the program must give a CONI DTS,AC and then use a test instruction [§2.8].

If the flags are enabled, the setting of Parity Error, Data Missed, Job Done, Illegal Operation, End Zone or Block Missed sets Flag Request, which in turn requests an interrupt on the flag channel (assigned by bits 33–35 of the last CONO DTC,). The setting of Mark Track Error or Select Error can request an interrupt as these flags in turn set Parity Error and Illegal Operation respectively. The setting of Data Request requests an interrupt on the data channel (assigned by bits 30–32 of the last CONO DTC,).

All status bits apply only to the tape currently selected.

- 6 The control is currently waiting through a selection, start or turn-around delay and is neither communicating with nor responding to the tape.
- 7 The control is now processing the tape or writing the mark track.



- 8 Bit 6 is 0 and the tape is moving at the correct speed for processing.
- 9 The control is now detecting a leading block number.
- 10 The control is now detecting a leading parity area.
- 11 The control is now detecting a data area except for the final half word.
- 12 The control is now detecting the final half word in a data area.
- 13 The control is now detecting a trailing parity area.
- 14 The control is now detecting the first half of a leading synchronizing area, a trailing synchronizing area, or a trailing block number.
- 15 A BN function is processing a block number or has just processed one and has not yet reached the leading parity area.
- 17 Data requests have ceased and the present function has or is being terminated either because data has been missed (bit 19 is 1) or the program has stopped the function [*see CONO DTS, above*]. Function Stop remains set until a CONO DTC, is given.
- 18 The control has read a data area with incorrect parity, or bit 28 is 1.
- 19 The program has failed to respond in time to a data request. The setting of this bit sets Function Stop (bit 17), terminating the function.
- 20 Function Stop (bit 17) has been set (either by the program or Data Missed being set) and the function has been terminated as described under bit 35 of CONO DTS,.
- 21 Any necessary initial delay has been completed (bit 6 is 0), and the command given by the program conflicts with a switch setting as follows:
- The function is Write Mark Track and the WRTM switch on the control panel is off (bit 25 is 0).
- The WRTM switch is on (bit 25 is 1) and the function is not Write Mark Track.
- The program has given a write function and the write switch on the selected transport is in the write lock position (bit 24 is 1).
- There is no transport in remote whose thumbwheel is dialed to the unit number selected by the program, or there is more than one such transport set to that number (bit 29 is 1).
- Note that the setting of this bit does not terminate the function, but the presence of a switch error does prevent the control from writing. Hence a read function will be executed although the control may be reading from two tapes at once or no tape at all.
- 22 The end zone has been encountered and the tape has been stopped.

Bits 9–14 indicate tape position; the control may be processing the zone in question or simply spacing over it.

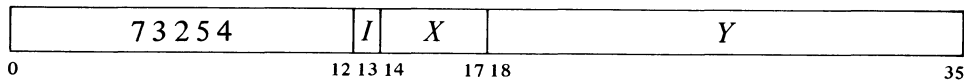
In other words, bit 15 goes on with bit 9 and goes off when bit 10 goes on.

Although the program transfers full words, transfers to and from the buffer within the control are in half words. Hence after Data Request sets, the program has half the current word transfer time in which to respond.



**DATAO DTS, Data Out, DECTape Maintenance**

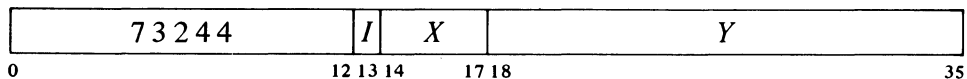
This instruction is for maintenance only.



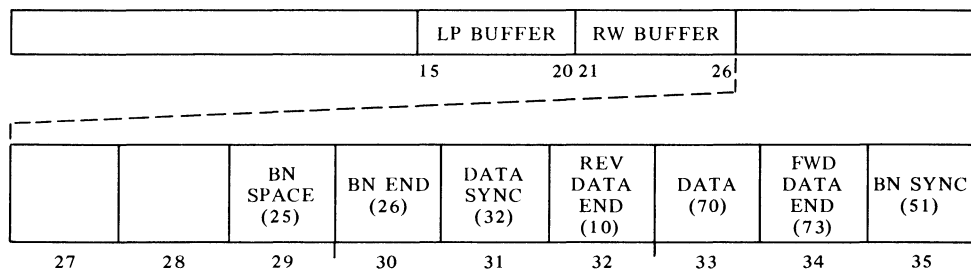
At DATAO CLR time trigger TP0, and if bit 21 of location *E* is 1, trigger the up-to-speed delay (TUPS0). At DATAO SET time trigger TP1 and load bit 35 of location *E* into the mark track shift register.

**DATAI DTS, Data In, DECTape Maintenance**

This instruction is for maintenance only.



Read the LP and RW buffers into bits 15–26 of location *E* and read the output of the mark track decoder into bits 29–35 as shown, where the current mark (if any) is indicated by a 0 in the listed bit.



**6.4 NORMAL PROGRAMMING**

Normal DECTape operation consists of reading block numbers until a desired block is found, reading or writing the desired number of words in the data area of that block or a number of contiguous blocks, and then waiting for Job Done. An interrupt program should first give a CONO DTS,0 to disable all flag interrupts. A CONO DTC, should then be given that both selects and deselects and that specifies the transport, the direction, the flag and data priority interrupt assignments, and the function Read BN; in other words, CONO DTC,*M3U2DF*, where *M* is the motion information, *U* is the unit number, and *D* and *F* are the data and flag PI assignments. Unless the programmer already knows the approximate position of the tape, it is best to search initially in the same direction that he intends to process the data. After starting the search an interrupt program should give a CONO DTS,660000 to enable all interrupts except Job Done and Block Missed, and wait for an interrupt produced either by an error or the data request for the first block number. When the transport gets up to speed and a block number is read, Data

Request is set requesting an interrupt on the channel assigned to it. The program must bring in the block number and compare it with the number of the desired block. If the desired block is moving toward the tape head, the program can simply wait until the correct number is read. If the block is moving in the opposite direction, the program gives a CONO DTC,3002DF to turn the tape around and read block numbers in the opposite direction. Once the correct number is found, if the tape is moving in the direction in which the programmer wishes to process the data, he can simply change the function to Read Data or Write Data (the DTC conditions would be 0003DF and 0007DF respectively). If the tape is traveling in the wrong direction, the program should turn it around and continue to read block numbers until the desired number is encountered in the right direction.

Along with starting the data function, the program should also give a CONO DTS,770000 to enable all of the flags to interrupt. While performing the Data function, the control sets Data Request every time it takes the second half word from the buffer in writing or fills the buffer in reading. The program must respond with a data IO instruction to device DTC, which is most easily issued by a block IO contained in the interrupt channel location. In Read Data, the control checks parity at the end of each data area and sets Parity Error if it is incorrect. Data transfers cease while the control is processing parity areas and spacing across the synchronizing areas and block numbers. However, the control continues to process one data area after another until Function Stop sets either because the program gives a CONO DTS,1 or Data Missed is set because the program does not respond in time to a data request. If Function Stop is set within a data area, Incomplete Block sets; and although data transfers cease (Data Request can no longer be set), the function continues as previously described until the control has processed the trailing parity, at which time Job Done sets to request an interrupt on the flag channel. The program should leave the tape selected until Job Done sets so that for Write Data the control will write the correct parity, and for Read Data it will set Parity Error if the parity in the final block is incorrect.

Once Job Done sets, the tape remains selected unless deselected by the program, but the control does not act on the tape in any way except to stop it if it runs into the end zone. The program must give a new CONO DTC, to stop the tape, to deselect the tape, or to execute another function.

To use the DECTape, without the interrupt, the program follows essentially the same scheme as given above: the program need not enable the flags, but it must test them to determine when data requests are made and whether anything is amiss.

**Timing.** As described in §6.2, start time for a transport is about 150 ms, turnaround time is 350 ms. If the program inhibits the start delay when reselecting a tape that is already moving in the right direction, tape processing can begin after 1 ms. A transport requires about 150 ms to stop the tape.

The timing of operations in the execution of a function by the control depends upon the direction of tape motion. If the tape is moving in the same

direction that the mark track was recorded, all times are essentially constant. Following a data request for a block number, the program has 400  $\mu$ s to switch to a Data function before the leading parity is encountered and Block Number Read clears. A Data function given after this time sets Block Missed requesting an interrupt on the flag channel, and the program should turn around and start the block over before attempting to process the data area. Once a Data function has begun, the program has 200  $\mu$ s following each data request to take care of the transfer before Data Missed sets; this flag not only requests an interrupt on the flag channel, but sets Function Stop to inhibit further data requests and terminate the function. The time between blocks is about 2 ms.

For processing in the opposite direction the mark track was recorded, times vary up to 20 percent from the times just given. Hence in any circumstances the program has at least 320  $\mu$ s to switch to a Data function following the identification of a block, and it has at least 160  $\mu$ s to respond to each data request during the Data function.

**EXAMPLE.** This routine finds the block whose number is contained in location NUMBER and approaches it for processing with the tape moving forward. The interrupt is not used, and an error flag causes a jump to location ERRS. In general the error routine should stop the tape (CONO DTC,400000) and take some corrective action.

	CONO	DTC,231200	;Forward, unit 1, Read BN
A:	CONSZ	DTS,640000	;Error?
	JRST	ERRS	;Yes, do something
	CONSZ	DTS,20000	;End zone?
B:	CONO	DTC,300200	;Turn around, Read BN
	CONSO	DTS,1	;Got BN?
	JRST	A	;No, retest all
	DATAI	DTC,2	;Bring BN into AC2
	SUB	2,NUMBER	;Compute this BN – desired BN
	CONSZ	DTC,100000	;Going reverse?
	TLCA	2,400000	;Yes, complement condition for ;turnaround and skip the found test
	JUMPE	2,FOUND	;At desired block going forward?
	JUMPGE	2,B	;No, is turnaround required?
	JRST	A	;No, start testing flags again

**Programming Suggestions and Cautions.** To turn the tape around as quickly as possible, *eg* if the tape is several blocks away from the desired block and is traveling away from it, give a CONO DTC,33U2DF. This deselects and selects the transport and thus triggers only the start delay. Note that the unit number must be given because deselection clears the unit register.

To process block 0 forward (or block 1101 reverse), simply go to the appropriate end zone and then (as already discussed in §6.2) give two CONO

DTC,s wherein the first makes the tape go in the same direction and the second turns it around. The function given by the second CONO cannot be Read BN because this function will miss the first block number. Thus the program should give a Data function, which will process the first data area encountered. Since no block number is read before the Data function, Block Missed will be set. The program can ensure proper processing of the first block by making sure that the tape moves far enough into the end zone (by the procedure already indicated).

Whenever a moving tape is temporarily not in use, the program should watch it or stop it so that the tape cannot run off the reel. Halting the processor does not affect the tapes, but IO reset generated by the program or from the console stops all tapes.

**Other Functions.** Do Nothing is used only to start, stop or otherwise control a tape without performing any function. The remaining functions are used primarily for generation and maintenance of the prewritten block format [§6.5], but with Read All and Write All the entire tape except for the parity areas can be used for data storage. When the program gives an All function, the control starts processing the tape at the first trailing block number encountered and continues processing every area of the tape until Function Stop is set. Data transfer timing is the same as for Data functions, but there is no space between blocks. The program has 200  $\mu$ s to respond to a data request before Data Missed sets if processing is in the same direction the mark track was written, otherwise the program has at least 160  $\mu$ s. The data transfer rate is a constant 400  $\mu$ s per word in the former direction, and averages 400  $\mu$ s over the entire tape in the latter direction; however, the time between transfers is increased by half when the control processes a parity area.

### Readin Mode

The only requirements (beyond those given in §2.12) for readin mode with DECTape are that the data must be in consecutive data areas beginning with block 0 and the tape must be mounted on transport 8. To select the DECTape with the readin device switches, turn on the second, third and fifth switches from the left (320, the DTC device code).

Pressing the readin key causes the processor to place the DECTape control in a special readin sequence in which it selects transport 8 (unit number 0) and places it in reverse motion. When the end zone is encountered, the automatic stop does not occur; instead the control turns the tape around and executes a Read Data function beginning at block 0. After retrieving the specified number of words from the buffer, the processor simply fails to respond further. If the program read in does not stop the tape, it will stop at the forward end zone.

Write BN exists primarily because of symmetry in the function bits and is unlikely to be used. It writes one word in the first leading BN position encountered.

## 6.5 FORMATTING A TAPE

Write Mark Track and Write All are used to write the timing and mark tracks and write the block numbers on the tape. Read All can be used to check that the tape is formatted properly.

Listed here, in half words, is the information the program must send in *full* words to write the various parts of the mark track. For either direction the *full* words are the same but are sent in the opposite order.

FORWARD	404404	END ZONE	
↓	040404	} BLOCK NUMBER	} EACH BASIC BLOCK (2-WORD DATA AREA)
↓	040440		
↓	044040		
	004000	} PARITY	
	004000		
	004000		
	* - - - - -	} DATA AREA	
	444044		
	444044		
	444044		
	444044	} PARITY	
	444044		
↑	404004	} BLOCK NUMBER	
↑	400404		
↑	040404		
REVERSE	040040	END ZONE	
	* 444000	} INSERT FOR EACH ADDITIONAL WORD IN THE DATA AREA	
	444000		

While the control is writing the mark track, it writes the data sent by the program in the data tracks of all zones, including end zones and parity areas (which therefore do not contain parity information).

After the mark track is written, Write All is used to write the block numbers. Forward block numbers are written straight, but reverse numbers must be written with the octal digits complemented and in the reverse order. *Eg* the word sent for the forward BN for block 1101 is 000000 001101, but the word for the reverse BN in that block is 676677 777777. Between block numbers the program should send zeros to clear the data areas and write correct parity (writing the mark track always produces incorrect parity).

To format a tape use a transport whose head has zero skew so the tapes can be processed on any other unit. The usual procedure is to write the mark track forward and write the block numbers in reverse. Load a blank tape or one that is to be repaired (perhaps because it was disrupted by a strong magnetic field), and turn on the WRTM switch on the control. After the program starts the tape, writing can begin as soon as the tape is up to speed. To generate the standard format send 7000 end zone codes (3500 words) and then send the marks for 578 blocks, each with a data area of 128 words (*ie* send 444000 444000 in every block 126 times at the position marked by the asterisk in the above table). Then write end zone codes until the tape comes off the reel.

Now put the tape back on the left reel, turn off WRTM, and restart the program at a location for writing the block numbers. Run the tape out of the end zone, turn the tape around with a CONO that gives Write All, and send a throwaway word at the first data request. Then send the correct forward and reverse block numbers for block 1101 with 130 zero words between them. If the data is being written in some other block, that is alright – simply put it in every block until the end zone comes up again. This time the end zone will stop the tape. Turn the tape around with two CONOs [§6.2], the second CONO gives Write All, and send 000000 001101 at the first data request (for the first trailing, *ie* forward, BN). Then give the data in the order reverse BN, 130 zero words, forward BN for the remaining 577 blocks (1100 to 0).

The skew will certainly be within acceptable limits on any relatively new transport that has not been seriously mistreated.

It is not necessary to use the entire tape, but there must be at least two end zone marks at each end and enough tape outside the blocks to load it on the reels.

Remember, the control handles the parity areas.

## PART II

### STANDARD MAGNETIC TAPE

A system for handling industry-standard magnetic tape consists of a TM10 control and up to eight tape transports; each unit in the system occupies a separate cabinet. DEC supplies several types of transports that differ in tape speed and tape handling characteristics. Each type is available in two versions, for recording information in seven tracks and nine tracks. Thus data transfer rates and timing depend on the transport, but each transport supplies information to the control such that transports of different speeds and recording formats can be operated by a single control. Transports currently available move tape at speeds of 45, 75 and 150 inches per second. Every transport accommodates two 10½-inch reels (one for supply, one for takeup) and can record information in three densities: 200, 556 and 800 bytes per inch (bpi). A full reel has 2400 feet of half-inch tape and at 800 bpi can store over 135 million bits of data in the 7-track format, or over 180 million bits in the 9-track format.

The program communicates with the tape control, which in turn governs all tape transports but communicates with only one transport at a time. Reading and writing (recording) can occur only when tape is moving forward (from supply reel to takeup reel), but the control can space the tape (*ie* move it to a new position) in either direction. Although only one transport can be reading, writing or spacing at a time, rewinding the entire tape onto the supply reel at high speed requires only initiation by the control. The rewinding transport then proceeds automatically while the control can operate another transport.

Data transfers between tape and control are governed entirely by the control. Transfers between a TM10A control and memory are handled by the program over the IO bus, whereas the TM10B control is connected to a data channel for automatic transfer of data to and from memory, thus bypassing the central processor [§5.1].

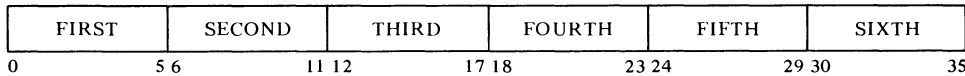
#### 6.6 TAPE FORMAT

The recording technique used is NRZI (nonreturn to zero, inverting). In a given frame (*ie* character position) a change in the direction of magnetization in any track represents a 1 in the character bit corresponding to that track. Thus if the same bit is 0 in a string of characters, there is no change in the track corresponding to that bit; but for a string of 1s, the flux direction changes in every frame.

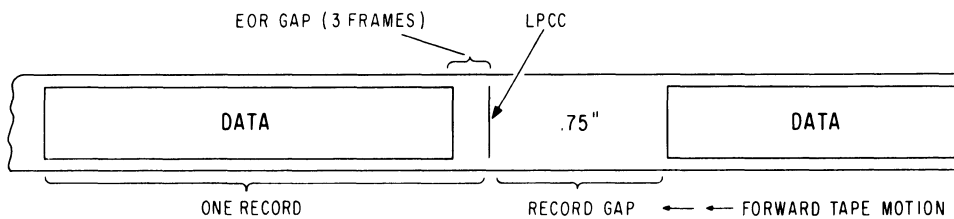
The control writes characters containing seven or nine bits of information; one bit is written in each track. Every character is part of a data record or a file mark. A data record contains both data characters and error-checking characters. Every data character consists of a data byte and a lateral parity bit, which the control generates so that the number of 1s in the character is odd or even as specified by the program. The data bytes in a record taken together correspond to a block of words sent from memory to the control. To separate adjacent records the control automatically erases a segment of tape between them; this segment is called a "record gap". The control always stops the tape in a gap.



Full words are transferred between memory and control even though the tape characters may contain 6-bit or 8-bit data bytes. To write, the control divides the words into data bytes, and when reading, the control reassembles the bytes into words. There are several ways in which this is done. For 7-track format, the program can select any density, and the control writes each word as six characters, each containing a 6-bit data byte. After the control writes the last data character for a record, it writes three blank frames (zero



7-TRACK BYTE DISTRIBUTION



7-TRACK RECORD FORMAT

characters) followed by a longitudinal parity check character (LPCC). The three blank frames constitute the end of record gap (EOR), which is used by the control to detect the end of record. The EOR is used in writing as well as reading since the tape encounters the write head first, and the control detects everything shortly after writing it. The LPCC (which may be zero) produces even parity in each of the tracks along the length of the record. The minimum record gap is .75 inch.

When the control reads or writes a data record, it checks that the (lateral) parity of every data character agrees with the parity specified by the program and checks that every track has even (longitudinal) parity.

The 9-track format is used for recording data compatible with systems based on 8-bit bytes. The program must select a density of 800 bpi, and the control writes bits 0–31 of each word in four characters, ignoring bits 32–35 altogether. The bits from left to right in each 8-bit byte are written in tracks 0–7. After writing the last data character, the control writes an EOR gap, a cyclic redundancy character (CRC), three more blank frames, and an LPCC. The control generates the CRC as described in §6 of USAS X3.22-1967, *USA Standard, Recorded Magnetic Tape for Information Interchange (800 CPI, NRZI)*. Taking the CRC bits as numbered in that document and the track scheme defined above, CRC bit 1 corresponds to the parity track and bits 2–9 correspond to tracks 0–7. The standard record gap is .5 inch minimum, .6 inch nominal, 25 feet maximum.

When the control reads a record in 9-track format, it assembles data into 36-bit words. Each word is composed of four data bytes in bits 0–31 and the corresponding character parity error indicators in bits 32–35 (eg a 1 in bit 33

For industry compatibility the program must select 800 bpi and odd parity.

The difference in nominal gap length between 7-track and 9-track format is due entirely to a difference in head spacing.

Characters are assembled into words in this manner by an IDPB loop or an ASCII or ASCIZ pseudoinstruction.

The "tracks" referred to here are simply a convenience for identifying the bits in the data bytes. The actual correspondence of character bits to physical tracks on tape is as follows:

BIT	2	0	4	P	5	6	7	1	3
TRACK	1	2	3	4	5	6	7	8	9

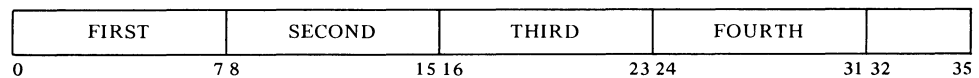
This scheme, which minimizes the effects of errors, is recommended in the standard referenced in the text.

A 1 in an error bit does not necessarily mean the corresponding byte is in error: the error could be in the parity track.

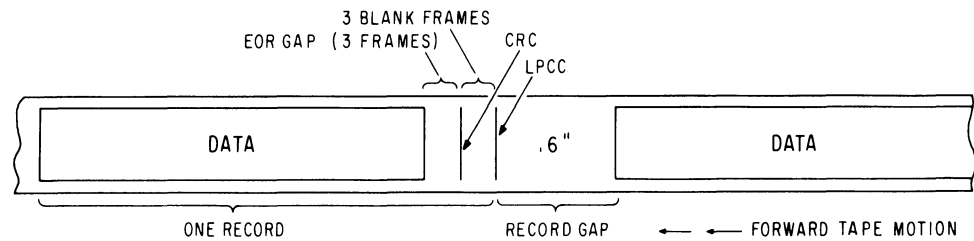
The program must use bit 32 and the parity being checked for in order to regenerate the bit actually read from the parity track of the CRC by the control, keeping in mind that the parity track does not contain an actual parity bit. The parity of the CRC will be odd if the number of data characters in the record is even, otherwise the CRC parity is even.

Errors discovered in a record in core dump format at 800 bpi can be corrected by re-reading the record in 9-track format, if investigation of the CRC indicates the errors are confined to a single track. The program must then reconstruct the original words, four from each group of five 4-byte sets supplied by the control.

Two or more contiguous missing characters would be interpreted by the control as an EOR gap. This sets the Bad Tape flag and terminates the function.



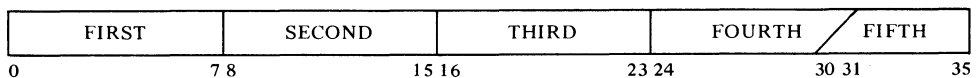
9-TRACK BYTE DISTRIBUTION



9-TRACK RECORD FORMAT

indicates an error in the character from which the byte in bits 8–15 was taken). If errors occur, the program can use the cyclic redundancy character to determine if the errors are confined to a single track, and if so, to correct them. After reading the data characters in a record, the control makes the CRC available in a word in which bits 0–7 contain information from the data tracks and bit 32 contains a parity error bit. The program can correct errors by using the procedure described in Appendix B of the standard. It is not necessary to reread the tape: the error pattern can be generated from the data in memory. If errors are confined to a single track, the program can correct the record by complementing the bit from the bad track in every byte whose error bit is 1.

To facilitate the use of 9-track tape for binary data applications, a core dump format is available in which the program can select any density, and the control uses the 9-track record format but writes full 36-bit words as five characters each. The first four bytes are taken from bits 0–31 of a word in the same manner as in 9-track format; the fifth data byte contains 0s in tracks



CORE DUMP BYTE DISTRIBUTION

0 and 1 and bits 30–35 of the word are contained in tracks 2–7. To reassemble the word during reading, the control ors the overlapping bits. The CRC is written in the usual fashion, but no error bits are supplied with the data bytes.

When writing in even parity in any recording format, the program must not supply a word containing a zero data byte, since this would result in a missing character (a blank frame), and no words beyond that point would be reassembled correctly. The control does not check for missing characters when reading, but such an event always terminates the function at the end of the current record.

To facilitate tape processing, the program can group sets of data records into files. The end of a file is indicated by a 3-inch gap followed by a file mark. The file mark is a special record containing a single, special data character and its LPCC, which is equivalent. The control always terminates a function when it encounters a file mark; in particular, the control can space by files as well as by numbers of records.

Each tape has two physical markers to indicate its extremities. These markers are reflective strips that are sensed by photoelectric cells in the transport (one marker can be seen on the tape in the illustration at the end of §H4.2). The loadpoint marker is located about fifteen feet in from the beginning of the reel and denotes the logical beginning of the tape. Reverse functions stop automatically at this marker. A load point gap of at least three inches (twenty-five feet maximum) precedes the first record on the tape. The endpoint marker is about twenty-five feet from the physical end of the tape; the final fifteen feet of tape should be left for trailer, *ie* the program should not record more than ten feet beyond the endpoint (this is enough for a 4000-word record at low density). A status bit indicates when the tape is beyond the endpoint, but this condition stops the tape automatically only when it is spacing forward.

An annular groove is molded into the back of every reel, and the control cannot write on the tape unless the supply reel has a plastic (write enable) ring in this groove. By leaving the ring out, the operator can protect the data on the tape from accidental destruction (overwriting or erasure).

While the control is actually processing the data portion of a record, the data transfer rate is fixed. However, in a lengthy tape run, the effective (average) transfer rate depends on record length, which determines the percentage of tape taken up by gaps (at the highest density in 7-track format, each record gap could hold 100 additional words). The effective transfer rate is therefore a function of record length as well as tape speed and density.

Tapes recorded on some IBM transports have a substandard loadpoint gap of only .5 inch and are thus not compatible with the TM10.

The markers are on the shiny side of the tape; the endpoint marker is against the edge nearer the transport, the loadpoint marker is against the opposite edge.

## 6.7 INSTRUCTIONS

Both versions of the tape control have a 36-bit buffer register BR. During writing, characters are transferred from the BR to the transport for storage. During reading, the characters are reassembled into words in the BR. In the TM10B, BR is connected directly to the channel bus; therefore, when the control is ready to receive or send data, the channel must respond within one character time. (This time requirement is quite reasonable since the data channel serves as a buffer between tape control and memory.) The TM10A has a 36-bit hold register HR, which provides data buffering between BR and the IO bus. Thus each time HR is free to receive a word from the bus or has a word for the bus, the program has one entire word time plus one character time in which to respond.

To run the tape, the program must select a transport and a function; the

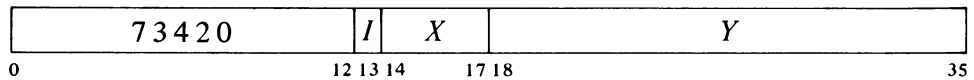
function requires specification of format information, such as parity and density. To use the interrupt, the program should also assign a channel for flags. The control makes data requests in functions that require the transfer of data to or from memory and in spacing a specified number of records. The TM10B uses the data channel for these functions; with the TM10A the program must either check the Data Request flag or assign a second interrupt channel for data. To write data, the control takes words from memory, divides them into characters and sends the characters to the selected transport. To read data, the control receives characters from the transport, assembles them into words and sends the words to memory. There is also a type of function, Read-Compare, which combines some of the characteristics of both Read and Write. During Read-Compare, the control actually reads the tape but does not send the words to memory; instead the control takes words from memory and compares them with the words assembled from tape, noting any discrepancy by means of a status bit. To space less than a file, the control makes data requests to allow the program or the word counter in the data channel to count the records spaced.

The tape control has two device codes, 340 and 344, mnemonics TMC and TMS. Device code 340 is used for transmitting data (with a TM10A) and command information; device code 344 is used for sending the initial data channel control word address to a TM10B and reading status. A second tape system would have device codes 350 and 354.

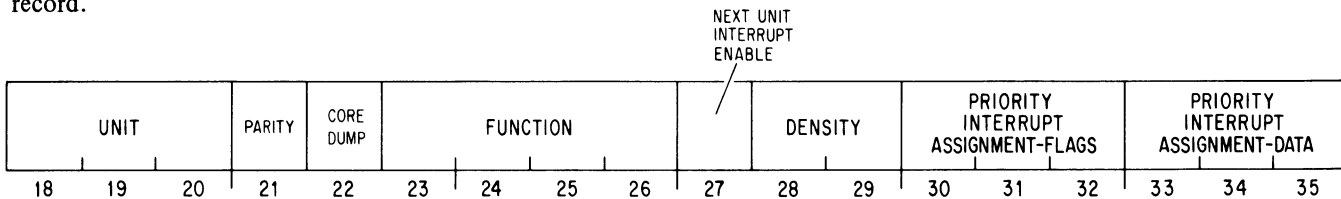
#### CAUTION

This instruction should not be given when the control is processing a tape (*ie* once given it should not be given again until Job Done sets), since it disconnects the control from the channel and stops the tape even if it is in the middle of a record.

#### CONO TMC, Conditions Out, Magnetic Tape Command



Clear BR, the Write EOR flag, and most status conditions other than those generated by the selected transport [*refer to CONI TMS,*]. Set up the tape command register according to bits 18–35 of the effective conditions *E* as shown.



18–20 Numbers 0–7 address transports 0–7 (the unit number is loaded into the next unit register).

21 1 selects odd parity, 0 selects even parity.

22 When bits 18–20 address a 9-track transport, a 1 in this bit causes the control to process tape in core dump format, and a 0 causes tape to be processed in 9-track format.

Bit 22 is ignored when bits 18–20 address a 7-track transport.

23–26 Perform the function specified by these bits on the addressed tape.

00	No-op	
10	Interrupt When Unit Ready	
01	Rewind	
11	Rewind and Unload	
02	Read Record	}
12	Read Multirecord	
03	Read-Compare Record	
13	Read-Compare Multirecord	
04	Write	}
14	Erase and Write	
05	Mark End of File	
15	Erase	
06	Space Records Forward	}
16	Space File Forward	
07	Space Records Reverse	
17	Space File Reverse	

Codes 11 and 14 actually specify double functions: for 11 the selected transport first executes Rewind and then Unload; for 14 the control first writes blank tape and then writes data.

All of the above are tape-moving functions except No-op and Interrupt When Unit Ready. The control actually reads tape (*ie* assembles characters into words in BR) only in Read and Read-Compare, but the control detects the information encountered on tape in all tape-moving functions except Rewind.

27 A 1 in this bit enables the 1 state of the Load Next Unit flag to request an interrupt on the flag channel (assigned by bits 30–32).

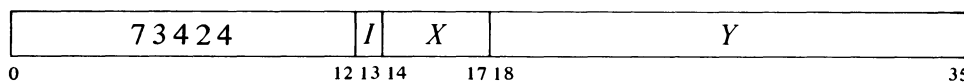
28–29 Process the addressed tape at the density specified by these bits.

00	200 bpi
01	556 bpi
10	800 bpi
11	

For all transports available at this writing, 11 selects 800 bpi. Should a new density be added, 11 will be used to select it.

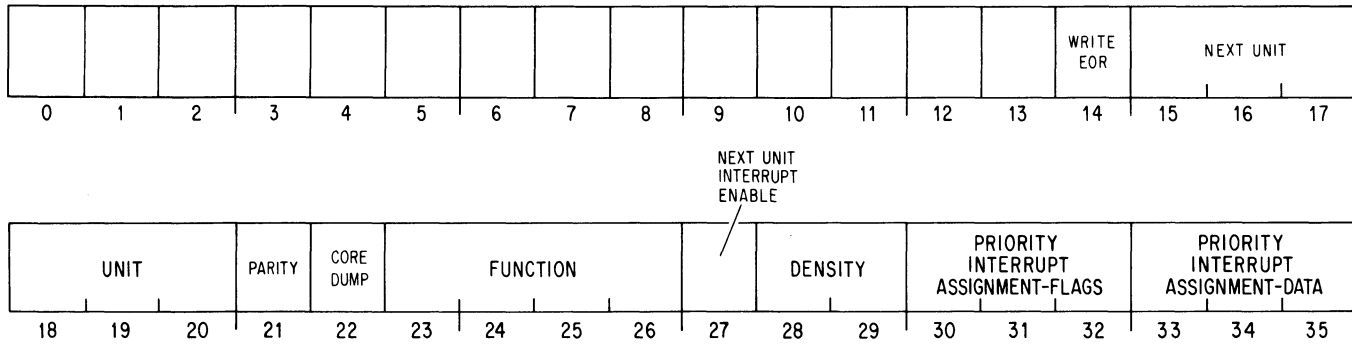
33–35 The TM10B ignores these bits.

**CONI TMC, Conditions In, Magnetic Tape Command**



Read the contents of the tape command register into bits 14–35 of location *E* as shown here. Bits 21–35 are simply the corresponding output conditions supplied by the previous CONO TMC, with the exception that in the TM10B, bits 33–35 are all 1s.

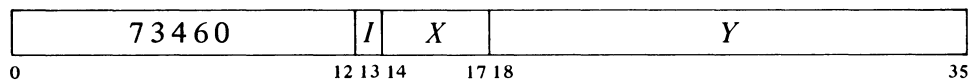
Note that CONSZ and CONSO can test only bits 18–35. To test bits 14–17 the program must give a CONI TMC,AC and then use a test instruction [§2.8].



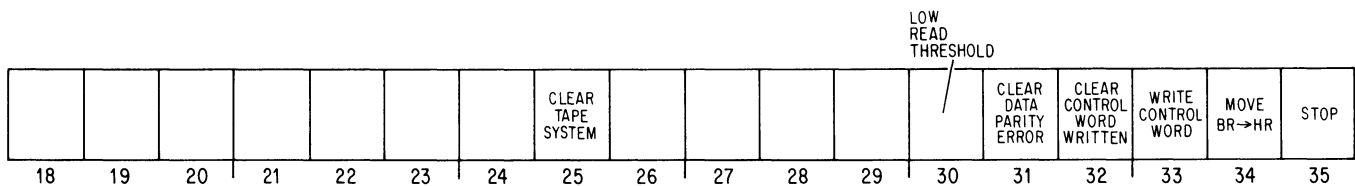
Bit 14 is for maintenance only. It is 1 whenever the control is terminating a record (*ie* writing blank lines and error checking characters). Bit 14 is also 1 for about a tenth of a millisecond after the control has encountered a file mark while spacing reverse.

If a CONO TMC, is given when the control is *not* moving tape (*ie* when the Load Next Unit flag is 1), the unit number specified by bits 18–20 of the CONO appears in both bits 15–17 and bits 18–20 of the CONI. If the CONO is given when Load Next Unit is 0 (this must be after Job Done sets, *ie* after the control has finished processing a record and is positioning the heads in the record gap), CONI bits 15–17 contain the unit number supplied by the CONO, but bits 18–20 continue to address the moving tape until Load Next Unit sets, at which time the number of the next unit is loaded into bits 18–20. Every CONO *addresses* a transport, but a transport does not become *selected* until its number appears in CONI bits 18–20 and the transport supplies status information to the control.

**CONO TMS, Conditions Out, Magnetic Tape Status**



Perform the functions specified by bits 25 and 31–35 of the effective conditions *E* as shown (a 1 in a bit produces the indicated function, a 0 has no effect), and specify the read threshold for the transport according to the state of bit 30.



Bits 31–33 are used only with the data channel [ §5.1 ] and have no effect on the TM10A.

- 30 A 1 causes the selected transport to accept a lower than standard signal level from the tape as representing a 1; a 0 returns the transport read detection circuits to operation at the standard read threshold. The program should read with Low Read Threshold set only after repeated attempts to recover a record correctly at the high threshold have failed.

34 Move the contents of BR to HR and clear BR.

When the control finishes reading the data in a 9-track record, the CRC is loaded into BR (and if the function is terminating, Job Done sets shortly thereafter). With a TM10B the program can then give a DATAI TMC, to transfer the contents of BR to location *E*. However, with a TM10A the program must first give a CONO TMS,2 to move BR to HR, and then give a DATAI TMC, to transfer HR to memory.

35 Stop tape at the end of the current record. Giving a 1 in this bit during a function inhibits further data requests, thus halting data transfers immediately. In Write the control terminates the record (and the function) after writing any data already sent. Read and space functions terminate at the end of the current record, and the TM10B continues to hold the data channel until termination except in Read-Compare (and of course in file spacing functions, which do not use the data channel).

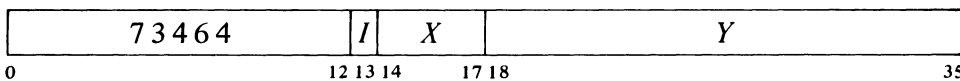
When the tape is stopped at the end of a record (or a file mark), the tape is positioned so that the heads are at the proper place in the record gap. If the program gives a CONO TMS,1 after the control processes the last word in a record but before the time at which the control would send the stop signal to the transport, the transport stops anyway. (Load Next Unit sets when the stop signal is sent.) After the last word in the record, the time interval within which the program can stop the tape in the current gap is approximately equal to the stop time listed for the transport plus about forty 800-bpi character times. A CONO TMS,1 given after the proper stopping position stops the tape at the end of the next record.

Note that termination of a function halts data transfers and releases the data channel, but a function never terminates until the end of a record even when transfers cease within the record.

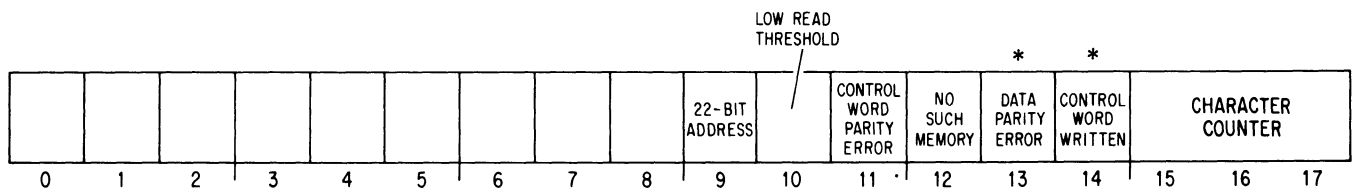
This bit is used primarily with the TM10A, but it may be used with the TM10B to stop a tape operation independently of the data channel control word list.

This applies only to reading and spacing, since the control cannot write more than one record at a time.

**CONI TMS, Conditions In, Magnetic Tape Status**



Read the status of the tape system into bits 9–35 of location *E* as shown.



*	REWINDING	*									*					*	*
UNIT HUNG		LOAD POINT	ILLEGAL	TAPE PARITY ERROR	END OF FILE	END POINT	READ- COMPARE ERROR	RECORD LENGTH DIFFERS	DATA LATE	BAD TAPE	JOB DONE	UNIT IDLE	CHANNEL FLAG	WRITE LOCK	7 TRACK	LOAD NEXT UNIT	DATA REQUEST
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

10-0649

\*These bits cause interrupts. Note that Bad Tape should *not* interrupt. Logic for an interrupt was added by ECO TM10-00014 but was later removed by ECO TM10-00022.

Of course Low Read Threshold is always cleared by a CONO TMC, given as an emergency shutdown while the control is actually processing a tape.

Bits 15–17 are primarily for maintenance: the character counter controls the division and assembly of data words.

CONO TMC, clears all of these status bits except 22-Bit Address, Load Next Unit, and those generated by the selected transport, namely Rewinding, Loadpoint, Endpoint, Unit Idle, Write Lock, and 7 Track. Low Read Threshold is cleared, however, only by an initial CONO TMC, *ie* one given to begin tape operations. Hence once Low Read Threshold is set by the program, then so long as the control has a tape in motion, a CONO TMC, has no effect on this flag and it is left to the program to determine its state.

Interrupts are requested on the flag channel (assigned by bits 30–32 of the last CONO TMC,) by the setting of Data Parity Error, Control Word Written, Unit Hung, Illegal, Job Done, and if enabled, Load Next Unit. In the TM10A the setting of Data Request requests an interrupt on the data PI channel (assigned by bits 33–35 of the last CONO TMC,).

Bits 9, 11–14 and 31 reflect conditions in the data channel and are always read as 1s (but cause no interrupts) in the TM10A. When the data channel sets Control Word Parity Error or No Such Memory, it simultaneously terminates operation; data transfers cease immediately and the tape control terminates the function at the end of the current record.

10 The transport will accept a lower than standard signal level from the tape as representing a 1.

18 An addressed transport has failed to respond within one second to a tape-moving function (because it is either off line or rewinding) or a selected transport has gone off line during a function. The setting of this bit requests an interrupt on the flag channel.

When a transport hangs up, the control does not terminate the function – it simply fails to continue, *ie* the control effectively hangs up too.

19 The selected tape is rewinding.

20 The selected tape is at loadpoint.

21 A transport has been selected but the tape control cannot place it in operation because of one of the following illegal conditions:

The program has given a read, write or space function to process in 9-track format but has specified a density other than 800 bpi.

The function requires reverse spacing and Loadpoint (bit 20) is 1.

The function requires writing and Write Lock (bit 32) is 1.

The setting of Illegal prevents the control from executing the function and requests an interrupt on the flag channel.



- 22 In Read, Write or Read-Compare, the control encountered a data character whose parity differed from that specified with the function or discovered a track with odd parity, and therefore the function terminated at the end of the record in which the error occurred.
- 23 The control has encountered a file mark on the tape.
- 24 The selected tape is beyond the endpoint.
- 25 In Read-Compare a word read from tape was not identical to the corresponding word supplied from memory. When this bit sets, data transfers cease, the data channel is released, and the function terminates at the end of the current record.
- 26 In Read or Read-Compare the program misjudged the number of words in the record or records being read. The program's "judgement" is reflected in the word count in the TM10B or in the timing of the CONO TMS,1 in the TM10A. Of course "misjudgement" exists on the assumption that the program meant to read an entire record.
- When all the records in a file are read, this bit gets set only if the number of words is underestimated; an overestimate simply allows the control to read the file mark, which terminates the function without setting Record Length Differs.
- When this flag sets, data transfers cease and the function terminates at the end of the current record; in Read-Compare the data channel is released immediately, in Read the data channel is held until the function terminates.
- 27 In Read, Write or Read-Compare, the program (TM10A) or the data channel (TM10B) failed to respond in time to a data request. In Write, the control writes a zero word and then terminates the record; in a read function, data transfers cease immediately and the function terminates at the end of the current record.
- 28 The control has encountered either data in a record gap or a false end of record (two or more contiguous blank characters inside the record). When this bit sets, the function terminates.
- 29 The control has completed a function and is ready for the program to give a new function. The setting of this bit requests an interrupt on the flag channel.
- 30 The currently addressed transport is selected and is not now in operation.
- 31 Bit 11, 12, 13 or 14 is 1.
- 32 The supply reel of the selected transport does not have a write enable ring inserted.
- 33 The selected transport handles 7-track tape (a 0 indicates 9-track tape or no transport selected).

*CAUTION*

Bit 22 is usually set and bit 23 may be set during a record, but neither is valid until Job Done sets (bit 29).

NOTE: This explanation is valid only for a function that goes to completion in a normal manner without errors. Any condition that causes data transfers to cease prematurely (eg the setting of No Such Memory or Data Late) also sets Record Length Differs. Bit 26 is also set in a TM10A if the program fails to give the

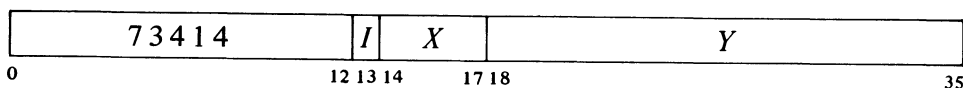
CONO TMS,1 in time at the end of a single-record function.

If the control is reading 7-track tape, and the program fails to retrieve the last two words (or the channel fails to retrieve the last word) in a record by the time the record gap is encountered, the function terminates without setting Data Late.

Note that writing a zero word in even parity produces a long EOR gap, which can cause problems when the tape is read.

- 34 The control is not moving tape and a CONO TMC, given now will select a transport. The setting of this bit requests an interrupt on the flag channel if Next Unit Interrupt Enable is 1 (CONI TMC, bit 27), and loads the next unit address to select a new unit (CONI TMC, bits 15–17 and 18–20 respectively) if the program has given a CONO TMC, since Job Done set.
- 35 The control is ready for a data transfer. In the TM10A the setting of this bit requests an interrupt on the data PI channel.

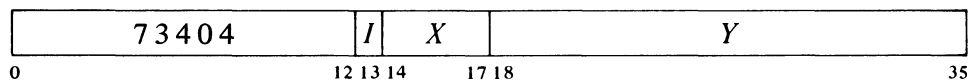
**DATAO TMC, Data Out, Magnetic Tape Control**



In the TM10B this instruction simply clears Data Request if the function is not Read.

*TM10A:* Load the contents of location *E* into HR, and if the function is not Read, clear Data Request.

**DATAI TMC, Data In, Magnetic Tape Control**

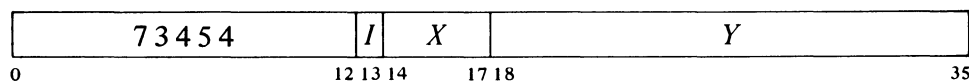


This instruction is used to read the CRC and is simulated by the hardware for readin mode.

*TM10A:* Transfer the contents of HR into location *E*, and if the function is Read, clear Data Request. If Job Done is 1, clear BR.

*TM10B:* Transfer the contents of BR into location *E*, and if the function is Read, clear Data Request. If Job Done is 1, clear BR.

**DATAO TMS, Data Out, Magnetic Tape Status**



In the TM10A this instruction is for maintenance: it moves HR to BR, the inverse of CONO TMS,2.

*TM10B:* Load the contents of bits 27–34 of location *E* into the initial data channel control word address register in the tape control. (If bit 35 of location *E* is 1, select even memory parity for tape input (read) operations. This condition is for maintenance only; in all ordinary circumstances bit 35 must be 0 so the data channel generates odd parity for words stored in memory.)

Once this instruction is given following power turnon, all tape operations use the same initial address until the program supplies a new one by giving another DATAO TMS,.

## 6.8 TAPE FUNCTIONS

Before starting any tape operation that uses the data channel, the program must give a DATAO TMS, unless the new function is to use the same initial control word address that was used for the previous function. When giving a function the program must give the unit address and must also supply the other initial conditions required for the function. Once the addressed transport has responded, the control tests the function to determine whether it is legal. If the initial conditions for any tape-moving function other than Rewind address a 9-track transport without specifying 800 bpi density or setting Core Dump, the control sets Illegal and shuts down (other tests depend upon specific functions as described below). If the function is legal, the control then begins its execution provided the control is a TM10A or the function does not require the data channel; for a function that requires the data channel in a TM10B, the control waits until the channel is connected before starting the function. The timing in the various functions is dependent upon the transport speed, tape handling characteristics and density, and is therefore treated in the discussion of each transport.

The program can choose the density and parity for writing. However, once data is written, it must be read in the same density and parity or it will not be read correctly. (Space functions do not check parity, but an EOR gap can be missed if the wrong density is specified.) The program should also write a file mark in the same density as the data records in the file to allow a read or space function that is processing the file to detect the mark.

Terminating a function and setting Job Done are essentially equivalent: the control sets Job Done when and only when it terminates a function. Termination can occur only when the control has reached a gap on the tape or has encountered the loadpoint. Following termination at a gap the control always stops tape upon reaching the proper position unless the control is already set up to execute another function with the same tape moving in the same direction [§6.10].

When a flag interrupt occurs, the program should determine if the function is finished by checking Job Done. If Job Done is clear, the other interrupt flags should be checked to determine what happened: Unit Hung, Illegal, and in the TM10B, Data Parity Error via Channel Flag (Control Word Written and Load Next Unit can be set only if the program had previously taken action that would allow them to be set). If Job Done is set the program should check error and other flags appropriate to the function. For a TM10B always check Channel Flag to determine if the function was aborted by Control Word Parity Error or No Such Memory. Other flags are listed at the end of each function.

**Interrupt When Unit Ready.** The control simply waits until the addressed transport becomes selected and indicates that it is ready for operation (Unit Idle is 1), at which time the control sets Job Done.

**Write.** The program selects the density and parity. If Write Lock is 1, the control sets Illegal and shuts down. Otherwise it continues as follows.

Of course, a function can be aborted at any time by giving a CONO TMC,.

This function is used primarily to wait for a transport to finish rewinding.

*TM10A*

The control starts the tape and sets Data Request. As soon as the program responds with a DATAO TMC, the control again sets Data Request in order to fill both HR and BR. Upon completing the record gap, the control writes the words it receives from the program, setting Data Request as each word is written. If at any time the program fails to supply a word in time for continuous writing, the control sets Data Late, writes a zero word, terminates the record, and sets Job Done. Otherwise, the control continues writing until the program gives a CONO TMS,1; the control then writes whatever data has already been received, terminates the record, and sets Job Done.

Flags: Tape Parity Error, Endpoint, Data Late, Bad Tape.

**Mark End of File.** The program should select the same density that was used when the data records in the file were written. If Write Lock is 1, the control sets Illegal and shuts down. Otherwise, the control erases at least three inches of tape (*ie* it extends the present record gap into a file gap), writes a file mark, and sets End of File and Job Done. Flags: Tape Parity Error, Endpoint, Bad Tape.

**Erase.** If Write Lock is 1, the control sets Illegal and shuts down. Otherwise, the control erases at least three inches of tape and sets Job Done. Flags: Endpoint, Bad Tape.

**Erase and Write.** The control executes Erase without setting Job Done and then executes Write.

**Read Record.** The program must select the same parity and density that were used when the data was written.

*TM10A*

The control reads one entire record, setting Data Request as each word is ready for the program. The program must respond before the next word is ready or Data Late sets. If the program gives a CONO TMS,1 or Data Late sets during the record,

*TM10B*

The control waits until the data channel connects, at which time it starts the tape and makes a data request to fill BR. Upon completing the record gap, the control writes the words it receives from the channel, making a data request as each word is written. If at any time the channel fails to supply a word in time for continuous writing, the control sets Data Late, writes a zero word, terminates the record, releases the channel, and sets Job Done. Otherwise, the control continues to write until the data channel terminates operation, at which time it writes any data already received, terminates the record, and sets Job Done.

The character used in a file mark is 017 on 7-track tape or 023 on 9-track tape. These have even parity regardless of the parity specified by the program, but a correct file mark cannot set Tape Parity Error.

Erase is used primarily to skip sections of tape on which the program has found it impossible to write data correctly, *ie* without parity errors or a bad tape indication.

The control will search for data indefinitely if it attempts to read a totally blank tape. Hence, any read program should include a time check and abort the function if nothing has been retrieved by the time the tape moves twenty-five feet.

*TM10B*

The control waits until the data channel connects and then reads one entire record and supplies the words to the channel. If the channel terminates or Data Late sets before the record is finished, there are no further transfers but the control con-

there are no further requests but the control continues to the end of record. If transfers cease before the last word is read or the program fails to give a CONO TMS,1 within two character times after the last word, Record Length Differs sets. When the control detects the end of record, it sets Job Done.

Flags: Tape Parity Error, End of File, Endpoint, Record Length Differs, Data Late, Bad Tape.

**Read Multirecord.** This function is similar to Read, but the control does not terminate the function automatically at the end of the first record. Instead, the control reads from one record to the next and terminates the function when it encounters a file mark or detects the end of any record in which the program gives a CONO TMS,1, the data channel terminates (TM10B), a parity error is discovered, Data Late or Bad Tape sets, or the number of characters detected in the record does not constitute an integral number of words. If any of these conditions except a parity error occurs before the last word in the record is read, data transfers cease, but the control continues to the end of record with the TM10B holding the data channel.

If the function terminates at a file mark, Record Length Differs cannot be set.

**Read-Compare Record.** The program must select the same parity and density that were used when the data was written.

#### *TM10A*

The control sets Data Request to get the first word from the program and then reads one entire record. As each word is read, the control compares it with the last word sent by the program and again sets Data Request if the two words are identical. If they are not identical, the control sets Read Compare Error and terminates data requests, but continues to the end of record. In all other respects this function is identical to Read Record.

continues to the end of record. If transfers cease before the last word is read or the channel does not terminate after the last word is read, Record Length Differs sets. When the control detects the end of record, it sets Job Done and releases the channel.

If a missing character or a file mark results in a partial last word, the control makes a data request for it when the end of record is detected. However, for the character in a file mark, Data Request stays set for only 1½ character times.

#### *TM10B*

The control waits until the data channel connects and then makes a data request for a word that is compared against zero. Hence, in order to execute this function properly, the program must set up the data channel control word list so that the channel supplies a zero throwaway word before supplying the words to be compared with those from tape.

The control then reads one entire record and compares each word read with the word supplied by the channel. If the two words are not identical, the control sets Read Compare Error and data transfers cease, but the control continues to the end of record. In all other respects this

function is equivalent to Read Record except that the setting of Record Length Differs by premature termination of data transfers releases the data channel.

Flags: Tape Parity Error, End of File, Endpoint, Read Compare Error, Record Length Differs, Data Late, Bad Tape.

**Read-Compare Multirecord.** This function is related to Read-Compare Record in exactly the same way as Read Multirecord is related to Read Record. The control terminates the function at the end of any record in which any of the conditions listed for Read Multirecord occur. Termination also occurs if Read-Compare Error is set. For the TM10B the program must set up the data channel control word list to supply a zero throwaway word at the beginning of every record.

**Space Records Forward.** The program must select the same density that was used when the tape was written.

#### TM10A

The control spaces the tape forward setting Data Request at the beginning of each record. If the program does not respond with a DATAO TMC, (with a throwaway data word) by the time the end of record is detected, the control sets Job Done. If the program responds in time, the control continues on to the next record. Hence, the number of records spaced over is one greater than the number of DATAOs supplied by the program.

To handle record spacing with the TM10A, use a BLKO where the count in the left half of the pointer is equal to the negative of the number of records to be spaced, and give a CONO TMS,1 when overflow occurs.

#### TM10B

The control waits until the data channel connects and then spaces the tape forward. At the end of each record the control makes a data request and the data channel responds by sending a throwaway word and incrementing its word counter. When the channel terminates — *ie* when the word counter overflows — the control sets Job Done. Hence, the control spaces a number of records equal to the word count given to the data channel, except that one record is spaced even if the word count is zero.

Besides terminating for a specific number of records as explained above, the control terminates Space Records Forward when it encounters a file mark or detects the end of any record in which Endpoint sets or the program gives a CONO TMS,1. Flags: End of File, Endpoint, Bad Tape.

**Space File Forward.** This program must select the same density that was used when the tape was written. The control spaces the tape forward until it encounters a file mark or detects the end of a record in which Endpoint sets or the program gives a CONO TMS,1. Flags: End of File, Endpoint, Bad Tape.

**Space Records Reverse.** The program must select the same density in which the tape was written. If Loadpoint is 1, the control sets Illegal and shuts down. Otherwise, the control executes a function that is equivalent to

In reverse spacing, a record can be detected only if it is a file mark or has at least three characters.

Space Records Forward except that the tape moves in reverse, Endpoint has no effect, and the control terminates the function if it encounters the loadpoint. Flags: Loadpoint, End of File, Bad Tape.

**Space File Reverse.** The program must select the same density in which the tape was written. If Loadpoint is 1, the control sets Illegal and shuts down. Otherwise, the control spaces the tape in reverse until it encounters a file mark or the loadpoint or detects the end of any record in which the program gives a CONO TMS,1. Flags: Loadpoint, End of File, Bad Tape.

**Rewind.** The control places the transport in operation and sets Job Done once the function is initiated. The control is then free for further use by the program while the transport rewinds the tape at high speed onto the supply reel and stops at loadpoint. Flags: Rewinding, Loadpoint.

**Rewind and Unload.** This function executes exactly like Rewind except that upon encountering the loadpoint, the transport pulls the entire tape off of the takeup reel, winds the tape onto the supply reel, and goes off line.

Some transports cannot be unloaded by the program.

## 6.9 PROGRAMMING CONSIDERATIONS

Before beginning tape operations, check the status of the control to determine if it is free, which is indicated by Load Next Unit being set. Then give a No-op for the desired transport to check if it is on line and ready: ready is indicated by Unit Idle being set. If the transport is not ready but Rewind is set, then the transport is on line but is rewinding. Before using the transport, make sure it is the right type — bit 33 is 1 for 7-track, 0 for 9-track; if the program is to write or erase tape, Write Lock must be 0.

Once the control is executing a function, the program must wait for Job Done to set before giving another function. After Job Done sets, the control continues to select the current transport while waiting for the tape to move the correct distance into the gap. Then the control signals the transport to stop tape and sets Load Next Unit; and at this time the control selects the next unit if the program has given a new function. The distance the tape moves into the gap for a given direction is the same regardless of whether it is a record gap, a file gap or simply an erased portion of tape. If the program gives a new function before Load Next Unit sets, and the function is for the same transport and direction as the previous one, the TM10A does not stop the tape at all. The TM10B always releases the data channel at the completion of a function; hence, it will keep the tape moving between functions only if it can reconnect to the channel, *ie* if the channel is still free and the control has priority. If some other device connects to the channel before the TM10B regains service, the tape stops.

In Rewind, Job Done and Load Next Unit are both set soon after the tape is up to speed, and the control is then free for operations with other transports. To wait for a transport to finish rewinding, give an Interrupt When Unit Ready, which sets Job Done when the tape reaches loadpoint and is

The most likely source of parity errors is dropping 1s. If repeated attempts to read a record fail, try it using the low read threshold: give a CONO TMS,40 and reread the record. But make sure to clear Low Read Threshold (by giving a CONO TMS,0) before going on to subsequent records, as the low threshold increases the possibility of reading 0s as 1s.

To space over a file mark with a TM10B, always use Space File Forward to avoid tying up the data channel.

ready to move forward again. Do not give a tape-moving function for a transport that is rewinding, as Unit Hung will set, requesting an interrupt, if the tape is not ready within one second. After any other function, the program can give any function for the same or another transport as soon as Job Done sets. When addressing a new unit or changing direction on the same one, the CONO TMC, can enable the setting of Load Next Unit to interrupt when the new unit is selected or the new function starts. If the program simply wishes to be able to check the status of a new transport at the earliest possible moment, it can give a No-op with Next Unit Interrupt Enable.

In Read Record or Read-Compare Record in a TM10A, Record Length Differs will set if the program does not give a CONO TMS,1 within two to three character times after Data Request sets for the last full word; note that this is considerably less time than the program has for retrieving words without data loss. If missing characters produce an incomplete final word and the program has not deliberately stopped transfers prematurely by giving a CONO TMS,1 before the record is complete, the end of record sets both Data Request and Record Length Differs. To read an entire file, it is best to read the data records and then space over the file mark. Of course if the length of the file is unknown, then simply dispense with the CONO TMS,1 altogether (or give a large word count in the TM10B), but remember that the final word transferred will contain only the file mark character. Timing for this is standard in the TM10B, but in the TM10A the program has only about 1½ character times to respond before Data Request clears.

After writing a record, always check Endpoint. If Endpoint is set, do not give another Write; give Mark End of File and do not use the tape beyond that point.

When reverse spacing is stopped by a file mark, the tape is positioned on the loadpoint side of the mark. To process the records in the file just spaced over, first space forward over the file mark; a read function would simply stop at the mark.

### Readin Mode

The only requirements (beyond those given in §2.12) for readin mode with standard magnetic tape are that the data must be in the first record on the reel, in core dump format if the reel is on a 9-track unit, with odd parity at 556 bpi, and the reel must be mounted on transport 0. To select magnetic tape with the readin device switches, turn on the second, third and fourth switches from the left (340, the TMC device code).

Pressing the readin key causes the processor to place the tape control in a special readin sequence in which it executes Rewind on transport 0. The control continues to select transport 0 (Unit Hung sets but this has no effect) and when the tape reaches loadpoint, the control executes Read Record selecting odd parity, core dump format, and a density of 556 bpi. The data is transferred over the IO bus regardless of the type of control (the data channel is not used).



6.10 TIMING

The timing of tape functions depends not only on the density selected by the program, but also on the speed and tape handling characteristics of the selected transport. This section discusses timing for the various transports.

**Tape Transport TU10**

The tape processing speed of this transport is 45 inches per second  $\pm 4\%$ . The character and word times and the average transfer rates within a record are as follows.

Density	Character time in $\mu s$	Word time in $\mu s$			Words per second		
		7-track	9-track	Core dump	7-track	9-track	Core dump
200	111	666		555	1500		1800
556	40	240		200	4170		5004
800	27.8	167	111	139	6000	9000	7200

The data channel has just under a character time to respond to a data request, but because of double buffering in the TM10A, the program has an entire word time.

The start time is the time from the beginning of a function with the tape at rest in a normal record gap to the first word processed. Stop time is from the setting of Job Done to the setting of Load Next Unit.

Between Job Done and Load Next Unit, the program can give a new function without stopping tape.

	Time in ms
Write start	
7-track	12.5
9-track	8.9
Read start	
7-track	19.2
9-track	12.2
Loadpoint start	180
Forward stop	.9
Reverse stop	
7-track	3.6
9-track	.9
Erase or Mark End of File	100
At loadpoint	220
Rewind initiation	230

Note that Job Done and Load Next Unit set simultaneously in Rewind and in reverse spacing that stops at loadpoint. Rewinding an entire reel takes less than four minutes. This transport cannot be unloaded by the program.

To determine memory buffer size in real time applications, the programmer must know the total time between records in reading and writing. The

interrecord time without stopping tape is 21.9 ms for 7-track, 14.9 ms for 9-track (in writing, the time to the initial data request for the next record is 9.4 ms for 7-track, 6 ms for 9-track). For stopping between records, add 8 ms.

**Tape Transport TU20**

The tape processing speed of this transport is 45 inches per second  $\pm 1\%$ . The character and word times and the average transfer rates within a record are as follows.

Density	Character time in $\mu s$	Word time in $\mu s$			Words per second		
		7-track	9-track	Core dump	7-track	9-track	Core dump
200	111	666		555	1500		1800
556	40	240		200	4170		5004
800	27.8	167	111	139	6000	9000	7200

The data channel has just under a character time to respond to a data request, but because of double buffering in the TM10A, the program has an entire word time.

The start time is the time from the beginning of a function with the tape at rest in a normal record gap to the first word processed. Stop time is from the setting of Job Done to the setting of Load Next Unit.

Between Job Done and Load Next Unit, the program can give a new function without stopping tape.

Times given in parentheses are applicable to a control that has ECO TM10-00021, which is unlikely unless the system includes TU40 or later transports.

	Time in ms
Write start	14.2
Read start	
7-track	20.9
9-track	17.5
Loadpoint start	185
Write stop	4.7 (2.9)
Read or forward space stop	3.9 (2.1)
Reverse stop	9.8 (8)
Erase or Mark End of File	100
At loadpoint	220
Rewind initiation	1.8

Note that Job Done and Load Next Unit set simultaneously in Rewind and in reverse spacing that stops at loadpoint. Rewinding an entire reel takes about three minutes. This transport cannot be unloaded by the program.

To determine memory buffer size in real time applications, the programmer must know the total time between records in reading and writing. The interrecord time without stopping tape is 26.5 ms for 7-track, 23.1 ms for 9-track (in writing, the time to the initial data request for the next record is 12.3 ms for 7-track, 8.9 ms for 9-track). For stopping between records, add 5 ms.

### Tape Transport TU30

The tape processing speed of this transport is 75 inches per second  $\pm 1\%$ . The character and word times and the average transfer rates within a record are as follows.

Density	Character time in $\mu s$	Word time in $\mu s$			Words per second		
		7-track	9-track	Core dump	7-track	9-track	Core dump
200	66.7	400		333	2,500		3,000
556	24	144		120	6,950		8,340
800	16.7	100	67	83	10,000	15,000	12,000

The data channel has just under a character time to respond to a data request, but because of double buffering in the TM10A, the program has an entire word time.

The start time is the time from the beginning of a function with the tape at rest in a normal record gap to the first word processed. Stop time is from the setting of Job Done to the setting of Load Next Unit.

	Time in ms
Write start	11.7
Read start	
7-track	15.7
9-track	13.7
Loadpoint start	115
Write stop	3.2 (2.1)
Read or forward space stop	2.7 (1.6)
Reverse stop	5.9 (4.8)
Erase or Mark End of File	63
At loadpoint	130
Rewind initiation	1.07

Between Job Done and Load Next Unit, the program can give a new function without stopping tape.

Times given in parentheses are applicable to a control that has ECO TM10-00021, which is unlikely unless the system includes TU40 or later transports.

Note that Job Done and Load Next Unit set simultaneously in Rewind and in reverse spacing that stops at loadpoint. Rewinding an entire reel takes less than three minutes. This transport can be unloaded by the program.

To determine memory buffer size in real time applications, the programmer must know the total time between records in reading and writing. The interrecord time without stopping tape is 19.4 ms for 7-track, 17.4 ms for 9-track (in writing, the time to the initial data request for the next record is 7.7 ms for 7-track, 5.7 ms for 9-track). For stopping between records, add 5 ms.

### Tape Transport TU40

The tape processing speed of this transport is 150 inches per second  $\pm 3\%$ . The character and word times and the average transfer rates within a record are as follows.

<i>Density</i>	<i>Character time in <math>\mu</math>s</i>	<i>Word time in <math>\mu</math>s</i>			<i>Words per second</i>		
		<i>7-track</i>	<i>9-track</i>	<i>Core dump</i>	<i>7-track</i>	<i>9-track</i>	<i>Core dump</i>
200	33.3	200		167	5,000		6,000
556	12	72		60	13,900		16,680
800	8.3	50	33	42	20,000	30,000	24,000

The data channel has just under a character time to respond to a data request, but because of double buffering in the TM10A, the program has an entire word time.

The start time is the time from the beginning of a function with the tape at rest in a normal record gap to the first word processed. Stop time is from the setting of Job Done to the setting of Load Next Unit.

Between Job Done and Load Next Unit, the program can give a new function without stopping tape.

	<i>Time in ms</i>
Write start	
7-track	4.0
9-track	3.5
Read start	
7-track	6.0
9-track	4.5
Loadpoint start	55
Forward stop	.3
Reverse stop	.3
Erase or Mark End of File	30
At loadpoint	65
Rewind initiation	68

Note that Job Done and Load Next Unit set simultaneously in Rewind and in reverse spacing that stops at loadpoint. Rewinding an entire reel takes 66 seconds. This transport can be unloaded by the program.

To determine memory buffer size in real time applications, the programmer must know the total time between records in reading and writing. The interrecord time without stopping tape is 7.1 ms for 7-track, 5.6 ms for 9-track (in writing, the time to the initial data request for the next record is 3.1 ms for 7-track, 2.1 ms for 9-track). For stopping between records, add 2 ms.

# 7

## Disks and Drums

A disk or a drum is generally the largest random-access storage device in a computer system (a single unit usually holds more bits than all of core), and it also provides the fastest storage outside of core. These devices are exceptionally desirable as backup storage for memory, especially for storage of large files and for swapping in time sharing systems – while the currently active user programs are in core, inactive programs are stored on a disk or drum. Unlike magnetic tape, a disk or drum is constantly in motion and has a predetermined format with data blocks of fixed length. Hence, individual blocks are addressable, and at the simplest level, reading and writing may be the only functions the system need perform.

In a disk unit, the storage medium is similar to a phonograph record. Data is stored in tracks that are concentric circles on the disk surface (there may be tracks on both surfaces or on only one surface). The disk is further divided into sectors, *ie* pie-shaped sections. Hence the tracks are divided into arcs, each arc being the intersection of a track with a sector. In common terminology these arcs of a track are also referred to as sectors. The basic data block consists of a set of data words in one sector of a track. Generally a disk is divided into  $n + 1$  sectors, where  $n$  sectors are for data. The extra sector can be used for maintenance, but it also provides time for changing tracks without skipping a data sector. Besides data, each sector may contain information for synchronization, error checking, and addressing. The disk also may have extra tracks for timing and addressing information.

The inner tracks are shorter than those on the outer part of the disk, but timing is usually kept constant, with the result that the information density in the tracks increases toward the center. Sometimes tracks are grouped into rings, where the inner rings have fewer sectors; this grouping means that the time is constant only within rings but the variation in density is less.

In a drum unit the data is recorded in tracks that are circles on the surface of a cylinder. In this case the tracks are all the same length, so both timing and density are constant. As with a disk, the drum is divided into sectors, and the basic data blocks are recorded in the sectors of the tracks.

In a disk pack unit the storage medium is a removable stack of disks. Hence, not only is the storage capacity much greater, but the data can literally

This ambiguity seldom leads to any confusion. In relation to the disk as a whole, “sector” refers to an area; in relation to tracks and data blocks, “sector” refers to a segment of a track.

Note that a single disk with tracks on both surfaces also has cylinders, each consisting of a pair of tracks. But a cylinder configuration is not ordinarily used in the addressing scheme: tracks are consecutive across the surface, not back and forth from one surface to the other (in other words, a surface is specified by the most significant bit of the track address, rather than the least significant).

A cylinder is often defined as all the tracks that can be processed without repositioning the heads.

be stored on the shelf like magnetic tape while the drive is being used with another pack. Each disk surface has tracks and sectors. However, there are many surfaces, and the set of identically numbered tracks on the various surfaces constitutes a cylinder (logically a disk pack is equivalent to a drum pack). As before, the basic data block is a sector of a track, which is addressed as the intersection of a cylinder and a surface. In terms of the addressing scheme used in continuous data processing, the disk pack is treated as though it were a drum pack; the hardware counts through all the tracks (surfaces) in one cylinder at a time.

If there is a separate read-write head for each track, the average random-access time is a little over half a revolution; otherwise additional time may be required for head positioning. Since the storage medium is continuous, has a fixed format, and is in constant motion (both in speed and direction), functions can be limited simply to read and write, with an automatic search for an initially specified sector. However, in more complex systems, there may be a separate search function and even special functions for handling nondata parts of a sector.

A disk or drum system always consists of a control and a number of disk, drum, or disk pack units. In all cases, the program communicates with the control, which in turn governs all the units but effectively communicates with only one at a time. Data transfers between the control and device are governed by the control. The control is always connected to the IO bus, but only for the transfer of initial conditions and status. Once the program sets up the system for reading or writing, data transmission between control and memory is handled automatically via a separate memory bus (*ie* bypassing the central processor). To accomplish this the control is connected to the memory bus through a data channel or contains the necessary hardware for direct connection to the bus.

## PART I RC10 DISK/DRUM SYSTEM

This system consists of an RC10 control, which must be connected to memory via a DF10 data channel [§5.1], and up to four RD10 disk drives or RM10B drum drives in any mix. The system is used primarily for swapping programs in and out of core in time sharing.

The RD10 disk, which is manufactured by Burroughs, can store 512,000 36-bit words in blocks of thirty-two words each. While a block is being processed, data transfers occur at the rate of one word every 13.3  $\mu$ s. The average transfer rate over a number of blocks is 74,035 words per second.

The RM10B drum, which is manufactured by Bryant, is smaller but faster than the disk. It can store 345,600 words in 64-word blocks. Individual transfers occur every 4.3  $\mu$ s, with an average transfer rate of 220,800 words per second.

NOTE: If the drum operates on 50 Hz power, individual transfers occur every 5.2  $\mu$ s, with an average rate of 184,000 words per second.

The disk and drum are controlled by the program in essentially the same manner, but they differ in format and timing. In the text the words “disk” and “drum” are used only to refer specifically to the type of equipment named. In the discussion of characteristics common to both and the operation of the control, the words “unit” and “device” shall be taken to refer to either the disk or drum.

## 7.1 DATA FORMAT

Since the disk and drum have a separate head for each track, there is no head positioning time. If the data channel is free when an operation is begun, the control need wait only for the specified sector to reach the head, thus the average random-access time is just over half a revolution. Moreover, the control has a sector counter for each unit, allowing the program to determine the current position of a disk or drum at any time. Tracks and sectors are addressed by the control in binary-coded decimal form (BCD), *ie* the program supplies addresses as numbers with four bits per decimal digit.

Full words are transferred between memory and control, but data is transferred between control and device in 6-bit bytes. At the disk each byte is recorded serially, so the track contains one long string of bits. In reading, the bits are reassembled into bytes at the disk just as the bytes are reassembled into words at the control. The drum actually has six physical tracks for each logical, addressable track, and therefore the data is processed as a string of 6-bit characters.

While the control is supplying characters to the device for writing, it keeps a running exclusive or function of the characters in a parity register and writes the result at the end of the sector. Hence the final character in a sector is a longitudinal parity check character (LPCC). On the drum this character actually produces even parity along the length of the sector in each of the six physical tracks, whereas on the disk the LPCC is a function of six sets of bits, each containing every sixth bit along the track. When the control subsequently reads the sector, it checks parity by again keeping a cumulative function, including the LPCC (a nonzero result indicates an error).

In reading or writing the control starts at a given sector in a given track, as specified by the program. So long as the data channel remains in operation and there is no error stop, the RC10 continues reading or writing from one sector to the next along the track, and upon reaching the end of one track it switches automatically to the next track. Switching occurs at the beginning of the extra, maintenance sector to allow all transients to die down before the first data sector of the next track is encountered. Upon completing the final track on a given disk or drum, the control starts over at track 0, sector 0 of the same unit.

To provide data protection, the entire storage area (of all four devices) can be divided into two parts, one of which is locked against writing by the program. Switches on the control allow the operator to define the two areas by selecting a unit and track which form the boundary between the protected and unprotected areas. Other switches select whether the protected area is above or below the boundary and whether the boundary is included in the protected area. The area below the boundary includes all units numbered lower than the selected unit and all tracks numbered lower than the selected track on that unit; the upper area is the remainder of the storage medium.

**Disk.** The disk has 200 data tracks, numbered 0–199 BCD (100 tracks on each of the two disk surfaces), and is divided into eighty-one sectors, numbered 0–80 BCD, of which 0–79 are for data. Each sector contains thirty-two words. Thus, there are 2560 words per track and the capacity of an entire disk is 512,000 words. The disk runs at 1735 rpm, giving an average latency time of 17.6 ms.

**Drum.** The drum has ninety tracks, numbered 0–89 BCD, and is divided into sixty-one sectors, numbered 0–59 and 80 BCD, of which 0–59 are for data. While handling a drum, the control counts sectors as numbered (*ie* it counts to 59, jumps to 80, and then returns to 0) but the track counter continues all the way to 199 before returning to 0 just as it does with the disk. Hence, if the program lets a transfer run beyond track 89, the control will simply wait through the normal counting from 90 to 199 before resuming transfers (this requires approximately two seconds). Each sector contains sixty-four words. Thus, there are 3840 words per track and the total capacity of an entire drum is 345,600 words. With 60 Hz power the drum runs at 3450 rpm, giving an average latency time of 8.8 ms; at 50 Hz power the drum runs at 2870 rpm, giving an average latency time of 10.6 ms.

The extra sector is numbered 80 in order to use the maintenance circuitry in the RC10.

The drum actually uses binary addressing, and the BCD numbers supplied by the program must be translated. For the sectors, which must be consecutive, the conversion is standard. But to use the simplest decoding, the tracks as numbered are not physically consecutive on the drum. The translation is as follows.

BCD	Octal
000–007	000–007
010–017	010–017
⋮	⋮
070–077	070–077
080–087	100–107
008	110
009	111
018	112
019	113
028	114
029	115
038	116
039	117
048	120
049	121
058	122
059	123
068	124

## 7.2 INSTRUCTIONS

The control has a 36-bit assembly register AR, from which characters are sent to the unit for writing, and in which characters are reassembled into words in reading. The AR is connected directly to the channel bus for full word transmission between the data channel and control. The RC10 also contains a 6-bit longitudinal parity register for calculating and checking the LPCC, one 8-bit BCD sector counter for each unit, a 9-bit BCD track counter for use with the unit currently selected, and an 8-bit BCD sector register that points to the sector specified by the program for beginning the current operation. The contents of this last register are compared with the sector address sent from the device when the control is searching for the sector in which to begin reading or writing.

To run a disk or drum, the program must select a unit, track, sector and function, and supply an initial control word address for the data channel. To use the interrupt, the program should also assign a PI channel. Since all

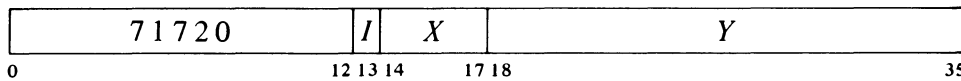


data transmission is handled through the DF10, interrupts are requested only when Done sets at the termination of an operation, either because the operation is complete or an error condition has terminated operation prematurely.

The RC10 device code is 170, mnemonic DSK. Both the condition and data IO instructions are used for control purposes. A second RC10 would have device code 174.

BCD	Octal
069	125
078	126
079	127
088	130
089	131

**CONO DSK, Conditions Out, Disk/Drum**



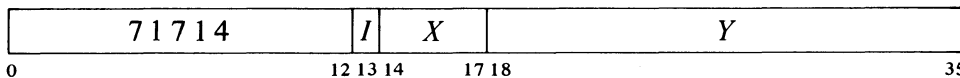
Select the sector counter specified by bits 18–19 of the effective conditions *E*, assign the interrupt channel specified by bits 33–35, and perform the functions specified by bits 20–32 as shown (a 1 in a bit produces the indicated function, a 0 has no effect).

SECTOR COUNTER	CLEAR UNIT ERROR	CLEAR TRACK- SECTOR ERROR	CLEAR NOT READY	CLEAR POWER FAILURE	CLEAR DEVICE PARITY ERROR	CLEAR DATA PARITY ERROR	CLEAR CONTROL WORD PARITY ERROR	CLEAR NO SUCH MEMORY	CLEAR ILLEGAL WRITE	CLEAR DATA LATE	WRITE CONTROL WORD	STOP	CLEAR DONE	PRIORITY INTERRUPT ASSIGNMENT			
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

*Notes.*

- 18–19      Unit addresses 0–3 select the corresponding sector counters for reading the current sector with a DATAI.
- 30          This bit requests that the channel write a control word and clears Control Word Written (CONI bit 30).
- 31          Clear Busy, set Done, requesting an interrupt on the assigned channel, release the data channel, and cease operation.

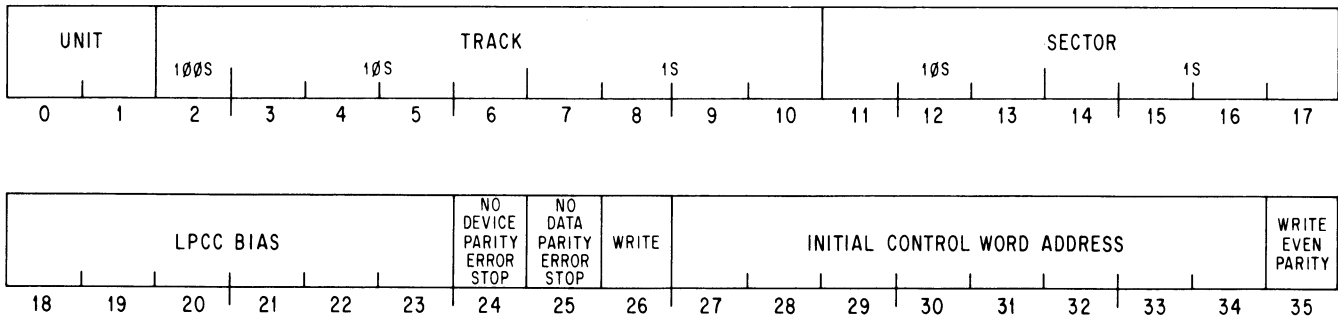
**DATAO DSK, Command Data Out, Disk/Drum**



If Busy is set, do nothing. Otherwise clear the assembly register, Search Error, Not Ready, Device Parity Error, Data Parity Error, Control Word

To load the RC10 command conditions, Busy must be 0 and the channel must not be connected, which always occurs when Busy clears. Should the combined condition fail even though Busy is 0, DATAO simply clears the registers it would otherwise load.

Parity Error, No Such Memory, Illegal Write, Data Late, Control Word Written and Done. Set up the RC10 command conditions according to the contents of location *E* as shown.



### Notes.

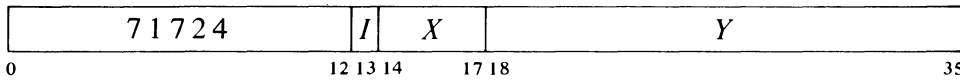
- 0-1 Numbers 0-3 address the device.
- 2-10 BCD numbers 0-199 address the track (only 0-89 are valid for a drum).
- 11-17 BCD numbers 0-79 address the sector (only 0-59 are valid for a drum).
- 18-23 Bias the cumulative LPCC calculation by this quantity. This is for maintenance only and should be zero in all normal programming.
- 24 A 1 in this bit prevents the RC10 from stopping when Device Parity Error sets.
- 25 A 1 in this bit prevents the RC10 from stopping when Data Parity Error sets.
- 26 A 1 in this bit causes the control to write data on the addressed unit when it finds the specified sector (0 specifies Read).
- 27-34 Send this address to the data channel when it is seized by the RC10.
- 35 A 1 in this bit selects even memory parity for disk/drum input (read) operations. This condition is for maintenance only; in all ordinary circumstances bit 35 must be 0 so the data channel will generate odd parity for words stored in memory.

The bias is included in the LPCC calculation (as an extra character) for all sectors in reading, but only for the first sector in writing (for writing the control clears the parity register between sectors). If there is no error, the result of reading a sector using zero bias is the bias used when the sector was written. Reading using the same bias should give a zero result.

The initial control word address is held in AR until the channel connects.

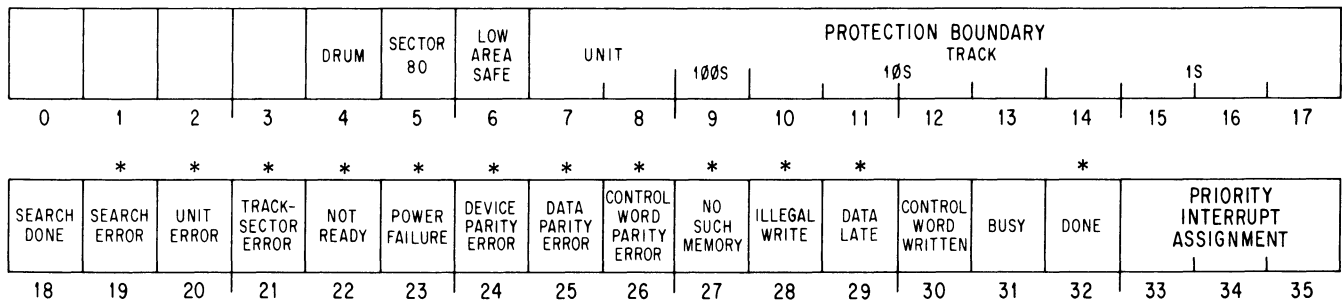
If Power Failure is set or the conditions supplied by this instruction result in the setting of Unit Error, Track-Sector Error, Not Ready or Illegal Write, the control sets Done, requesting an interrupt on the channel assigned by the last CONO DSK, and shuts down. If none of these error flags is set, the control sets Busy and waits for the data channel to connect.

**CONI DSK, Conditions In, Disk/Drum**



Read the status of the disk/drum system into bits 4–35 of location *E* as shown.

Note that CONSZ and CONSO can test only bits 18–35. To test bits 4–17 the program must give a CONI DSK,AC and then use a test instruction [§2.8].



*Notes.*

\*These bits cause interrupts.

DATAO clears bits 19, 22–30 and 32.

The setting of Done requests an interrupt on the assigned channel. This is the only flag that directly requests an interrupt, but Done is set by the setting of any error flag, although in some cases not immediately.

In many cases the setting of a flag causes the control to terminate; this means that the control clears Busy, sets Done, requesting an interrupt on the assigned PI channel, and ceases operation. If the data channel has not already been released by the control or disconnected of its own accord, then termination releases it.

NOTE: Unit Error sets Done only if it is already 1 when the RC10 is placed in operation; Done is not affected if Unit Error is set during execution of a function.

- 4 The currently addressed unit is a drum (0 indicates a disk).
- 5 The operator has selected the maintenance sector at the switch panel and the program can process only sector 80.
- 6 The protected area is below the boundary specified by bits 7–17 (0 indicates the high area is protected).
- 7–17 This is the unit and track selected at the switch panel as the boundary between the safe and unsafe areas.
- 18 The control has found the addressed sector and a block transfer is in progress.
- 19 The control began a search but terminated because it was unable to find the addressed sector within two revolutions.
- 20 There were two units set to the unit address given by the last DATAO; therefore the control set Done and did not go into operation.

The program cannot determine if the boundary is included in the protected area.

During operation Unit Error can be set only by a malfunction or *unwarranted* operator intervention, in which case the system hangs up.

*Notes (Continued)*

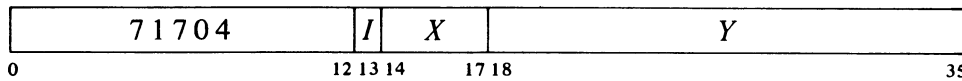
During operation this flag can be set only by a control logic malfunction, in which case the control terminates.

- 21 The last DATAO specified a track or sector address that was not a BCD number; therefore the control set Done and did not go into operation.
- 22 The addressed unit is not available to the program. If this bit is set when a DATAO is executed, the control sets Done and does not go into operation. If the unit goes off line during operation, the control terminates.
- 23 The voltage levels in the RC10 are not within tolerance. If this condition is indicated when a DATAO is executed, the control sets Done and does not go into operation. If power fails during operation, the control completes the current sector and then terminates.
- 24 The control read a sector with incorrect parity, and therefore terminated unless the last DATAO disabled the error stop.
- 25 The data channel discovered a parity error in a data word read from memory. Unless the last DATAO disabled the error stop, the setting of this bit causes the control to release the data channel immediately, continue to the end of the current sector writing zeros, and then terminate.
- 26 The data channel discovered a parity error in a control word read from memory. When this bit sets, the channel disconnects from the RC10, which continues to the end of the current sector, either writing zeros or throwing away the data read, and then terminates.
- 27 The memory addressed by the data channel did not respond within 100  $\mu$ s. When this bit sets, the channel disconnects from the RC10, which continues to the end of the current sector, either writing zeros or throwing away the data read, and then terminates.
- 28 In Write the control specified a track that fell in the protected area. If this bit is set when a DATAO is executed, the control sets Done and does not go into operation. If the control counts into the protected area while writing, it terminates.
- 29 The data channel did not send data to the RC10 or accept data from the RC10 when it was expected to. When this bit sets, the control releases the channel but continues to the end of the current sector, either writing zeros or throwing away the data read, and then terminates.
- 30 The data channel has stored a control word in memory as requested by CONO bit 30 through the RC10.

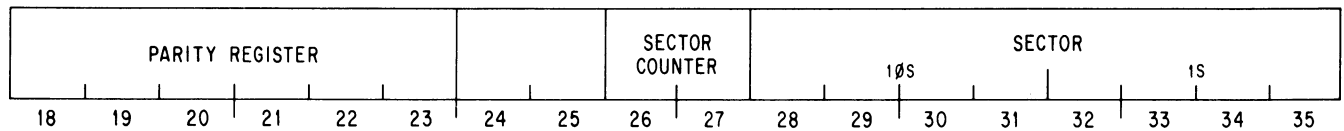
While waiting for the memory, the control may set Data Late, which releases the channel so the memory error does not show up at all.

- 31 The control is in operation, *ie* it has received correct conditions from a DATAO and is either waiting to seize the data channel, searching, or processing the tracks on a disk or drum.
- 32 The RC10 has terminated, either because a data channel block transfer has been completed or an error condition shut down the system prematurely.

**DATAI DSK, Command Data In, Disk/Drum**



Read additional status information from the disk/drum system into the right half of location *E* as shown.



*Notes.*

Following a sector in which Device Parity Error sets and stops the control, the program can inspect the result of the parity calculation by reading the parity register. Assuming zero bias is used, 1s in bits 18–23 indicate which of the six sets of bits definitely contained errors (these bits are in six physical tracks on the drum).

Bits 28–35 indicate the current position of the disk or drum whose unit address appears in bits 26 and 27; the latter bits indicate the sector counter selected by bits 18 and 19 of the previous CONO.

While data is being processed, the parity register changes too fast for the program to read it meaningfully. Since the storage medium (and hence the counter) is in constant motion, the program should repeat the DATAI until two consecutive readings agree.

**7.3 PROGRAMMING CONSIDERATIONS**

To write data the program should first give a CONI to determine what area is protected. Of course the program can also check Busy to make sure the system is not already in use. To use the interrupt, give a CONO to assign the channel. To place a disk or drum in operation, give a DATAO that specifies the initial sector (unit, track, and sector), the function to be performed (Read or Write), the initial control word address for the data channel, and whether the control is to stop if parity errors are discovered in the data received from memory or the device.

If the control is already busy, it simply ignores the DATAO. Otherwise, the control loads the command information and checks the various conditions listed in the DATAO instruction description to determine if it can perform as expected; if it cannot, the control sets Done and shuts down. If the control can continue, it sets Busy, waits until it is connected to the data channel, then waits an additional 100  $\mu$ s to settle down and begins the search for the specified initial sector. If data transfers can begin at an arbitrary sector, the program could save search time by selecting the sector counter for the addressed unit when giving the initial CONO that assigns the PI channel. The program can then issue a DATAI to determine the position of the unit, and specify a sector that will be encountered soon when giving the DATAO. As soon as the control finds the addressed sector, it starts transferring data to or from the addressed track. If the control fails to find the desired sector within two revolutions of the disk or drum, it sets Search Error and terminates.

The program can incorrectly address a track or sector within a disk only by giving a non-BCD number. On the other hand, there are BCD numbers that are not valid for the drum, *ie* there are no tracks or sectors corresponding to the numbers. An erroneous track selection is actually taken care of by one of the other conditions: selecting an initial track in the range 90–199 causes the addressed drum to generate its not-ready signal, preventing the control from starting. However, specifying an initial sector in the range 60–79 does not stop the control; instead the control goes ahead, then gives up when its search for the nonexistent sector is unsuccessful. Note that with a disk, a search error can result only from a hardware malfunction – all BCD numbers are valid.

Once transfers begin, the control processes data continuously, going through all the sectors from one track to the next until it is stopped by an error or the data channel retrieves a zero control word. Some errors cause immediate termination even in the middle of a sector, whereas for other errors the control releases the channel but continues to the end of the sector before terminating (see the descriptions of the flags read by a CONI). When the control processes part of a sector without the data channel, it simply continues the function it is already doing – in Write the control writes zeros in the remaining data positions in the sector and then writes an LPCC; in Read the control reads the data and checks the parity. The program can stop the control at any time by giving a CONO DSK,20.

While processing data, the control counts from one sector to the next and changes to the next track at the beginning of sector 80. If the control should count into the protected area while performing Write, it terminates immediately. Upon completing track 199, the counter simply recycles and the control begins over at sector 0 of track 0 on the same unit. If the addressed unit is a drum and the control counts beyond track 89, data transfers cease for about two seconds while the control counts through all the missing tracks and returns to zero.

Usually a block transfer processes an integral number of sectors, so the channel stops in the gap between them. In Write the channel sends a word before it is written, so there is plenty of time to stop. However, in Read the channel must wait for the last word after it is read. Thus, if the word count overflows at a gap, the channel may not have time to get a halt from its control word list before the device starts reading the first character in the next sector, even though the RC10 has not received any data from the device. When this happens the control terminates, but the device reads the rest of the sector. However, the program may set up the system so that the channel reconnects before the device finishes. To compensate for this, if the device is still reading a data track when the data channel is seized, the control waits until the end of the sector before beginning the 100  $\mu$ s wait that precedes the search.

**Timing.** At 1735 rpm the disk takes 34.9 ms per revolution. The time to traverse one sector is 431  $\mu$ s including nondata time (gap plus LPCC) of 4.4  $\mu$ s. Search time after the data channel connects is 35.3 ms maximum, 17.6 ms average. During processing, word transfers occur every 13.3  $\mu$ s. Between sectors in Read the channel has about 17  $\mu$ s to store the last word and execute control words; in Write the channel has about 13  $\mu$ s for control words.

At 3450 rpm (60 Hz power) the drum takes 17.4 ms per revolution. The time to traverse one sector is 285  $\mu$ s including nondata time (gap plus LPCC) of 8.6  $\mu$ s. Search time after the data channel connects is 17.7 ms maximum, 8.8 ms average. During processing, word transfers occur every 4.3  $\mu$ s. Between sectors in Read the channel has about 8  $\mu$ s to store the last word and execute control words; in Write the channel has about 6  $\mu$ s for control words.

If the drum operates on 50 Hz power, it rotates at 2875 rpm, taking 20.9 ms per revolution. The time to traverse one sector is 342  $\mu$ s including nondata time (gap plus LPCC) of 10.3  $\mu$ s. Search time after the data channel connects is 21.2 ms maximum, 10.6 ms average. During processing, word transfers occur every 5.2  $\mu$ s. Between sectors in Read the channel has about 10  $\mu$ s to store the last word and execute control words; in Write the channel has about 7  $\mu$ s for control words.

**EXAMPLE.** Consider this situation: The main program is processing data from a disk and writes the results back on the same disk. Data is handled in terms of single tracks, and it is always desirable to complete a block transfer as quickly as possible. Each time the program has a block ready or needs a block, it determines whether the disk and data channel are available, and when they are, the program jumps to a disk service routine DSKSER. In the location following the subroutine call, the program supplies the address of the first location in the block in the right half, the binary number of the

If the operator selects the maintenance sector from the switch panel but the system is under program control, a DATAO will read or write in sector 80. The sector selected by the program is ignored, protection does not apply, and the control terminates upon completing the one sector.

track in the left half, and a 0 or 1 respectively in bit 0 to indicate whether the block should be read or written. Hence, the call is of this form:

```
JSP    T,DSKSER
...    ;Arguments
...    ;Return here
```

The service routine uses four accumulators, T, T1, T2 and T3. In an accumulator of flags addressable as F (most likely AC 0), bits 30 and 31 are reserved for these disk operations. The interrupt routine sets bit 31 when the block is complete, and the service routine uses bit 30 to tell the interrupt routine whether the track is being processed as one continuous block from sector 0 to sector 79 or as two subblocks beginning somewhere within the track. The example uses disk 2 and interrupt channel 3, although these can be changed easily by the main program. Being one complete track, each block transfer is 2560 words ( $5000_8$ ). The disk control word list starts at location 750, which contains a jump to 752 so that 751 is available for writing control words; the first control word computed by the routine thus goes in 752. The service routine and associated interrupt routine are as follows.

```
DSKSER:  CONO    DSK,400003    ;Select sector counter 2, PI channel 3
          DATAI  DSK,T1      ;Read counter
          ANDI    T1,377      ;Strip it out
          DATAI  DSK,T2      ;Read again
          ANDI    T2,377      ;Strip again
          CAME    T1,T2       ;Are they the same?
          JRST    DSKSER+1    ;No, try again

          MOVEI   T3,-1(T)    ;Get block address -1

          CAIGE   T1,170      ;≥ BCD 78 = 1708 ?
          JRST    DSKANY      ;No, must compute
          CAIE    T1,200      ;= BCD 80 = 2008 ?
          JRST    .+3         ;No, 78 or 79
          MOVEI   T1,↑D32     ;Yes, start at sector 1 (33d word)
          JRST    DSK1

          HRLI    T3,-5000    ;Get word count
          MOVEM   T3,752      ;Deposit control word
          TRZ     F,40         ;Bit 30 = 0 means one block
          CLEAR   T3,         ;Start at sector 0
          JRST    DSKTRK     ;Go to track calculation
```



```

DSKANY:  MOVEI  T2,17      ;Mask for units character
         ANDI   T2,(T1)    ;Strip out units
         LSH   T1,-4      ;Right justify tens character
         IMULI  T1,↑D10    ;Convert to binary
         ADDI   T1,2(T2)   ;Add units plus 2

         LSH   T1,5        ;Multiply by 32 (words/sector)
DSK1:    MOVN   T2,T1      ;Negate number of words
         HRL   T3,T2      ;Control word for second part
         MOVEM  T3,DSKDF   ;Put in interrupt routine
         HRLI  T3,-5000    ;Get full word count
         HRLS  T1         ;Duplicate count for second part
         ADD   T3,T1      ;Add it to word count and address
         MOVEM  T3,752     ;Control word for first part
         TRO   F,40       ;Bit 30 = 1 means two blocks

         HLLI  T1,0       ;Clear AC left
         LSH   T1,-5      ;Get back sector number
         IDIVI  T1,↑D10    ;Get BCD tens character
         LSH   T1,4       ;Move to tens position
         IOR   T1,T2      ;Insert units
         HRLZ  T3,T1      ;Put in DATAO position

DSKTRK:  HLRZ   T1,(T)    ;Get track
         ANDI  T1,377     ;Wipe out read/write bit
         IDIVI  T1,↑D10    ;Get tens
         CAIL  T1,↑D10    ;Shortcut: range (0-199)/10 = 0-19
         ADDI  T1,6       ;Get BCD hundred and tens
         LSH   T1,4       ;Put in position
         IOR   T1,T2      ;Insert units
         ROT   T1,-↑D11   ;Put in DATAO position

         SKIPGE (T)      ;Is read/write flag set?
         TRO   T1,1000    ;Yes, set DATAO write bit

         MOVE  T2,DSKCOM  ;Get basic DATAO word
         IOR   T2,T1      ;Make one with sector 0
         MOVEM  T2,DSKCM  ;Put in interrupt routine
         IOR   T3,T2      ;Make another with starting sector
         MOVE  T2,DSKINW  ;Get interrupt instruction
         MOVEM  T2,46     ;Set up for channel 3
         CLEARM 753      ;Put halt in control word list
         DATAO DSK,T3    ;Start disk
         JRSTF 1(T)      ;Return

```

Processing will begin at the second sector following the present one.

It is assumed the program has given a proper track address ( $\leq 199 = 307_8$ ).

```

DSKINW: JSR      DSKINT      ;Interrupt instruction
DSKCOM: 400000000750      ;Basic DATAO word

DSKINT: 0                  ;Interrupt routine
        CONSZ  DSK,377720  ;Check all errors (including Busy)
DSKERO: JRST     DSKERR     ;Out to disk error routine
        TRCN   F,40        ;Complement flag, track finished?
        JRST   DSKDON     ;Yes

        EXCH   17,DSKDF    ;No, finish track
        MOVEM 17,752      ;Data channel control word
        CLEARM 753        ;Halt
        EXCH   17,DSKDF    ;Restore AC 17
        DATAO DSK,DSKCMI ;Start second part
        JEN    @DSKINT     ;Return

DSKDON: TRO      F,20      ;Set track done flag
        CONO    DSK,10     ;Clear Done to drop interrupt
        JEN    @DSKINT     ;Return

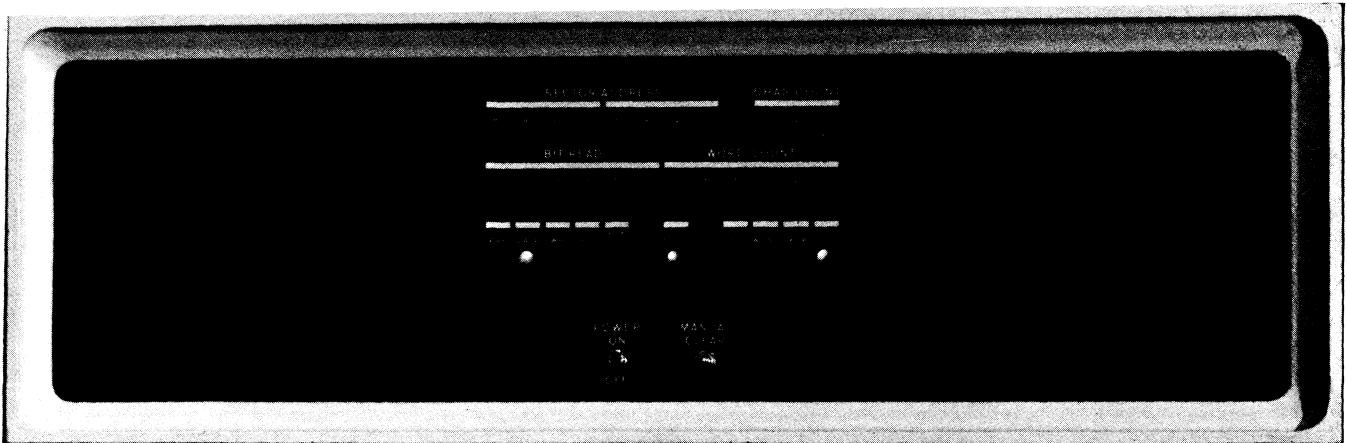
DSKDF:  0
DSKCMI: 0

```

#### 7.4 OPERATION

The RD10 disk has pushbuttons for turning power on and off (POWER ON lights green when on) and a red not-ready indicator.

The RM10B drum unit contains extra DEC logic to make it compatible with the disk for operation from the control. The panel at the top of the left cabinet in the unit is associated with this logic. The upper two rows of lights display counters that keep track of the sector, the characters within each word, and the words in each sector. The left half of the second row displays each character read from the drum (a buffer is required in reading because of possible skew in picking up the parallel characters across six tracks). Among the five lights at the left in the bottom row, WR and RD indicate the function, and the remaining lights, PRE, DATA and BLK END, indicate the position in a sector (*ie* in the part preceding the data, in the data area, or at the last word or the LPCC). RCOV controls the spacing between characters. The next two lights indicate sector and word counting errors, *ie* that the sector count was not 59 when the drum indexed to the next track, or the word count was not zero at the beginning of a sector. SKEW is not used; SUR is the ready indicator. Besides the drum conditions that generate the not-ready signal (*ie* that turn off SUR), the drum is also unavailable to the program if a sector or word counting error occurs or the control supplies an address for a track that does not exist on the drum. The switches at the bottom of the panel control power and clear the logic following a sector or word counting error.



Controls for the RC10 are located on a switch panel behind the doors just below the indicator panel. The rotary switches at the bottom assign unit addresses to the devices, where A, B, C and D correspond to the device cable connectors at the back of the RC10. The off position takes the unit off line.

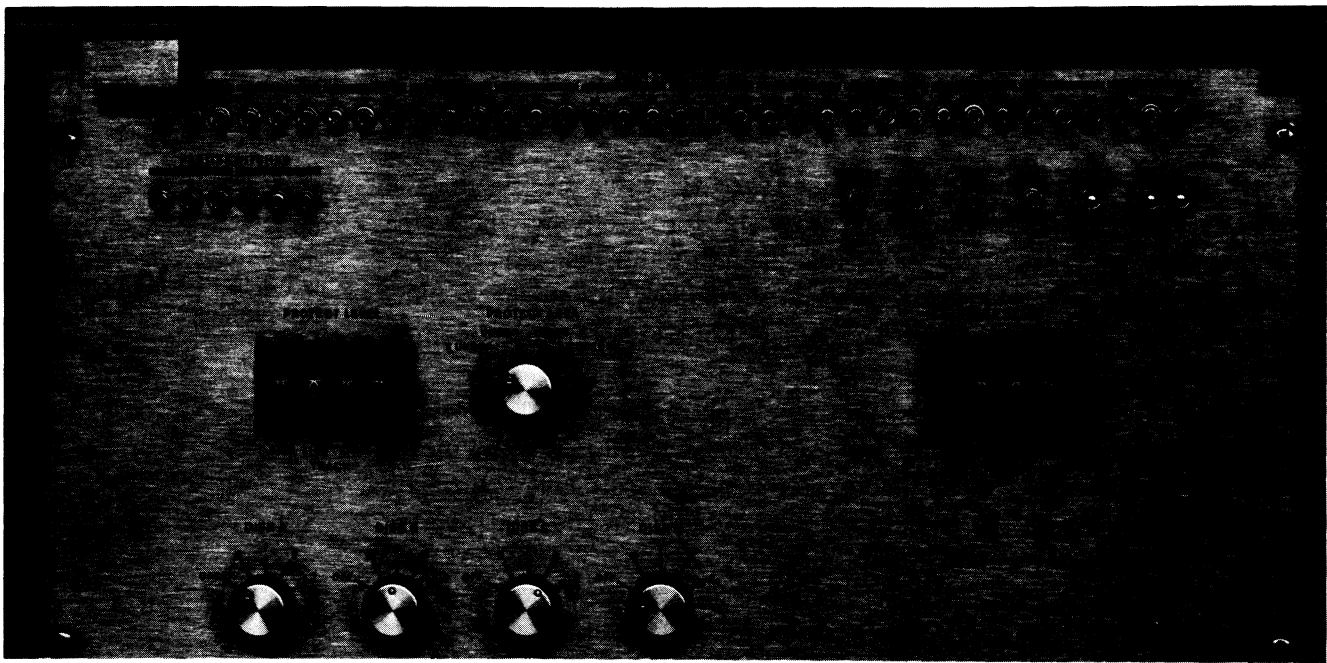
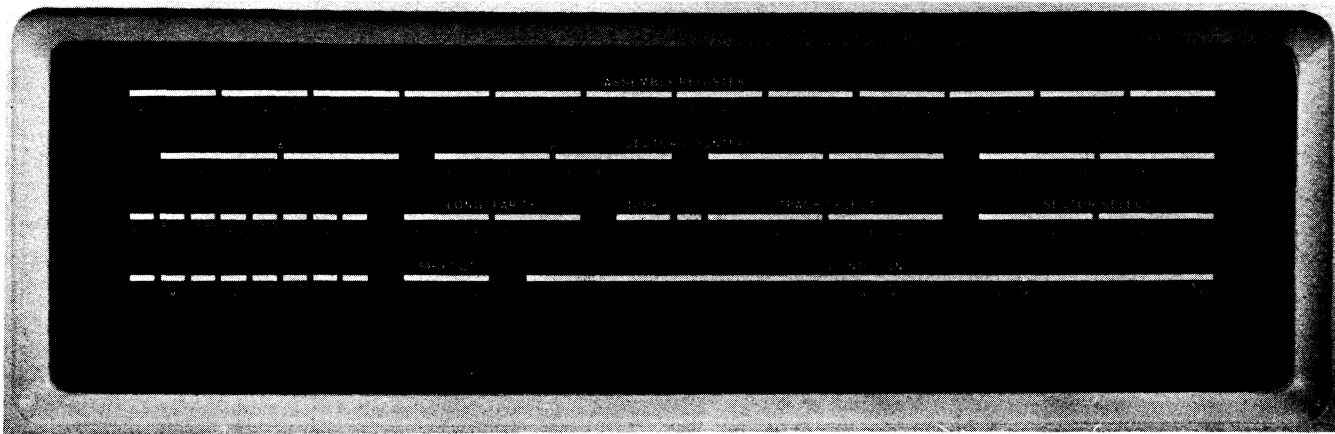
At left center are thumbwheel switches and a rotary switch for selecting the boundary between the safe and unsafe areas and for specifying which area is safe; the protected area may be under (including the boundary) or exclusively under, over or exclusively over. The boundary thumbwheels allow selection of decimal digits where the device digit (DISK) is interpreted modulo 4, the hundreds digit in the track modulo 2.

The data and parity switches at the top supply data and LPCC bias for offline operations. Control over system operation is accomplished by means of the lever switches and pushbuttons at the upper right below the data switches (the first two at the left are momentary contact, the next two are toggles). For online operation the toggles must be set to NORMAL and REMOTE. Setting the third switch to MAINT SEG allows the program to process sector 80. Setting the fourth switch to LOCAL disables the third switch but enables the remaining switches. In local control, the RC10 can read sector 80 or write the word supplied by the data switches into sector 80. To select Write push up the second switch (WRITE); to specify Read do not operate the switch; to change from Write to Read during offline operation, press CLEAR. Pressing START causes the control to process the maintenance sector in the track selected by the thumbwheels at right center. The control repeatedly processes the same sector unless the operator pushes up the CHANGE TRACK switch, in which case the control counts through the tracks. Pressing STOP causes the control to terminate the next time it reaches the end of sector 80. Pressing CLEAR stops the control immediately and clears the logic. To process the maintenance sector only once, hold STOP on while pressing START.

RM10B Control Panel

*CAUTION*

Do not manipulate these switches while the RC10 is operating or can be placed in operation. Assign unit addresses only when the control is off line or the processor is not running.



RC10 Switch and  
Indicator Panels

The upper three rows of lights on the indicator panel display the contents of the assembly register, the four sector counters associated with the devices, the parity register, the device (DISK) and track currently selected, and the initial sector selected by the last DATAO. Of the flag and control indicators, BUSY, DONE and the PI assignment lights at the right end of the bottom row are self-explanatory. The remaining lights are as follows.

ACTV	The control is waiting for the channel or is connected.
ACTV BUF	The control is waiting for the channel.
CHAN STTD	The channel has been seized (started).

INH	The control has released the channel.
CHAN PLS	The control has received the channel pulse.
CHPS BUF	The control has received at least one channel pulse.
SRCH ER BUF	The control has searched for one revolution without finding the addressed sector.
COIN	The low order digits of the specified sector address coincide with the sector address from the device.
SRCH	The control is searching.
SRCH CMP	Search Done (complete).
SRCH SYNC	Synchronizing flipflop for SRCH.
GAP	The head is in the gap between sectors.
GEN CLR	The control is not in operation.
MAINT SEG	The control can process only sector 80.
LCL	The control is off line (local).
SRCH RDY	The control is ready to start a search.
SHIFT CT	Switch-tail ring counter that controls distribution and assembly of characters between AR and device.
SCDS 1,2	Sector counter selected by bits 18 and 19 of a CONO, and indicated by bits 26 and 27 of a DATAI (sector counter device select).
SRCH ERR	Search Error.
SUDD FAIL	Suppress device designation failure (flipflop set by CONO bit 20 – clear Unit Error).
SUTS FAIL	Suppress track select failure (flipflop set by CONO bit 21 – clear Track-Sector Error).
PS FAIL	Power Failure.
DPE	Device Parity Error.
DDPE STOP	Disable device parity error stop (flipflop set by DATAO bit 24).
CPE	Data (channel) Parity Error.
DCPE STOP	Disable channel (data) parity error stop (flipflop set by DATAO bit 25).
CHAN CPAR	Control Word Parity Error.
WRIT	Write function.
NOEX MEM	No Such Memory (nonexistent memory).
ILL WRIT	Illegal Write.
SURL LTCH	Not Ready (selected unit ready level latch).
OVER RUN	Data Late.
CWX COMP	Control Word Written (control word transfer complete).
WTEV PAR	Write even parity (flipflop set by DATAO bit 35).

## PART II

### RP10 DISK PACK SYSTEM

The disk pack system consists of an RP10 control, which must be connected to memory via a DF10 data channel [§5.1], and up to eight RP02 or RP01 disk pack drives in any mix. The system is primarily for storage of large files or files for a large number of users.

Both disk pack models store data in blocks of 128 words and operate at the same rotational speed. The RP02 has twice as many surfaces and twice as many sectors as the RP01, and thus has four times the storage capacity at twice the transfer speed. The capacity of the RP02 is 5,120,000 36-bit words. While the data part of a sector is being processed, transfers are at the rate of one word every 14.8  $\mu$ s; although an entire cylinder of 25,600 words can be processed in 500 ms, the average transfer rate over a number of consecutive cylinders is 48,762 words per second. The RP01 can store 1,280,000 words and individual transfers occur every 29.6  $\mu$ s; a single cylinder containing 6400 words can be processed in 250 ms, but the average rate over consecutive cylinders is 23,273 words per second.

#### 7.5 DATA FORMAT

The RP02 has eleven disks and the RP01 has six, but the outer surfaces are not used.

The DEC software that formats disk packs also determines which sectors are unusable and records this information in the pack. The Monitor reserves the last three tracks on every surface (310–312) for diagnostic procedures and makes the rest, except for bad sectors and a few Monitor overhead sectors, available for user storage.

The program may be able to skip a bad sector in writing by supplying a throwaway block. In reading, the control should terminate the function and restart following a bad sector.

The RP02 disk pack has twenty surfaces divided into ten sectors, addressed as octal 0–23 and 0–11 respectively; the RP01 has ten surfaces divided into five sectors, numbered 0–11 and 0–4 octal. Each surface has 203 tracks, addressed as 0–312 octal, of which the manufacturer guarantees 200 tracks to be usable. Any bad tracks in a pack are indicated on a label on the base plate. Since there are seldom as many as three bad tracks on a surface, and even in a bad track some or most of the sectors may be usable, the initial capacity of an individual pack is usually greater than that given above.

The drive has one head per pack surface, with all heads mounted in a single carriage so that they are positioned simultaneously. The maximum time required to move the heads from one cylinder to the next is 20 ms, and at most 80 ms are required for motion from one extremity to the other. Once the positioning operation is complete, the control must wait for the specified sector to reach the heads. This requires just over half a revolution on average (13.1 ms on the RP02, 13.7 ms on the RP01).

In reading or writing, the control starts at a given sector in a given track, as specified by the program. So long as the data channel remains in operation and there is no error stop, the RP10 continues from one sector to the next along the track, and upon reaching the end of one track it switches automatically to the next track on the cylinder (*ie* to the same track on the next surface). Upon completing the final track on the cylinder, the control terminates the function. Since each sector contains 128 data words, the RP02 with ten sectors and twenty surfaces has 1280 words per track, 25,600 words per cylinder; 200 cylinders gives a total capacity of 5,120,000 words. The

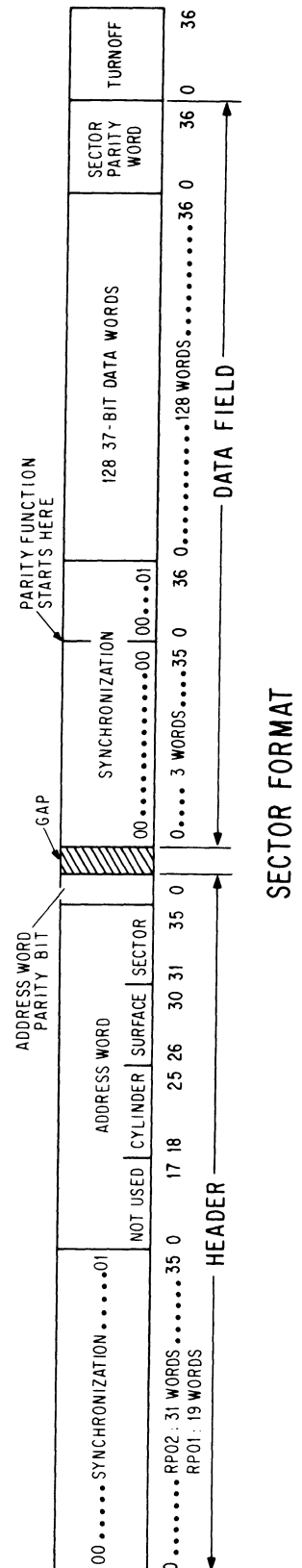
RP01 with five sectors and ten surfaces has 640 words per track, 6400 words per cylinder, and a capacity in 200 cylinders of 1,280,000 words. Each data function executed by the control can process one complete cylinder (provided of course that the cylinder contains no unusable sectors). Since repositioning the heads from one cylinder to the next requires a maximum of 20 ms, and at 2400 rpm each revolution takes 25 ms, the control can process a large file in contiguous tracks simply by pausing for one revolution between cylinders.

Transfers between memory and control are of full words, but data is transferred between control and device in a serial string of bits. Each word is written as thirty-seven bits, where the last bit produces odd parity for the word. The control also keeps a running exclusive OR of the 37-bit words and writes the complement of the result as an odd longitudinal parity word at the end of the sector. When the control subsequently reads a sector, it checks the parity of each word and of the whole sector.

Of the functions the system can perform, three actually process the disk sectors; these are Write Format, Write Data, and Read Data. Sector boundaries are defined by physical properties in the disk pack, but every sector must be formatted as shown here by a program that supplies the necessary information through the data channel. The synchronization zone at the beginning of the sector requires thirty zero words (eighteen for the RP01) and then a single word of 0s with a 1 in bit 35. Following this is an address word whose left half may contain any information desired by the program, but whose right half must contain the number of the sector and the numbers of the cylinder and surface in which the sector lies. The program must then give a word in which bit 0 is an odd parity bit for the address word and the remaining bits are 0s (the slight discrepancy between this discussion and the diagram is explained below). This is followed by three more zero words, but the control turns on the parity circuits following the second of these, so the third word (the fourth of the data synchronization zone) is written as thirty-seven bits with 0s in bits 0–35 and a 1 (for odd parity) in bit 36. The program completes its role in the procedure by supplying 128 data words that are automatically written as 37-bit words with parity. Following the data the control writes a 37-bit parity word for the sector and then rewrites it before turning off the write circuits (this action garbles the end of the last word, but this is of no consequence since the word is never read).

Data functions that subsequently process the sector synchronize on the leading 0s, and the 1 indicates the beginning of the address word. For reading or writing, the control reads the address word and compares the address information with that supplied by the program for the current function. If the two addresses are identical and the address word with bit 0 of the next word has odd parity, the search is regarded as successful and the control processes the rest of the sector (note that although the left half of the address word is not used in any way by the control, it is included in the parity check).

- After the first in a series of consecutive sectors processed by a single function, there is no search and no check of the address word.



For Write Data the control waits  $2.5 \mu s$  following the parity bit before turning on the write circuits. Hence, once a sector is processed by Write Data, its format varies slightly from the original in that there is a small gap between the address word parity bit and the data synchronization zone (the gap contains the remnants of the beginning of the original synchronization). The control now automatically writes four zero words where the fourth has thirty-seven bits, containing a 1 in the parity position. The control then writes 128 words supplied through the data channel, writes a parity word, and repeats it to complete the sector.

After a search is successful for Read Data, the control syncs to the first four words in the data field, sends the 128 data words to memory through the data channel, checking the parity of each, and then reads the parity word to check the parity of the data field.

To provide data protection, a switch at each drive allows the operator to lock out the entire pack against writing by the program.

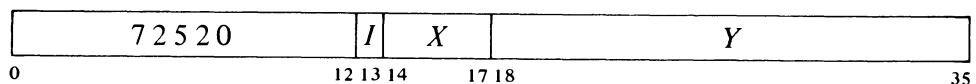
## 7.6 INSTRUCTIONS

The data path between the data channel and a disk pack includes a 36-bit assembly register AR and a 6-bit shift register SR. In writing, words supplied by the channel are received by AR, which parcels them out in 6-bit bytes to SR, which transmits the bytes serially to the selected pack. In reading, SR assembles the bits into bytes, which in turn are assembled into words in AR for transmission to the channel. The RP10 also contains one 5-bit sector counter for each drive and a set of eight Attention flags that request interrupts for individual drives. Each drive contains an 8-bit register that indicates the last cylinder to which the program ordered the drive to position its heads.

To execute any function the RP10 can perform, the program must specify the op code of the function and select a drive. To position the heads requires specification of a cylinder. To process the disk tracks, the program must also select a surface and sector, and supply an initial control word address for the data channel. To use the interrupt, the program should assign a PI channel. Since all data transmission is handled through the DF10, interrupts are requested only when an Attention flag sets or when Done sets at the termination of an operation, either because the operation is complete or an error condition has terminated it prematurely.

The RP10 device code is 250, mnemonic DPC. Both the condition and data IO instructions are used for control purposes. A second RP10 would have device code 254.

**CONO DPC,            Conditions Out, Disk Pack Control**





Assign the interrupt channel specified by bits 33–35 of the effective conditions *E*, and perform the functions specified by bits 20–32 as shown (a 1 in a bit produces the indicated function, a 0 has no effect).

		CLEAR POWER FAILURE	CLEAR SEARCH ERROR	CLEAR DATA LATE	CLEAR NO SUCH MEMORY	CLEAR PARITY ERROR FLAGS		CLEAR ILLEGAL WRITE	CLEAR ILLEGAL DATAO	CLEAR SECTOR ADDRESS ERROR	CLEAR SURFACE ADDRESS ERROR	WRITE CONTROL WORD	STOP	CLEAR DONE	PRIORITY INTERRUPT ASSIGNMENT		
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

*Notes.*

- 20 Clear Power Failure if the failure condition has been corrected.
- 24 Clear all of the parity error flags (CONI bits 14–17).
- 30 This bit requests that the channel write a control word and clears Control Word Written (CONI bit 30).
- 31 Clear Busy, set Done, requesting an interrupt on the assigned channel, release the data channel, and cease operation immediately.

**DATAO DPC, Command Data Out, Disk Pack Control**

7 2 5 1 4				I	X	Y											
0				12 13 14	17 18												35

If Busy is set, do nothing but set Illegal DATAO. Otherwise clear the assembly register, all Parity Error flags (Control Word, Sector, Channel Data Word, Disk Word), Search Error, Data Late, No Such Memory, Illegal Write, Illegal DATAO, Sector Address Error, Surface Address Error, Control Word Written and Done. Set up the RP10 command conditions according to the contents of location *E*. The meaning of bits 6–35 of the condition word depends on the function whose code is given in bits 0–2. The typical interpretation of the several parts of the word is as follows (variations in format for individual functions are given below).

CODE	DRIVE	CYLINDER			SURFACE		SECTOR		INITIAL CONTROL WORD ADDRESS								
0	2 3	5 6	13 14			18 19		23		27							34

0–2 Code for the function as follows.

- 0 Read Data
- 1 Write Data
- 2
- 3 Write Format
- 4 Position Heads
- 5 At Ease
- 6 Select Drive
- 7 Recalibrate

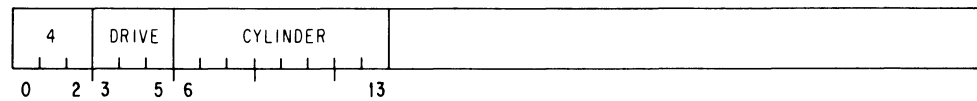
**CAUTION**

Do not use op code 2. It is not a no-op, and is subject to change for future use.

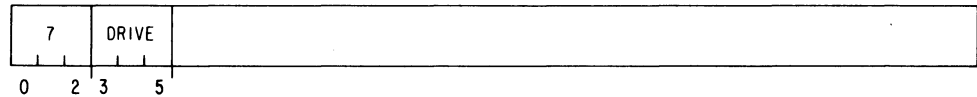
- 3-5 Numbers 0-7 address the drive.
- 6-13 Numbers 0-312 address the cylinder.
- 14-18 Numbers 0-23 address the surface in the RP02; numbers 0-11 address the surface in the RP01.
- 19-23 Numbers 0-11 address the sector in the RP02; numbers 0-4 address the sector in the RP01.
- 27-34 In functions that process the disk sectors, this address is sent to the data channel when it is seized by the RP10.

The initial control word address is held in AR until the channel connects.

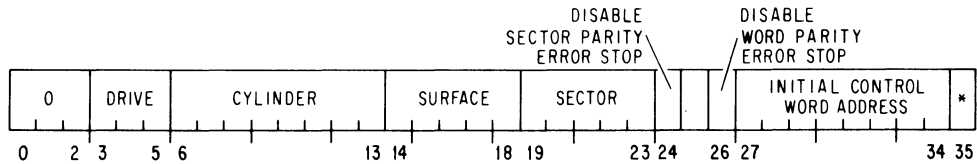
*Position Heads*



*Recalibrate*



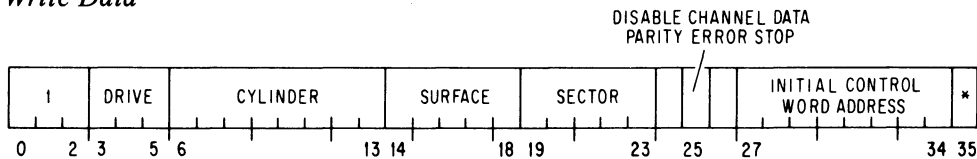
*Read Data*



- 24 A 1 in this bit prevents the RP10 from stopping when Sector Parity Error sets.
- 26 A 1 in this bit prevents the RP10 from stopping when Disk Word Parity Error sets.

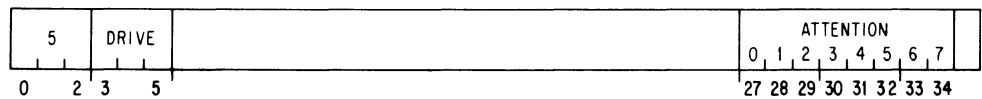
\*A 1 in this bit selects even memory parity for data channel input (read) operations. This is for maintenance only; in all ordinary circumstances bit 35 must be 0 so the data channel will generate odd parity for words stored in memory.

*Write Data*



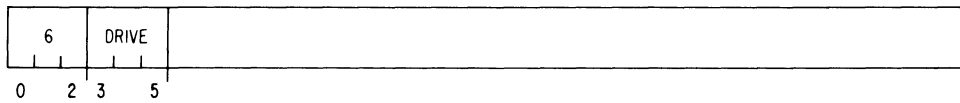
- 25 A 1 in this bit prevents the RP10 from stopping when Channel Data Parity Error sets.

*At Ease*

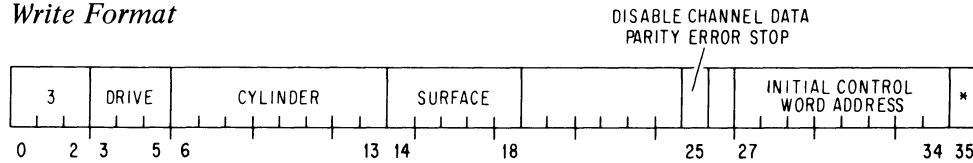


- 27-34 Clear the Attention flags selected by these bits.

Select Drive



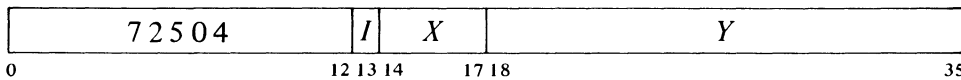
Write Format



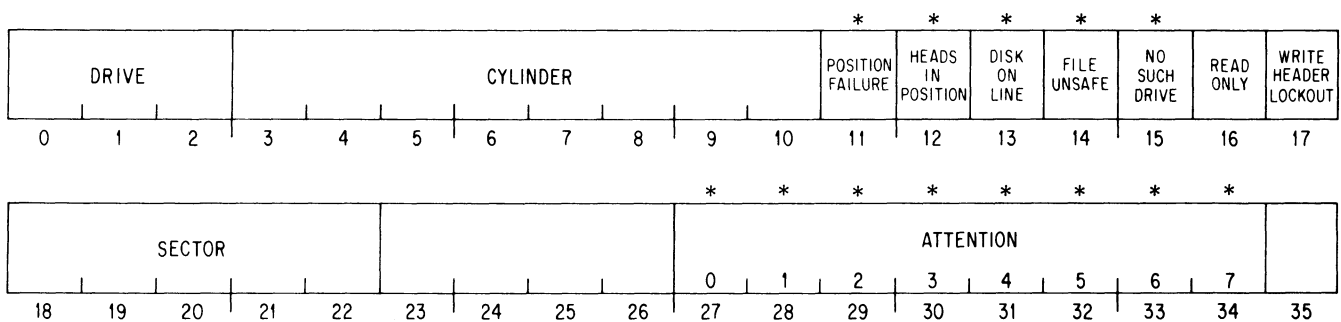
25 A 1 in this bit prevents the RP10 from stopping when Channel Data Parity Error sets.

If the function is Read Data, Write Data, or Write Format, the control checks as follows to determine whether the function can be executed properly: If Power Failure is set or the conditions supplied by this instruction result in the setting of No Such Drive, Not Ready, Illegal Write, Sector Address Error or Surface Address Error, the control sets Done, which in turn sets Interrupt, requesting an interrupt on the channel assigned by the last CONO DPC, and the control shuts down. If none of these error flags is set, the control sets Busy and waits for the data channel to connect.

DATAI DPC, Drive Data In, Disk Pack Control



Read the disk pack drive status into location E as shown.



Notes.

\*These bits cause interrupts.

The setting of Done or any Attention flag sets Interrupt, which requests an interrupt on the channel assigned by the last CONO DPC. Failure or success in positioning the heads in any drive sets the Attention flag for that drive. While Busy is set, Heads In Position or Disk On Line being clear or File Unsafe or No Such Drive being set in turn sets Done (for other conditions that set Done, see CONI).

- 0-2 The drive selected by the last legal DATAO.
- 3-10 The heads in the selected drive are positioned at or are being positioned to the cylinder specified by these bits.
- 11 The selected drive failed within 100 ms to position the heads at the cylinder specified by bits 3-10.
- 12 The heads in the selected drive are positioned at the cylinder specified by bits 3-10. If this bit is clear when a DATAO for a disk processing function is executed, the control sets Done and does not go into operation.
- 13 The selected drive is enabled by the operator and is at operating speed with a pack mounted, dust cover closed and heads loaded. If this bit is clear when a DATAO for a disk processing function is executed, the control sets Done and does not go into operation.
- 14 There is an electrical malfunction in the selected drive or the RP10 drive interface circuitry. If this bit is set when a DATAO for a disk processing function is executed, the control sets Done and does not go into operation.
- 15 There is no drive with the address specified by bits 0-2. If this bit is set when a DATAO for a disk processing function is executed, the control sets Done and does not go into operation.
- 16 The pack on the selected drive is protected against writing by the program.
- 17 The Write Format function cannot write on the disk (this flag is generated by a switch on the RP10).
- 18-22 The current position of the pack on the selected drive. A sector counter counts the twenty notches in the base plate of the pack. Thus for an RP02, bits 18-21 indicate the sector, and a 0 or 1 in bit 22 indicates the first or second half of the sector. For an RP01, bits 18-20 indicate which quarter of the sector is currently at the heads.
- 27-34 Bits in this field that are set indicate drives that have failed or succeeded in positioning their heads as required.

Note that bits 11 and 12 are both clear while the drive is attempting to position the heads.

This bit is clear if bit 11 is set. A drive that is off line only because of a position failure can be placed back on line by the Recalibrate function.

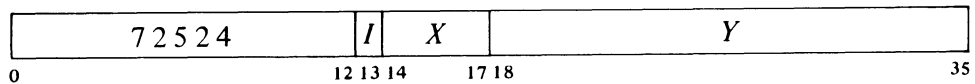
While Busy is set, bits 12 and 13 can be cleared and bits 14 and 15 set only by operator intervention or a physical or electrical malfunction, in which event the control terminates.

If inspection of the sector counter (which is guaranteed stable when read) shows the disk to be at the beginning of a sector, the program can start processing at the next sector. But if the disk is at the end of a sector, the program should not begin processing until the second sector after the one specified.

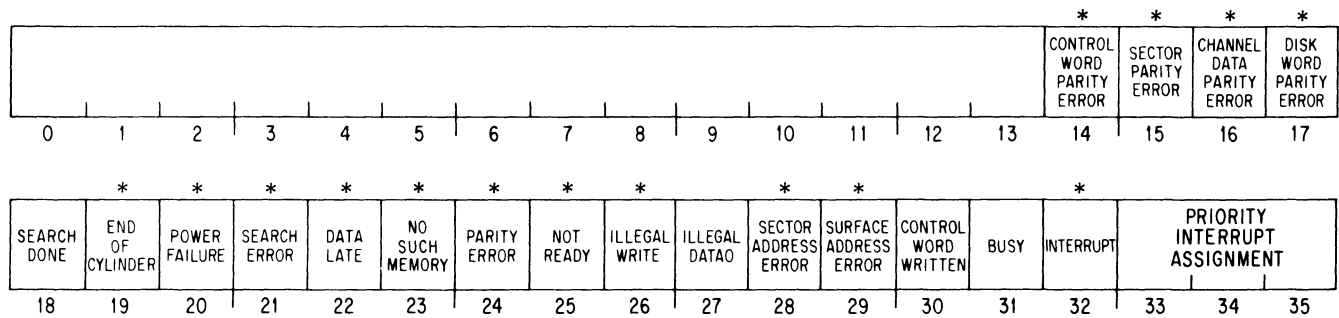
Note that bits 11 and 12 apply only to the selected drive, whereas these bits reflect conditions in all drives. Moreover, the program can clear any of these bits without affecting the associated drives.

Note that CONSZ and CONSO can test only bits 18-35. To test bits 14-17, the program must give a CONI DPC,AC and then use a test instruction [§2.8].

**CONI DPC, Conditions In, Disk Pack Control**



Read the status of the disk pack system into bits 14-35 of location *E* as shown.

*Notes.*

\*These bits cause interrupts.

DATAO clears Bits 14–17, 21–24 and 26–30.

The setting of Interrupt requests an interrupt on the assigned channel. This is the only flag that directly requests an interrupt. However, Interrupt is set by the setting of any Attention flag (see DATAI) and by the setting of Done, which is itself set by the setting of any error flag except Illegal DATAO (for other conditions that set Done, see DATAI).

In many cases the setting of a flag causes the control to terminate; this means that the control clears Busy, sets Done, requesting an interrupt on the assigned PI channel, and ceases operation. If the data channel has not already been released by the control or disconnected of its own accord, then termination releases the data channel.

- 14 The data channel discovered a parity error in a control word read from memory. When this bit sets, the channel disconnects from the RP10, which continues to the end of the current data field, either writing zeros or throwing away the data read, and then terminates.
- 15 The control read a sector with incorrect parity, and therefore terminated unless the last DATAO disabled the error stop.
- 16 The data channel discovered a parity error in a data word read from memory. Unless the last DATAO disabled the error stop, the setting of this bit causes the control to release the data channel, continue to the end of the current data field writing zeros, and then terminate.
- 17 The control read a word from the disk with incorrect parity, and therefore terminated unless the last DATAO disabled the error stop.
- 18 The control has found the addressed sector and a block transfer is in progress.
- 19 The data channel attempted to continue a block transfer beyond the end of the cylinder (*ie* the last sector of the last surface) and the control terminated.

Bit 18 means the sector address information matched that supplied by the DATAO, and the address word had correct parity.

- The control searches until it encounters the end of the track the third time; hence the search lasts from two to three revolutions.
- Some malfunctions associated with writing a particular sector (eg a bad header) set Search Error and terminate the function before the file unsafe condition can arise. This usually allows the program to continue using the disk pack despite the malfunction [see §7.8].
- While waiting for the memory, the control may set Data Late, which releases the channel so the memory error does not show up at all.
- During operation, bits 25, 26, 28 and 29 can be set only by hardware malfunction, in which case the control terminates.
- 20 The voltage levels in the RP10 are not within tolerance. If this condition is indicated when a DATAO is executed, the control sets Done and does not go into operation. If power fails during operation, the control completes the current data field and then terminates.
- 21 If no data has been transferred, a 1 in this bit indicates the control began a search but terminated because it was unable to find the addressed sector within a minimum of two revolutions.
- NOTE: The drive searches only on the addressed surface in the cylinder at which the heads are currently positioned.
- If inspection of the control word written by the data channel shows that some data has been transferred, bit 21 being set indicates that the control has terminated because of a hardware malfunction.
- 22 The data channel did not send data to or accept data from the RP10 when it was expected to. When this bit sets, the control releases the channel but continues to the end of the current data field, either writing zeros or throwing away the data read, and then terminates.
- 23 The memory addressed by the data channel did not respond within 100  $\mu$ s. When this bit sets, the channel disconnects from the RP10, which continues to the end of the current data field, either writing zeros or throwing away the data read, and then terminates.
- 24 Bit 14, 15, 16 or 17 is 1.
- 25 Heads In Position or Disk On Line is 0 or File Unsafe is 1 (DATAI bits 12–14). If this bit is set when a DATAO for a disk processing function is executed, the control sets Done and does not go into operation.
- 26 The last DATAO specified Write Data or Write Format but selected a protected pack (*ie* Read Only, DATAI bit 16, is set), so the control set Done and did not go into operation.
- 27 The program gave a DATAO while Busy was set.
- 28 The last DATAO specified a sector address greater than 11 for an RP02 or greater than 4 for an RP01, so the control set Done and did not go into operation.
- 29 The last DATAO specified a surface address greater than 23 for an RP02 or greater than 11 for an RP01, so the control set Done and did not go into operation.
- 30 The data channel has stored a control word in memory as requested by CONO bit 30 through the RP10.

- 31 The control is now performing a Read Data, Write Data or Write Format function.
- 32 Either Done or an Attention flag (DATAI bits 27–34) is set, and an interrupt is being requested on the assigned channel.

## 7.7 DISK PACK FUNCTIONS

Every legal DATAO clears the flags, and any function specified by a DATAO selects a drive, but only those functions that actually process the disk sectors set Busy and place the control in operation. A nonprocessing function may place a drive in operation, but the control remains free so the program can give other functions and can have a number of drives in operation simultaneously.

Select Drive and At Ease place neither the control nor the drive in operation. The first function simply selects a drive for status checking. At Ease clears the selected Attention flags.

Position Heads causes the selected drive to position its heads at the specified cylinder. When a drive completes a head-positioning command, it sets Heads In Position provided the drive is currently selected; similarly, if the drive fails within 100 ms to position its heads at the specified cylinder, it sets Position Failure provided it is the currently selected drive. In either event, the drive sets its Attention flag.

The program can give a DATAI to inspect the Attention flags for all drives as well as to check cylinder, sector and other status information for the currently selected drive, *ie* the drive specified by bits 0–2 of the word read by the DATAI. Besides setting appropriate flags, a drive that fails to position its heads correctly also goes off line. However, if a position failure is the only reason the drive is off line, the program can place it back on line by giving the Recalibrate function, which moves the heads to cylinder 0 independently of the previous position. Once a drive has been recalibrated, another attempt can be made to position the heads as desired.

The remaining functions all handle data through the DF10 and must therefore place the control in operation. With a DATAO that specifies any of these functions, the program must select a drive, cylinder and surface, and specify the initial control word address for the data channel. After loading the command information, the control checks the various conditions listed at the end of the DATAO instruction description to determine if it can perform as expected: if it cannot, it sets Done and shuts down. If the control can continue, it sets Busy and waits for the data channel to connect. While Busy is set, the control will accept no other DATAOs; if the program gives a DATAO, the control sets Illegal DATAO but otherwise ignores the instruction.

For Write Data and Read Data, the program must also specify the sector at which operations are to begin, and whether the control is to stop if parity errors are discovered in the data received from memory or the pack. Once the

data channel connects, the control begins the search for the specified initial sector, searching along the specified track in the cylinder at which the heads are already positioned. As soon as the control finds a sector whose address word matches the cylinder, surface and sector addresses supplied with the function, it starts transferring data to or from the track. If the control fails to find the desired sector by the third time it encounters the end of the track, it sets Search Error and terminates. Once transfers begin, the control processes data continuously, going through all the sectors from one surface to the next until it reaches the end of the cylinder, unless it is stopped by an error or the data channel retrieves a zero control word. Some errors cause immediate termination even in the middle of a sector, whereas for others the control releases the channel but continues to the end of the sector before terminating (see the descriptions of the flags read by a CONI). When the control processes part of a sector without the data channel, it simply continues the function it is already doing: in Write Data it writes zeros in the remaining data positions in the sector and then writes the parity word; in Read Data it reads the data and checks the parity. The program can stop the control at any time by giving a CONO DPC,20. If the data channel attempts to keep the control in operation after the last track in the cylinder is completed, the control sets End Of Cylinder and terminates.

For Write Format the program must specify whether the control is to stop if a parity error is discovered in the data received from memory. After the data channel connects, the control waits until it receives an index signal from the drive. Then the control writes the data supplied by the data channel on the addressed surface in the cylinder at which the heads are already positioned. This function processes one track, and the program must supply the data specified in §7.5 for the proper format to be written. The function terminates at the end of the track unless it is stopped prematurely by an error or the program.

Write Format cannot write in the header areas unless the write header lockout switch is off.

## 7.8 PROGRAMMING CONSIDERATIONS

Before giving a CONO or DATAO for the RP10, the program should check Busy to make sure the system is not already in use. To use the interrupt, give a CONO to assign the channel. Most of the bits read by a DATAI or CONI are indicated in the instruction descriptions as causing interrupts. However, only the Attention flags can request interrupts independently of drive selection, and then only when the control is not busy; when the program sets Busy, any interrupts being requested by the Attention flags go away but return when Busy subsequently clears. Among the other bits, those that reflect conditions in a given drive can affect the control only when the drive is selected. Furthermore, for all but the Attention flags, a condition can produce an interrupt only if said condition exists when the control either is attempting to begin a disk processing function or is already busy executing one; and the interrupt request does not occur until Busy clears, *ie* until Done sets. In most cases Done sets at the same time the condition is recognized, but in



some instances an error that occurs within a sector does not terminate the function (and hence request an interrupt) until the end of the sector. If a nonprocessing function selects a drive that does not exist or even attempts to position its heads, there is no interrupt — the function is simply ignored. Hence before attempting to use a given drive, the program should select it and at least check CONI bit 25 to make sure it is available. Giving a DATAI allows the program to check the individual drive conditions and also to determine whether the pack is protected, should writing be contemplated. If Disk On Line is clear, give the Recalibrate function just in case the drive was disabled by a position failure.

When a drive sets its Attention flag, the underlying condition (failure or success in positioning the heads) shows up in the DATAI only if the device is selected. Moreover, if the program subsequently clears the Attention flag, the condition remains and can show up in the DATAI until the drive receives a new position command.

Only a Position Heads function can reposition the heads. A read or write function does not move the heads even though it must specify a cylinder. This specification is for address checking only — the heads must already be positioned at the cylinder to be processed.

When the program is simply setting up one or more drives, an interrupt can be due only to an Attention flag. If an interrupt occurs when the control has been busy executing a disk processing function, the program should check status to determine whether the function was completed properly. If Search Error is set, check the control word written by the data channel. If no data has been transferred, the control has failed to find the desired sector. But if data has been processed, the flag is due to some malfunction, such as a bad header or an incorrect word count in the data field. While the control is writing, such malfunctions set Search Error and terminate the function before the file unsafe condition can arise and make the drive unavailable to the program. Should this situation occur, first repeat the entire function. If the second attempt fails, write one sector at a time and do not use the sector in which the malfunction occurred.

**Timing.** The maximum time required to move the heads from one cylinder to the next is 20 ms, and at most 80 ms are required for motion from one extremity to the other; average random positioning time is therefore at most 50 ms. Recalibration requires approximately 400 ms. These times are of course for the drive; a function that moves the heads takes about 6  $\mu$ s of control time.

#### CAUTION

Do not use a string of consecutive DATAOs to set up several packs. Following a DATAO DPC, that specifies Position Heads or Recalibrate, the program must give some other instruction (eg a no-op) before giving another DATAO DPC. If the program gives two consecutive DATAOs where the first moves the heads, the control will execute only that function supplied by the second.

If the disk is to be formatted, check that the operator has turned off the lockout switch (DATAI bit 17 is 0).

The control can execute At Ease followed immediately by Position Heads.

Search time can be minimized by giving a DATAI to determine the current pack position and starting operation at a sector that will be encountered soon.

At 2400 rpm the disk pack takes 25 ms per revolution. Although the time to traverse one complete RP02 sector is 2.46 ms, traversing the data field within a sector takes only 1.90 ms. Search time for a random data field after the data channel connects is 25.6 ms maximum, 13.1 ms average. During processing, word transfers occur every 14.8  $\mu$ s. The data channel has about 40  $\mu$ s between sectors.

Traverse time for one complete RP01 sector is 4.56 ms, of which 3.79 ms is taken up by the data field. Search time after the data channel connects is 26.2 ms maximum, 13.7 ms average. During processing, word transfers occur every 29.6  $\mu$ s. Between sectors the data channel has about 440  $\mu$ s.

## 7.9 OPERATION

The disk pack drive is illustrated on the next page. If main power is on, a pack loaded and the cover closed, pressing START turns on the drive. After a stabilization delay of about a minute, the drive loads the heads and positions them to cylinder 0. Pressing STOP unloads the heads and turns off the drive. The numbered lights indicate the last cylinder to which the drive was commanded to position its heads. FILE UNSAFE indicates an electrical malfunction in the drive or the control (pressing STOP resets this indicator if the condition has been cleared).

The remaining switches affect the drive only when it is not selected by the control. If the switches are manipulated while the drive is selected, the conditions they represent are applied to the drive when it is deselected at the beginning of the next function (deselection occurs even if the new function reselects the same drive). With the drive in proper operating order, pressing ENABLE places it on line, lighting the large indicator at the left center of the panel. Pressing READ-WRITE allows unrestricted use of the pack by the program; pressing READ-ONLY lights the associated indicator at the right and prevents the program from writing on the pack. Pressing DISABLE takes the unit off line.

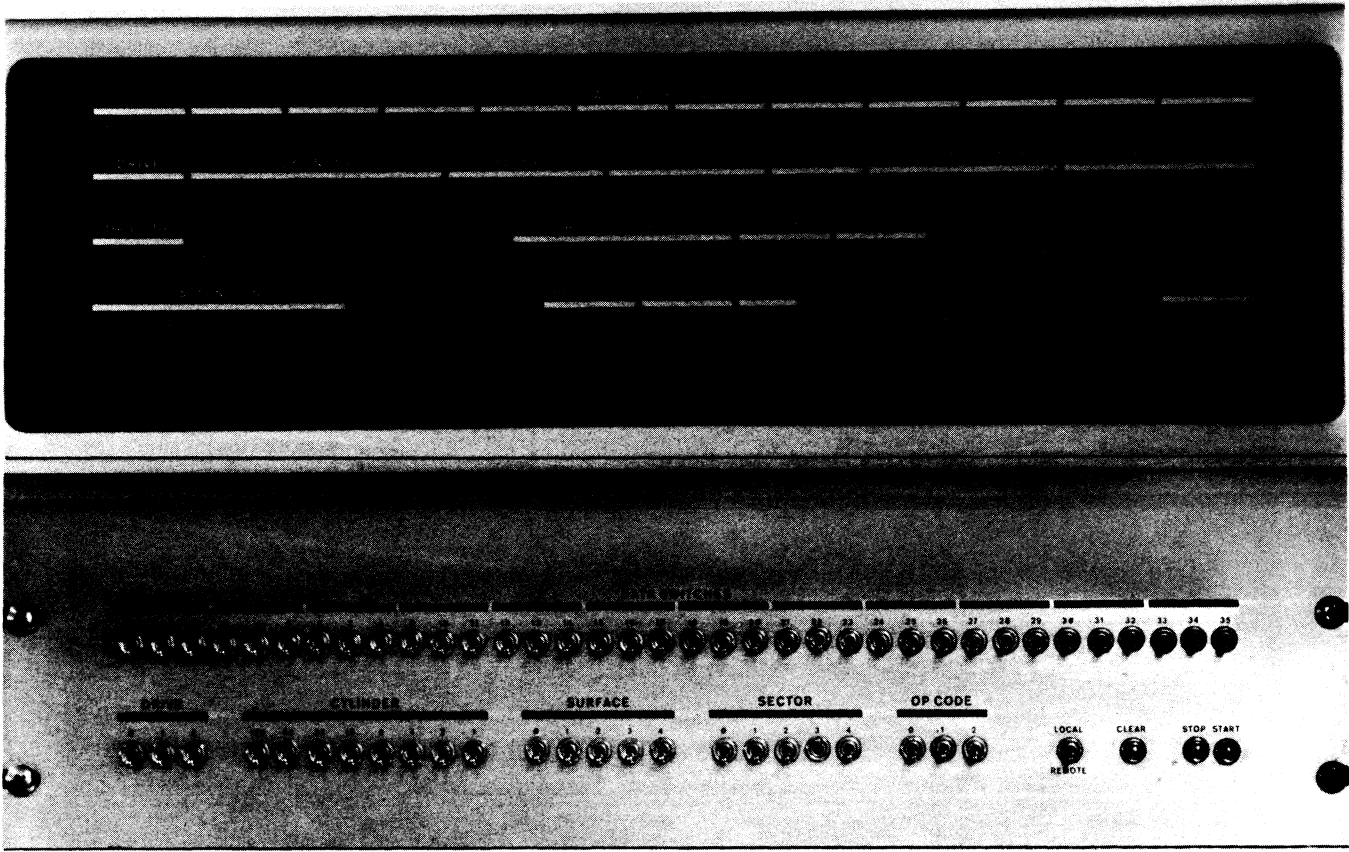
To load a pack, hold it by its handle and unscrew the bottom of the container a quarter turn (counterclockwise). Set the pack down over the spindle and turn it clockwise until it stops. Lift off the cover and close the door. To remove a pack, place the cover over it and turn it counterclockwise until a click is heard; the pack is now free of the spindle and is held tightly inside the cover, so it can be lifted out. Screw the bottom back onto the package.

Controls for the RP10 are located on a switch panel behind the doors just below the indicator panel. For online operation the single toggle switch at the right end of the panel must be set to REMOTE. To operate the system manually, switch the toggle to LOCAL, place the desired command data in the drive, cylinder, surface, sector and op code switches (data switches supply data for writing), and press START. The specified function will be repeated till STOP is pressed; for single cycle operation, hold STOP on while

Plastic plates with numeral cutouts are provided for insertion over the online indicator lamp. Thus when the drive is on line, the indicator can display the drive number (which is determined solely by the position of the drive cable at the control).



Disk Pack Drive



RP10 Switch and Indicator Panels

pressing START. Pressing CLEAR stops the control and clears the logic. The write header lockout is a toggle switch located between panels M and N at slots 13–16; this switch must be set to the right (ON) to prevent an inadvertent Write Format from disturbing the information in the headers.

The top row of lights on the indicator panel displays the contents of the assembly register. The two sets of six lights at the right end of the second row display the shift register, through which bytes are transmitted between the control and the drive, and a buffer that holds each byte stable for computing parity. The four sets of lights at the left display the last drive, cylinder, surface and sector addresses supplied by a DATAO or the switches. The remaining three lights in the row indicate designation errors for drive, surface and sector; these correspond to the flags No Such Drive, Surface Address Error, and Sector Address Error. The last function specified by a DATAO is indicated by the op code lights at the left end of the third row.

The remaining lights in the third row are for maintenance. The three sets of lights in the center display counters that keep track of the data words in each sector, the bytes in each word, and the bits in each byte.

**READ IM**      The control is executing the restricted function whose op code is 2.

**READ LPR**     The control is reading a parity word.

READ DATA	The control is reading a data field.
READ REF	The control is reading the synchronization area (refuse) at the beginning of a sector.
READ GAP	The control is reading the narrow gap between a header and a data field.
READ HDR	The control is reading a header.
WR HDR	The control is writing a header.
WR GAP	The control is writing the narrow gap between a header and a data field.
WR DATA	The control is writing a data field.
WR LPR	The control is writing a parity word.
ACTV	The control is waiting for the channel or is connected.
ACTV BUF	The control is waiting for the channel.
CHAN STTD	The channel has been seized (started).
INH	The control has released the channel.
CHAN PLS	The control has received the channel pulse.
PS FAIL	Power Failure.
LCL	The RP10 is under local control (off line).
GEN CLR	The RP10 is idle.

At the left end of the bottom row are the Attention flags for the various drives. The three SEARCH ERR lights keep track of the index pulses received during a search; light number 2, which is equivalent to the Search Error flag, goes on when the beginning of the track is encountered the third time (*ie* when the second complete revolution is finished). Of the flag and control indicators, BUSY, DONE and the PI assignment lights at the right are self-explanatory. The remaining lights are as follows.

DISK WDPE	Disk Word Parity Error.
DSPE	(Disk) Sector Parity Error.
CCPE	(Channel) Control Word Parity Error.
CDPE	Channel Data Parity Error.
SRCH	The control is searching for the addressed sector.
SRCH COMP	Search Done (complete).
DSPE STOP	Disable sector parity error stop (condition specified by DATAO bit 24).

CDPE STOP	Disable channel data parity error stop (condition specified by DATAO bit 25).
DWPE STOP	Disable disk word parity error stop (condition specified by DATAO bit 26).
PAR	This bit computes the parity function for each word.
PAR CONT	The control is reading or writing the parity (37th) bit of a word. When this light is on, PAR indicates the value of the parity bit being written or the parity of the word being read.
ILL WR	Illegal Write.
ILL COM	Illegal DATAO (command).
CWX COMP	Control Word Written (transfer complete).
WR EVEN	Write even parity (condition specified by DATAO bit 35).
OVER RUN	Data Late.
DISK NRDY	Not Ready.
NXM	No Such Memory (nonexistent memory).

# 8

## Data Communications

The equipment described here provides for the transfer of information in serial form between the computer and one or more other points, usually some distance away. Such equipment can simply connect to teletypewriters or other terminals located in a number of offices in a single plant, allowing engineers and other personnel to communicate directly with a centrally located DECSYSTEM-10; or make a large time-sharing facility available (through private lines or the standard telephone network) to many users located over a large geographical area; or allow high-speed communication between a large computation center and other computer installations located throughout the world.

Basically there are two types of serial communication. In a synchronous system, data is transmitted as a continuous bit stream, beginning with a pre-arranged special sequence through which the receiver synchronizes to the stream. In an asynchronous system, all information is transmitted in distinct characters bounded by start and stop bits. Running an asynchronous channel at its maximum rate does result in a continuous bit stream, but each data character is still separated from the next by stop and start bits, and the receiver synchronizes on each character separately. Communication at low speed (up to 300 bits per second) is invariably asynchronous, whereas high-speed transmission (generally above 2500 bits per second) is usually synchronous. Either technique is used for medium speeds (above 300 bits per second but within the capacity of a voice-band channel).

A DC10 or DC68A system services a number of asynchronous lines, and by providing conversion of data between parallel and serial forms, acts as a message concentrator and distributor. In the DC10, these operations are carried out largely by hardware, and the unit is connected directly to the IO bus for the transfer of data in single characters. The DC68A on the other hand includes a PDP-8/I computer, which is connected to the PDP-10 IO bus through a DA10 interface; hence, data conversion, channel scanning, and the transfer of data between the two computers (in characters or words) are determined entirely by the software. The DS10, which handles a single synchronous line, is connected directly to the IO bus and converts between full words and serial data. The DC75 is analogous to the DC68A for synchronous lines: it is based on a PDP-11, which communicates with the DECSYSTEM-10 central processor

The data rate may or may not be the same as the bit rate. For a Model 35 teletypewriter operating at the maximum speed of 10 characters per second (where each character consists of a start unit, eight data units (bits) and two stop units), the data rate is actually 80 bits per second, whereas the code element rate is 110 bits per second (*ie* 110 baud). On the other hand, a transmitter for a synchronous line using phase modulation might encode two bits of data per code element and achieve a data rate that is twice the code element rate. The program must necessarily be

concerned primarily with the data rate when that is the faster of the two. For a slower data rate, the situation depends on the type of equipment: when the software has to deal with the character structure including start and stop elements, the program must operate at the code element rate; when the hardware handles the characters, the programmer need be concerned only with the character or word rate (although the user may have to consider the line rate to select the proper clocks when the equipment is being set up).

Generally with a half-duplex channel (and sometimes an alternate-simplex channel), the originating station receives its own output data back as input as it is being transmitted, although some DEC equipment includes provision for the program to inhibit this echoing. Full-duplex operation with local copy uses a full-duplex channel in which each terminal receives its own output data back to provide a "local copy" of the data. To the program, this arrangement is indistinguishable from half-duplex.

In communication terminology, data signals for 1 and 0 are referred to as "mark" and "space". Control signals are regarded as being off or on, and the binary signals for supplying digits in automatic calling are invariably the numerals 1 and 0. The EIA standard for mark, off, or 1 is a voltage level  $\leq -5$  volts; space, on, or 0 is a voltage level  $\geq +5$  volts.

"Modem" is an abbreviation of "modulator-demodulator".

and memory via a DL10 interface. This interface connects not only to the IO bus, but also to the memory bus in such a way that the small computer actually uses part of DECsystem-10 memory as its own.

All DEC equipment is organized to handle each communication channel as two separate devices, so that data can be transmitted and received simultaneously. This is usually referred to as duplex or full-duplex operation. In general, the equipment can also be used to handle a half-duplex or alternate-simplex channel (wherein data can be transmitted in either direction but not both at once) or a simplex channel (one-way transmission only).

#### NOTE

For purposes of this discussion, a half-duplex channel is defined as one in which the data transmitted by any terminal is *ored* with the data transmitted by all other terminals and presented to all terminals as input data. Such a channel is usually implemented by relays in a floating current loop configuration. No formal protocol is required to reverse the direction of transmission, but if two terminals transmit simultaneously, all receive garbled information.

Alternate simplex is defined as a two-way circuit in which at any instant there exists only a one-way path, and a formal protocol must be established and followed to reverse the direction of transmission. A reverse channel (that operates at very low speed compared to the main channel) is sometimes provided to assist in the turn-around procedure.

Industry usage of these two terms is not consistent, but throughout this chapter the terms are used only as defined here.

A low-speed asynchronous channel may be used for communication with a Model 35 teletypewriter or equivalent terminal; local stations can be handled by simple transmitter and receiver modules utilizing 20 mA signals, and telegraph equipment is available for connecting to more distant stations utilizing 20–60 mA signals. Communication with any of the more recent terminal equipment such as the Model 37 and various visual displays requires EIA standard levels, *ie* signal levels as defined in EIA Standard RS-232-C and CCITT Recommendation V.24. Remote communication at speeds greater than 75 baud generally requires a modem, which also uses EIA standard levels. Such a modem is usually some type of Bell System data set or equivalent, which may have a connected or built-in telephone for voice communication and for manual answering and placing of calls for subsequent data transmission. Optional equipment is generally available in DEC communication systems for computer-controlled answering and placing of calls through the dial telephone network or private switched lines.

Industry standards term the modem as "Data Communications Equipment", whereas the computer and its associated equipment (including circuits for signal conversion between computer levels and EIA standard levels) is



termed "Data Terminal Equipment". At the other end of every channel there must be other Data Terminal Equipment, which can be another computer, teletypewriter, display, remote batch station, or other device. Indeed, the console teletypewriter is a local terminal connected via a single asynchronous channel to the IO bus through the console teletypewriter interface.

Except for communication with 110-baud teletypewriters or via telegraph lines, all DEC equipment described here uses levels that generally meet the EIA standard for transmission, reception, and control. The functions of the modem control signals used are generally as defined in RS-232-C or V.24, although there is some variation depending on the type of data set and the use of the system. §8.1 describes the basic control signals, discusses the characteristics of some of the more common data sets, and outlines the procedures for transmission, reception, automatic answering and automatic calling. The rest of the chapter describes the communication systems. Various terminals supplied by DEC are discussed in Chapter 9.

Particular interface equipment is also generally available to meet other specifications when required by local regulatory authorities.

## 8.1 COMMUNICATION SIGNALS AND PROCEDURES

There are six basic control signals used in interfacing to a modem, but all modems do not necessarily use all of them.

◆ *Data Terminal Ready (EIA circuit CD, CCITT circuit 108/2)*. The terminal equipment turns on this signal to connect the modem to the communication channel (place it in data mode) or to allow it to be connected by external means (eg manual calling, manual answering, or automatic calling). Usually the data set has a data pushbutton. For manual answering or originating, Data Terminal Ready must already be on, and the attendant places the set in data mode by pressing the data button until it lights. For automatic answering, the set must be in auto mode, and it goes into data mode when both Data Terminal Ready is on and a ringing signal is received. Turning off Data Terminal Ready disconnects the data set.

All DEC hardware is configured to use circuit 108/2; the hardware will work if connected to 108/1, but the user must take care of any regulatory requirements.

Generally, a data set may be wired in auto mode or have an auto button.

◆ *Request To Send (EIA circuit CA, CCITT circuit 105)*. The terminal equipment sends this signal to the local modem to turn on the transmitter and otherwise condition the modem for data transmission. On an alternate-simplex channel, Request To Send held on maintains the modem in transmit mode; otherwise, the modem is in receive mode. The signal is seldom used for full-duplex service.

Switching a data set from data to voice terminates the data link but holds the channel for voice communication until the attendant hangs up.

◆ *Data Set Ready (EIA circuit CC, CCITT circuit 107)*. The local data set sends this signal to the terminal equipment when it is connected to a communication channel and is capable of handling data.

This signal is generally not used by DEC terminal equipment.

◆ *Clear To Send (EIA circuit CB, CCITT circuit 106)*. The local modem returns this signal to the terminal equipment in response to Request To Send when it has established a connection with a remote terminal and is ready to transmit data.

Usually the receiver is held marking when Carrier Detected is off. This signal is also known as Received Line Signal Detector.

Generally, Ring Indicator follows the intermittent ringing signal, *ie* it goes on and off until the call is answered or abandoned. In some data sets, however, Ring Indicator remains on throughout the call.

Another signal, which is often used in switched network TWX service, is Restrain Detected. Suppose a 4-row station (such as a computer-data set combination) is connected to a 3-row station through an intermediate translator (the 4-row station operates at 100 or 150 words per minute, whereas the 3-row station operates at 60 words per minute). In this and other similar situations, Restrain Detected informs the computer that the translator buffer is full and cannot receive more characters. The computer must stop transmitting until Restrain Detected goes off or the translator will disconnect the data circuit.

The time required to answer a call, *ie* from Data Terminal Ready to Carrier Detected, is on the order of 3½–4 seconds.

In other words, the number is encoded in BCD.

◆ *Carrier Detected (EIA circuit CF, CCITT circuit 109)*. The local modem turns on this signal to the terminal equipment when it is receiving carrier from a remote terminal. On an alternate-simplex channel, Carrier Detected may be held off whenever Request To Send is on, or the data set may echo the transmitted data back to the receiver.

◆ *Ring Indicator (EIA circuit CE, CCITT circuit 125)*. The local modem sends this signal to the terminal equipment when it is receiving a ringing signal from the telephone or data network.

Occasionally, a data set may have all six of these signals available. Most sets use Data Set Ready, Clear To Send, and Carrier Detected, and a set for unswitched lines uses Request To Send, whereas a modem for switched lines uses Data Terminal Ready and Ring Indicator. With unswitched private lines (dedicated lines), Data Terminal Ready is often not used, and the modem is either always in data mode or is placed in data mode externally. To transmit data, the terminal sends Request To Send to the local modem and transmission can begin when the modem returns Clear To Send. Any data appearing on the incoming line can be received whenever Carrier Detected is on. A full-duplex modem often does not require Request To Send, and the transmitter is on whenever the set is in data mode. Using an associated hand set, the attendant can place and answer calls and put the modem in data mode manually provided Data Terminal Ready is on. The terminal can transmit when Clear To Send is on and receive when incoming carrier is present.

When the local terminal originates a call and data is transmitted as soon as Clear To Send comes on, the remote station must be capable of receiving data before turning on its own carrier. The CB-CF common option, which is available in most low-speed data sets and is required by most asynchronous DEC terminal equipment, delays the return of Clear To Send until incoming carrier is present, thus, reception by the remote station is guaranteed. Moreover, since the two signals come on simultaneously, the terminal equipment need check only Carrier Detected for both transmission and reception.

**Answering a Call.** If the local modem is left in automatic answering mode, the program can answer incoming calls placed through a switched network. A call to the local modem from a remote terminal is indicated by the presence of Ring Indicator; the local terminal equipment answers by turning on Data Terminal Ready. Establishment of the channel is indicated by Carrier Detected, at which time transmission and reception can begin. If Data Terminal Ready is left on, all incoming calls are answered automatically by the data set.

**Placing a Call.** For the computer to be able to place calls through a local modem requires that the system have an automatic calling unit (ACU) associated with the modem and an ACU control that is itself controlled by the program. The ACU supplies these signals: Power On, Data Line Occupied, Data Set Status (the modem is in data mode), Present Next Digit, and Abandon Call. Signals supplied to the ACU by the control are Call Request, Digit Present, and the telephone number, one digit at a time in binary form (four bits).

After selecting a line on which to place a call, the program can turn on Call Request if power is on in the ACU and the modem for the selected line is ready (*ie* it is not already in use, it is not in talk or test mode, and its Data Terminal Ready signal is on). In response to the request, the ACU takes control of the line from the data set and signals off-hook to the network exchange; on receipt of a dial tone, the ACU turns on Data Line Occupied and sends Present Next Digit to request the first digit of the telephone number. As the program supplies each digit, the control turns on Digit Present; on delivering the digit to the switched network, the ACU turns off Present Next Digit, at which time the control should turn off Digit Present and wait for the next digit request.

After dialing is complete, the program can let the ACU wait for the answer from the called station, provided the ACU can detect the answer tone. On detecting this signal, the ACU returns the line to the data set, which then goes into data mode and proceeds to establish the channel. Otherwise, the program can indicate the end of the number by responding to the last Present Next Digit with an end-of-number code (12, *ie* binary 1100). This returns the line to the data set and places it in data mode immediately; the modem itself must then detect the answer signal and establish the channel.

Although the ACU can be configured to stay with the data set throughout the call and terminate it by dropping Call Request, it is better to get the option that allows Call Request to be dropped as soon as the channel has been reconnected to the modem (*ie* as soon as the modem goes into data mode). Then the call can be continued by the modem control, and the call control can be used for placing calls on other lines. Either way, Call Request must remain off until Data Line Occupied goes off.

The ACU includes a timer that is started initially by the turnon of Call Request and is then restarted by each event in the call procedure to determine whether each subsequent event occurs within a reasonable time. If the timer runs out before the network returns a dial tone, before the ACU receives a digit from the program and sends it to the network, or before the data set goes into data mode, the ACU turns on Abandon Call to signal the program to hang up and try again. After dialing is complete, the timer is ordinarily turned off by the data set going into data mode. This arrangement assumes that Data Set Status coming on means that the call has been answered, which is the case if the ACU waits for the answer tone. If it is expected that the program will generally give an end-of-number code to let the data set wait for the answer, then the ACU should have the option that prevents Data Set Status from stopping the timer. In this case, the timer will always run out and can be taken as a signal that the program should check to see if the call has been answered.

**Terminating a Call.** An unswitched line remains permanently connected, but the modem control can turn off the transmitter by dropping Request To Send, and the data set can be switched manually from data to voice operation. A switched channel can be disconnected manually by changing to voice

Procedures outlined here are reasonably complete, but many details of timing and specific control interaction depend on the characteristics of the modem and the communication network itself. The user should consult the information supplied by the manufacturer of his data communication equipment and the common carrier supplying his communication channel or the manufacturer of his private network.

Typical automatic calling units are the Bell 801A for dial phones and the 801C for touch-tone phones. The 801C1 requires an end-of-number code; the 801C2 can detect the answer tone but will also respond to an end-of-number code.

After completing a message, the transmitting station should wait (perhaps a character time) before turning off the transmitter or terminating

the call, so that proper transmission of the final data over the line is ensured. Similarly, an end-of-transmission character keeps the receiving station from confusing the transients accompanying loss of carrier with additional data.

There is also an optional .4 second short space disconnect for use with 4-row TWX network terminals. Carrier Detected goes off on the order of 30–60 ms after carrier is lost. Some data sets have an automatic disconnect on loss of carrier.

No attempt is made here to describe all data sets nor even to describe completely the Bell sets included. There are other Bell sets besides these, and many other manufacturers as well. In the remainder of this chapter, typical data sets that can be used with the equipment under discussion are mentioned, but to determine whether any particular set is usable, the user should consult the manufacturer's manual and DEC's engineering representatives. For additional information on data sets, and in particular on the options and channel configurations recommended for use with DEC systems, refer to the interface and installation manual for the system and Chapter 5 of the *DEC Communications Equipment Handbook*.

mode and then hanging up the telephone. Under program control, a call is terminated by dropping Data Terminal Ready (or Call Request), generally for about 100 ms minimum, but in any event until Data Set Ready goes off. It is recommended that an end-of-transmission character (or better, character sequence) be transmitted before terminating so the remote station will also terminate. A modem can have an option to disconnect on receiving a long space (1.5 seconds). For this, the data set at the other end must be equipped with the send disconnect option, so that Data Terminal Ready going off causes the modem to transmit three seconds of space signal before disconnecting the line.

Besides terminating a call after completing transmission or receiving a prearranged amount of data, the modem control should disconnect the channel on receipt of an end-of-transmission character, loss of carrier, and when an incoming call is answered (in response to Ring Indicator) but no incoming carrier (or answer back code) appears within 10 seconds. This last situation may occur when a call is placed in error from a voice telephone to the local data set.

### Bell System Data Sets

The 103A, E, G, and H data sets operate at speeds up to 300 baud and are used typically for asynchronous communication over a switched network. These modems have connections for all the basic interface signals defined above except Request To Send. The 103A has the CB-CF common configuration and automatic answer capability, and it can be used in conjunction with a data auxiliary set 804B1 for voice communication. Available options are: send disconnect, long space disconnect, and operation with an automatic calling unit.

The 103E, G and H sets are different versions of the same modem; the G is the data set in a housing including a telephone, the H consists of data set, attendant's set, and hand set in three separate units, whereas the E is the basic modem without housing, power supply, or controls. Options include Ring Indicator only during ringing or throughout an incoming call, automatic answering, send disconnect, long or short space disconnect, CB-CF common, and disconnect on loss of carrier. These sets can also have an abort timer, which disconnects if the channel is not established within 8–16 seconds after the set answers a call.

A number of 103E modems can be mounted in a rack with additional control equipment as a 103E data station having additional features. Among the more important of these features are that automatic calling is possible and each modem has a Make Busy interface connection. The latter allows the modem control or the attendant to make an individual set appear busy to an incoming call; this is especially convenient when the sets are in a "hunting"

or “rotary” group, as it enables the telephone network to skip over a set that is out of order or will not answer for some reason. Since the 103E does not require Request To Send, the DEC terminal wiring can be modified to apply the Request To Send signal from the modem control to the Make Busy lead.

For use with asynchronous dedicated lines, the 103F provides communication at rates up to 300 baud for two-point or multipoint unswitched networks. Request To Send turns on the transmitter, and the set has the CB-CF common configuration. Provision for voice transmission must be made external to the set.

Typical data sets for asynchronous communication at medium speeds are the 202C and D. The 202C operates at 1200 bits per second over the telephone network and has a hand set in the unit. The 202D is used for dedicated lines and operates at 1000, 1400 or 1800 bits per second depending on the characteristics and configuration of the channel; alternate voice operation requires addition of the data auxiliary set 804A. These modems use all six interface signals: Data Terminal Ready connects the data set to the channel, and Request To Send turns on the transmitter. In alternate simplex operation, the latter signal controls the direction.

Typical data sets for medium speed synchronous operation are the 201A and B. The former is designed for transmission and reception at a fixed rate of 2000 bits per second over a switched voice frequency channel, public or private; the 201B is primarily for private line use and operates at 2400 bits per second. A similar data set is the 205, which operates at 2400 or 4800 bits per second.

High-speed communication is almost always over private wide-band lines. A typical set for an unswitched channel is the 301B, which operates at 40,800 bits per second. The various 303 type data sets are available for switched or unswitched applications. These modems are capable of both synchronous and asynchronous operation at several speeds depending on the bandwidth utilized. On a group channel (12 voice circuits), the speed is 50,000, 48,000 or 40,800 bits per second.

In most synchronous sets, transmission may be timed by a clock in the modem or an external clock supplied by the terminal equipment.

The 205 contains a very stable (and hence expensive) clock and is generally used only in special applications.

High asynchronous speeds are usually for facsimile operation and are not generally used with computers.

## 8.2 DATA COMMUNICATION SYSTEM DC68A

The DC68A handles transmission and reception of serial data over 128 full-duplex asynchronous communication lines at speeds up to 300 baud. Individual characters generally are three to eleven bits in length, including one start bit, one to nine data bits, and one or two stop bits. Transfer of data between the DC68A and DECsystem-10 memory is in characters or full words via a DA10 interface [§5.2] connected between the IO buses of the PDP-8/I and the PDP-10.

The heart of the system, as shown at the lower left in the illustration on the next page, is a DC08A serial line multiplexer and a PDP-8/I computer



containing a special DL8I data line interface. The computer usually contains a KA8I for driving a positive IO bus, which connects to the DC08 units as well as to other devices. This positive bus is also converted through a DW08A to provide the negative bus required by the DA10. Without a KA8I, the computer supplies a negative bus, which drives the DA10 directly but must be converted through a DW08B to drive the DC08A and other positive-bus devices. The bus supplies device selection bits and IO pulses to the in-out interfaces to implement program control over the various devices. The DC08A is no exception to this rule insofar as its clocks and control registers are concerned. However, the instructions that handle the transmission and reception of data through the multiplexer are executed by the DL8I; this interface actually takes control of the PDP-8/I processor, adding new hardware cycles whose timing differs from that of standard IO instructions.

For transmission, the PDP-10 supplies one or more messages in words or characters through the DA10. The PDP-8/I program divides the data into characters of the proper size and adds the necessary start and stop bits. It then transmits the characters over selected communication channels one bit at a time, interleaving bits from different messages and routing them to the proper channels through the multiplexer. In the opposite direction, bits received through the multiplexer from the various channels are sorted out and assembled into characters. On completing each character, the program strips off the start and stop bits and places the character in the part of memory where the corresponding message is being reconstructed. As each message is completed, the program sends it through the DA10, either one character at a time or with the characters assembled into PDP-10 words. Timing for transmitting bits and sampling the receiver inputs is provided by clocks in the DC08A. To guarantee accurate input sampling and to allow more even load distribution in transmission and reception, each clock has a frequency five times the code element rate. A single clock is standard, and is usually set at 550 Hz (110 baud). However, the clock can have any frequency up to 1500 Hz (300 baud), and the user can order up to three additional clocks, option DC08Y, for distribution of the channels into as many as four frequency groups (the distribution is software controlled).

The DC08A requires one M750 module for each pair of communication channels (full-duplex or otherwise) and can handle a maximum of 128 channels. From the M750s, the data lines are connected to various panels on which are mounted the circuits that convert between computer levels and the signals used on the communication channels. Communication with local terminals and data-only communication on remote lines is handled by a DC08B, which requires one W076 or BC01C circuit for each line. The W076 handles 20 mA signals and is used for connection to a local Model 33 or 35 teletypewriter (or equivalent). The BC01C connects to a local Model 37 or other terminal that uses EIA standard levels, but it can connect to a modem (such as the 103A) for data-only communication with a remote terminal. This circuit

This entire section is written from the point of view of the PDP-8/I rather than the PDP-10. The "computer" is the PDP-8/I, the "program" is the PDP-8/I program, "AC" is the PDP-8/I accumulator, and so forth.

For information on the PDP-8/I, including programming and operation, refer to the *Digital Small Computer Handbook*.

The fastest clock available is actually for 480 baud, but the objective is to handle a large number of low-speed lines. The number that can be handled depends both on the data characteristics of the channels and the efficiency of the software.

A local line is defined as one that is operating in an electrically clean environment and has a loop resistance of less than 40 ohms (eg a terminal connected by 800 feet of 24 gauge cable).

The BC01C can be used with the 103F by modifying it to apply the Data Terminal Ready signal to the Request To Send lead.

Use of this circuit with switched lines assumes that any problem, such as a wrong number or an inadvertent disconnect, is handled by an attendant.

Current is generally provided by a 793 power supply connected to the DC08C through an 893 fuse panel. The supply has a current capacity of 2.5 amperes at  $\pm 80$  volts.

Connection to the bus is not direct, but is made through the DC08A.

Neither of these data sets requires Request To Send. With a 103E data station, the connector can be modified so that Request To Send is applied to the Make Busy lead.

supplies no control signals except a fixed Data Terminal Ready and is principally for use with dedicated lines. It can be used, however, in a switched network wherein the local modem automatically answers all calls and outgoing calls are placed by hand; in this case, the local transmitter would always be on, and since there are no indicators, the program would have to monitor the line at all times to ensure that no incoming data is missed. The DC08B can handle a total of 48 lines distributed in any manner among the three types. To handle all 128 lines in this fashion requires three DC08B panels with the lines divided 48-48-32.

Telegraph communication requires the equipment shown at bottom right in the block diagram; this setup is also used for communication with nearby Model 35-type terminals that are too far away for a W076. The DC08CS telegraph relay panel handles 32 lines with one DC08CM telegraph line module, handling 20–60 mA signals, for each line pair. Connections from the modules to the telegraph lines are made at a DC08D terminator panel. The equipment shown in the drawing is for use with polar or neutral lines where the user supplies the line current. In this case, the lines connect to the terminator panel through a DC08EB monitor panel, which has rheostats for adjusting the send and receive currents. For neutral lines with an external line current supply, no monitor panel is needed. A complete complement of 128 telegraph channels requires four sets of the equipment shown.

Unlike the local and telegraph panels, the DC08F connects to the IO bus for program control of the communication hook-up. This panel has one DC08G for control and data connections to local modems for two channels. The modem control scans the channels, and on detecting a ring or a change in carrier status, signals the program and makes the number of the channel available to it. In-out instructions control Request To Send and Data Terminal Ready, to operate the transmitters and answer incoming calls. The DC08FX option (which is standard in a DECsystem-10) allows the program to step through the channels checking the carrier status of each. The basic DC08F can handle 64 lines; for an expanded system, a DC08FE can handle an additional 32 lines, and the full complement of 128 is reached by adding a DC08FF. Typical modems used are the 103A or a 103E data station (all modems must have the CB-CF common option).

The DC08F allows complete control over communication channels except for enabling the program to place calls without human assistance. To fulfill this need, a DC08H is required. This control is connected to the IO bus and may contain up to ten DC08J interfaces, each of which is connected to an 801-type automatic calling unit (ACU), which is, in turn, permanently connected to one of the modems controlled through the DC08F.

Some systems have the 689AG modem control [*upper left of the block diagram*] in place of the DC08F and DC08H units. The 689AG must be connected to a negative IO bus, and it requires one 689LM for each local modem (which must have the CB-CF common option). A ring or a change in carrier status signals the program via the interrupt, and the program must count



through and check the channels in groups of eight to find the line. In-out instructions control Request To Send and Data Terminal Ready. The 689AG handles 32 lines, and a second can be added for a maximum of 64. For placing calls, the first 689AG can contain up to four 689ACs, each of which controls an 801-type ACU, which is, in turn, permanently connected to one of the controlled modems.

The total complement of 128 lines can be distributed in groups of eight contiguous lines among these three classes: controlled remote, data-only or local, and telegraph – with the exception that the 689AG alternate modem control is limited to 64 lines. The order of the groups is entirely up to the user. Moreover, the lines in the second group can be distributed in any way among local dc, local EIA, and data-only remote.

Note that among the various panels for the different types of communication channels, only the modem control units (and their associated call controls) are connected to the IO bus and have instructions of their own; and these instructions are limited to control functions – they do not actually process any data. All data lines are connected to the M750s in the multiplexer and simply pass through the conversion circuits in the various panels. All transmission and receipt of data is handled by the program through the DC08A, regardless of whether the communication channel goes to an office down the hall or via satellite to Samarkand.

Jumpers in the M750s allow the user to select each input direct, inverted, or filtered.

### Data Multiplexing

The DC08A is very flexible in terms of the length and structure of the serial characters it handles, especially in transmission, but asynchronous communication invariably requires that every character begin with one bit time of space (0) for start and end with at least one bit time of mark (1) for stop. If the program wishes to restrict output characters to the PDP-8/I word size, then each such character can contain at most 11 data and stop bits combined. The hardware limits input sampling to 11 bits per character including start. As assembled or used in AC or a memory location, the start bit is at the right. The channel over which a single bit is received or transmitted at any given time is selected by the 7-bit line select register LSR; the program can load this register to select a line at random and can increment it to step through a set of consecutive lines.

Timing for transmission and reception is provided by a clock, which periodically interrupts the program. The program responds by sending bits to appropriate output lines and sampling appropriate input lines. If the clock frequency were the same as the bit rate, there would be no tolerance in line input timing, and the program could easily miss a bit (including the start bit) or attempt to read an entire character at the transitions between bits. The clock is, therefore, set at five times the bit rate. Then when the program detects a space that signals the beginning of a character, it can sample each

For each channel, the DC08A has a flip-flop that holds the bit on the output line.

A break is transmitted by sending a string of zero characters with no stop bits. Actually a suitable program could transmit anything acceptable to the remote terminal. Characters can be any length, with any length stops (even fractional). Indeed, the program could even convert a channel to synchronous operation by sending a continuous bit stream.

The TTI is unusually complex (it is equivalent to a small subroutine) so a flow chart is included with the detailed instruction description [see *below*].

Each line requires this set of four words:

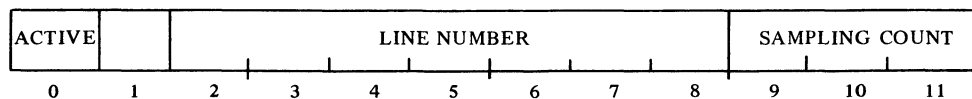
TTI  
LSW  
CAW  
JMS

bit at approximately the middle of the bit time. The DC08A can contain four clocks for four sets of lines at different frequencies. To distribute the lines into different frequency groups, the program simply determines which clock is interrupting and selects only the lines appropriate to that clock.

For output, the program takes the appropriate number of data bits from a message, sets up a character by appending start and stop bits, and stores it in a location associated with the line to be used. Then at every fifth interrupt, the output routine selects the line, puts the character in AC, right shifts one bit out to the selected line, and stores the shifted character back in memory. If a significant number of output lines are being used, the program should spread out the load by handling bits on perhaps a fifth of the lines at each clock interrupt and staggering the characters at different bit times, so that it will not have to set up new characters for all the lines within one bit time (*ie* within one set of five interrupts).

The user has no control over input except to choose the lines (he may simply ignore some). Hence, on every interrupt the program must carry out at least some operations for all accepted lines of the corresponding frequency. An inactive line must be checked on every pass in order to detect the beginning of an asynchronous character properly. After a character has begun, the input routine for the line samples it only at every fifth interrupt, but must be executed at every interrupt in order to guarantee sampling at the correct position in each bit time. After assembling a complete character, the program must then process it, *ie* take out its data bits, put them in the appropriate part of memory for assembling the message, and set up the input routine to look for the next character.

The instruction that checks a line and processes the input bits uses three memory locations, the first of which contains the input instruction TTI. The sequence for each line actually requires four locations, as it is assumed that the third location following the TTI contains a jump to a subroutine to process the character when it is assembled. The location following the TTI contains a line status word (LSW), which has the format shown here. The TTI



selects the line specified by LSW bits 2–8. When a character is detected, the TTI places a 1 in LSW bit 0. On subsequent passes, Active being set causes the TTI to keep a sampling count in LSW bits 9–11 so that it can sample the line in the middle of each bit; the count is from 0 to 4 in every bit time with sampling at 2 (every fifth interrupt).

At the count of 2, the TTI retrieves the character assembly word (CAW) from the third location. For the standard assembly procedure, the CAW is set up initially with a single 1 in a bit position such that the number of 0s at the right of it is two greater than the number of data bits in the expected input character. For a character with eight data bits, the CAW is initially 2000; for

seven bits it is 1000, and for five bits, 0200. To sample the input, the TTI shifts the CAW one place to the right, bringing the input bit in at the left. Shown here is the sequence of states for reception of a typical input character with eight data bits. As long as the character is incomplete, the TTI skips over the fourth location to continue the input routine. But when the initial 1 is shifted into the right most bit of the CAW, the routine continues by executing the fourth location, which should contain a JMS (or equivalent) to a subroutine to process the completed character. At completion, a character with *N* data bits is left in AC with the data in AC bits 1–*N*.

Note that the number of stop bits (even if fractional) is irrelevant, as the hardware regards the character as complete when the initial 1 is shifted all the way over. To sample two stop bits, the CAW is initialized with the 1 offset one place to the left from the standard position; to sample only the data bits in a character, the CAW is initialized with the 1 offset one place to the right. Sampling beyond the expected data in a character (*ie* what are expected to be stop bits) allows the program to recognize the reception of a break or long space.

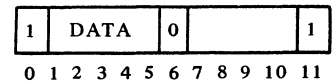
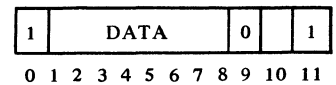
The sampling procedure, even for a large number of lines, is relatively quick. The processing of characters, however, can add significantly to the time required for a single pass. If characters are coincident on many lines, the time required for character processing offsets the sampling times for later lines from their usual positions with respect to the clock time. To provide a limit to this offset, the DC08A has a Line Hold flag for each channel and a 5-bit loading control counter LCC. At a clock interrupt, the input routine can load LCC with the (twos complement) negative of the maximum number of characters to be processed in that pass. Each character-processing subroutine should then increment the counter. When a TTI completes a character and discovers that LCC has been counted up to zero, it sets the Line Hold flag for the channel and stores the completed CAW back in the third location instead of going to the subroutine. In a subsequent pass, Line Hold being set causes the TTI to retrieve the completed CAW and go to the subroutine without sampling the line.

**Instructions.** The DL8I-DC08A combination has eight device codes, 40–47. There are no standard mnemonics defined for instructions that use these codes, but for convenience the mnemonics defined in the *X680* software are given here. Interrupts are requested by the setting of any clock flag.

Standard Assembly Procedure  
DATA = 01 001 011

INITIAL	010 000 000 000
START	001 000 000 000
1	100 100 000 000
1	110 010 000 000
0	011 001 000 000
1	101 100 100 000
0	010 110 010 000
0	001 011 001 000
1	100 101 100 100
0	010 010 110 010
STOP	101 001 011 001

With the standard assembly procedure, completed CAWs with eight and five data bits appear like this in AC:



**CK1ON**                      **Clock 1 On**                                      4.25  $\mu$ s                      6424

Clear Clock 1 Flag and enable clock 1 to set the flag, requesting an interrupt.

**CK1SKP**                      **Clock 1 Skip**    4.25  $\mu$ s                      6421

Skip the next instruction in sequence if Clock 1 Flag is set.

**CK1OFF**      **Clock 1 Off**      4.25  $\mu$ s      6422

Clear Clock 1 Flag and inhibit clock 1 from setting the flag.

Instructions analogous to the above are available for clocks 2, 3 and 4 as follows.

CK2ON	6434	CK3ON	6444	CK4ON	6454
CK2SKP	6431	CK3SKP	6441	CK4SKP	6451
CK2OFF	6432	CK3OFF	6442	CK4OFF	6452

To select a new line, the program must clear and load. This can be done in one instruction by ORing the codes, *ie* giving 6413.

**LSRCLR**      **LSR Clear**      4.25  $\mu$ s      6411

Clear the line select register.

**LSRLOD**      **LSR Load**      4.25  $\mu$ s      6412

Inclusive OR the contents of AC bits 5–11 into the line select register.

By ORing these codes, *ie* giving 6405, the program can do one instruction that first sends a bit, then selects the next consecutive line for the next bit. (Note that this is opposite the standard order implied by the codes.)

**TTO**      **Teletype Out**      1.5  $\mu$ s      6404

Send AC bit 11 out on the selected line. Shift AC right one place, bringing 0s into the link and bit 0; data shifted out of bit 11 is lost.

**LSRINC**      **LSR Increment**      1.5  $\mu$ s      6401

Increment the contents of the line select register by 1.

To produce a true transfer, this instruction must be preceded by CLA.

**LSRRED**      **LSR Read**      4.25  $\mu$ s      6414

Inclusive OR the contents of the line select register into AC bits 5–11.

**LCCCLR**      **LCC Clear**      4.25  $\mu$ s      6471

Clear the loading control counter.

To reset the counter, the program must clear and load. This can be done in one instruction by ORing the codes, *ie* giving 6473.

**LCCLOD**      **LCC Load**      4.25  $\mu$ s      6472

Inclusive OR the contents of AC bits 7–11 into the loading control counter.

**TTI                      Teletype In                      6402**

Retrieve the line status word LSW from the next location and select a line by loading LSW bits 2–8 into the line select register LSR. If Active (LSW bit 0) is clear or Line Hold is set, adjust Active to the state opposite that of the line, *ie* for a space (start) set Active, for a mark (stop) clear it. If Active is set and Line Hold is clear, add 1 to the sample count (LSW bits 9–11) or clear it if it has already reached 4. In any event, store the new LSW in the same location (the first after the TTI).

If the count is not 2, skip the next two locations and continue with the instruction in the fourth location beyond the TTI. If the count is 2, get the character assembly word CAW from the next location (the second beyond the TTI); and if Line Hold is clear, shift CAW right one place, reading the input from the selected line into CAW bit 0. Store the CAW (whether revised or not) back in the same location.

If the initial 1 has not yet been shifted into CAW bit 11 (*ie* if bit 11 is not now 1), skip the next location and continue with the instruction in the fourth location beyond the TTI. If bit 11 is 1 but the loading control counter LCC is zero, set Line Hold and skip one location to get the next instruction. On the other hand, if the assembly of the character is complete and the program has not yet processed all the characters it can on this pass (CAW bit 11 = 1 and LCC ≠ 0), load the CAW into AC, clear Line Hold, and continue with the instruction in the next location (the third beyond the TTI).

TTI takes 4.9  $\mu$ s if it retrieves the CAW from memory; otherwise, it takes only 3.4  $\mu$ s.

In practice these operations are usually separable. While the line is inactive, each iteration does nothing until a space is encountered, at which time Active sets. When a character is assembled but not processed, the next TTI clears Active (since a stop is being received).

Failure to process a character as soon as it is assembled sets Line Hold, so the count remains at 2 during subsequent iterations.

**LCCINC                      LCC Increment                      4.25  $\mu$ s                      6461**

Increment the contents of the loading control counter by 1.

**LCCRED                      LCC Read                      4.25  $\mu$ s                      6464**

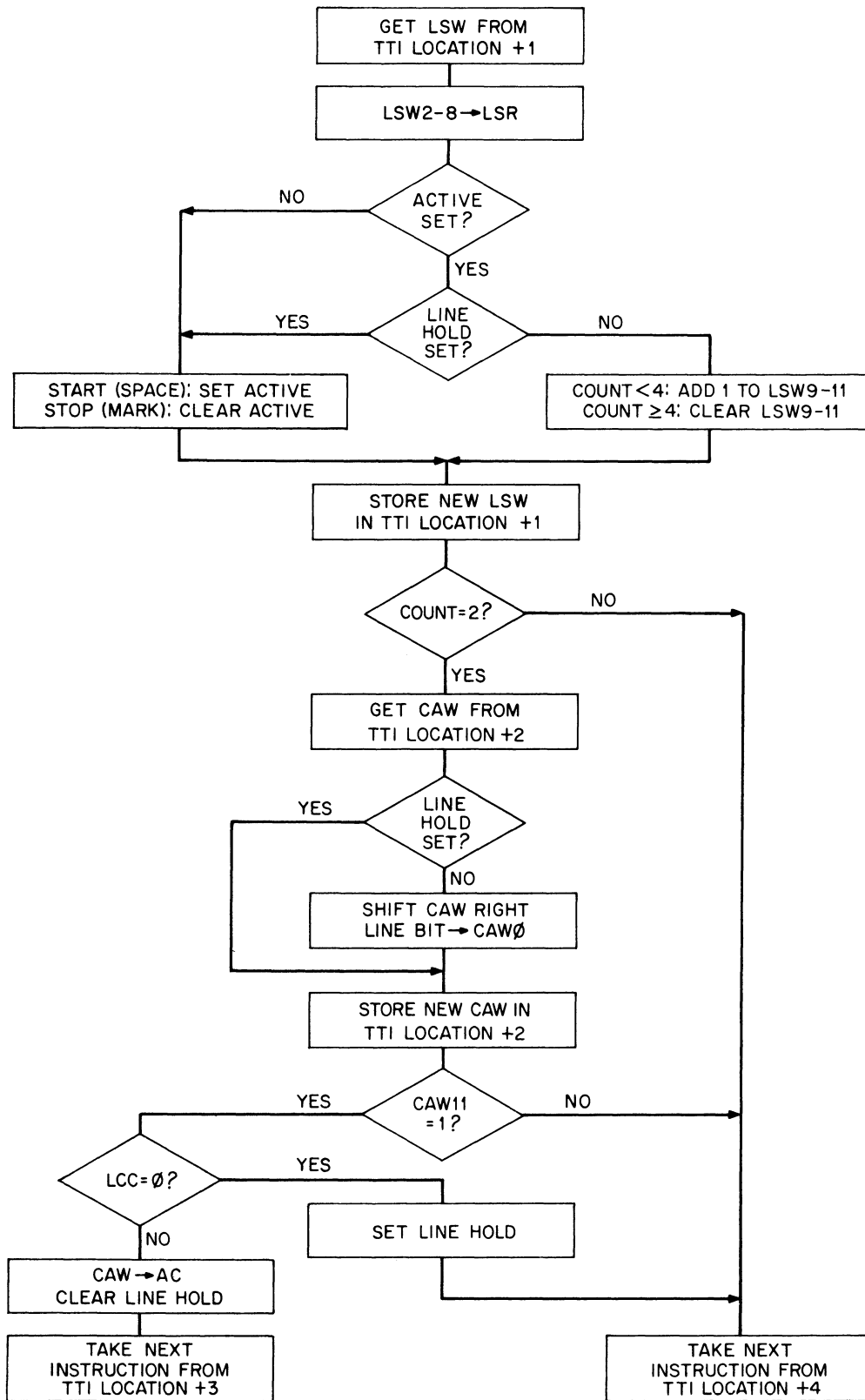
Inclusive OR the contents of the loading control counter into AC bits 7–11.

To produce a true transfer, this instruction must be preceded by CLA.

**Programming Considerations.** If several frequencies are used, interrupts from different clocks will occasionally occur simultaneously. The program should, therefore, check the clock flags in order of decreasing frequency so as to service the faster lines first. If frequencies differ greatly, the interrupt should be left on while the program is servicing slow lines so that it can stop to service fast lines as needed and then go back to complete a slow pass.

Because input cannot be distributed, the program should spread output servicing evenly over the five passes per bit time. In general, it is best to keep the lines in approximately the same timewise position in every pass. *Eg* as a character is finished, the output routine can jump to a subroutine to set up a new character for that line. But it is better for the program to go through the pass simply flagging every line on which character transmission is complete;

Remember that the interrupt is also being used for communication with the PDP-10 through the DA10.



TTI INSTRUCTION FLOW

then after the interrupt is dismissed, the main program can set up whatever new characters are required. The same procedure could be used for input, but the loading control counter allows the program to prevent reception from being offset too much. Generally, one can allow character processing on 20–40% of the lines in a given pass. However, even this upper limit could conceivably cause trouble for channels that use only one stop bit, especially if they are operating at maximum rate. To guard against this, the input routine should service such lines first, thus ensuring that characters from them will be processed in the same pass.

The setting of Line Hold does clear Active on the next pass and inhibit further counting, but the character-processing subroutine should nonetheless put a 0 in LSW bit 0 and allow the TTI to reactivate itself when a new character is encountered; otherwise, the program samples a second stop bit or a marking line as a new character. The subroutine should also clear the LSW sampling count, store the data bits of the character in the message being prepared for the PDP-10, indicate whether the message is complete, and if not, set up a new assembly word for the next character.

By using the second count out of five in each bit time, sampling occurs at the center  $\pm 10\%$ . This means that the input allows considerable tolerance in signal timing. Detailed consideration of timing for transmission and reception depends on the frequencies used, the efficiency of the program, and other factors.

### Modem Control DC08F

This unit has two 7-bit counters that scan through the channels checking for a Ring Indicator or a change in carrier status. Counting is done by a 2400 Hz clock, thus each modem is checked about 20 times per second. Although the same clock controls both counters, they function independently. If the unit detects the Ring Indicator on the line selected by the ring scanner or detects Carrier Change set on the line selected by the carrier scanner, it stops the corresponding counter (by disconnecting it from the clock) and sets the Ring Flag or Carrier Flag respectively, requesting an interrupt. The program can then read the number of the line on which the event occurred and take appropriate action. The DC08FX prevents the initialize pulse (generated by the computer start switch) from affecting the state of the modems and allows the program to step through the channels checking the carrier status of each with the automatic carrier scan disabled.

The DC08F has four device codes, 70–73. There are no standard mnemonics defined for instructions that use these codes, but for convenience the mnemonics defined in the *X680* software are given here. Interrupts are requested by the setting of the Ring Flag and Carrier Flag.

If a large number of channels use a single stop bit, the program can provide more leeway by sampling only the data bits. However, if the final data bit can be 0, the input routine must not give a TTI for the line again until the stop bit arrives, because the final data bit could be interpreted as the start of another character.

The carrier scanner determines whether the carrier has changed by checking the Carrier Change flag for the line; this flag is set whenever Carrier Detected goes on or off.

<b>MODLOD</b>	<b>Modem Load</b>	4.25 $\mu$ s	6704
---------------	-------------------	--------------	------

Enable Ring Flag and Carrier Flag to request interrupts and enable the decoding of AC bits 5–11 to select a modem for control signals.

<b>MODCLR</b>	<b>Modem Clear</b>	4.25 $\mu$ s	6712
---------------	--------------------	--------------	------

Disable Ring Flag and Carrier Flag from requesting interrupts and disable the decoding of AC bits 5–11 for selecting a modem. Disconnect the carrier scanner from the clock and increment the carrier counter by one.

By ORing these codes (6703) the program can do one instruction that checks the flag and reads the scanner. Note that the read (combined or not) must be preceded by CLA to produce a true transfer.

<b>RNGSKP</b>	<b>Ring Skip</b>	4.25 $\mu$ s	6701
---------------	------------------	--------------	------

Skip the next instruction in sequence if Ring Flag is set.

<b>RNGRED</b>	<b>Ring Scanner Read</b>	4.25 $\mu$ s	6702
---------------	--------------------------	--------------	------

Inclusive OR the number of the line selected by the ring scanner into AC bits 5–11.

<b>RNGCLR</b>	<b>Ring Scanner Clear</b>	4.25 $\mu$ s	6734
---------------	---------------------------	--------------	------

Clear the Ring Flag and enable the ring scanner (*ie* connect the ring counter to the clock).

By ORing these codes (6715) the program can do one instruction that checks the flag and reads the scanner. Note that the read (combined or not) must be preceded by CLA to produce a true transfer.

<b>CARSKP</b>	<b>Carrier Skip</b>	4.25 $\mu$ s	6711
---------------	---------------------	--------------	------

Skip the next instruction in sequence if Carrier Flag is set.

<b>CARRED</b>	<b>Carrier Scanner Read</b>	4.25 $\mu$ s	6714
---------------	-----------------------------	--------------	------

Inclusive OR the number of the line selected by the carrier scanner into AC bits 5–11 and the state of Carrier Detected on that line into AC bit 0.

<b>CARCLR</b>	<b>Carrier Scanner Clear</b>	4.25 $\mu$ s	6724
---------------	------------------------------	--------------	------

Clear the Carrier Flag, enable the carrier scanner (*ie* connect the carrier counter to the clock), and clear the Carrier Change flag for the line selected by AC bits 5–11.

<b>DTRLOD</b>	<b>Data Terminal Ready Load</b>	4.25 $\mu$ s	6731
---------------	---------------------------------	--------------	------

Turn on the Data Terminal Ready signal to the modem selected by AC bits 5–11.



**RTSLOD          Request To Send Load                  4.25  $\mu$ s          6732**

Turn on the Request To Send signal to the modem selected by AC bits 5–11.

Note that these codes can be combined so that the program can turn both signals on at once (6733) or both off at once (6723).

**DTRCLR          Data Terminal Ready Clear                  4.25  $\mu$ s          6721**

Turn off the Data Terminal Ready signal to the modem selected by AC bits 5–11.

With a 103E data station or similar setup, Request To Send can be used for Make Busy.

**RTSCLR          Request To Send Clear                  4.25  $\mu$ s          6722**

Turn off the Request To Send signal to the modem selected by AC bits 5–11.

The features of the DC08FX option are useful principally when something goes wrong in software or hardware. The program can determine exactly which channels have carrier and which do not without any interference in the state of the modems by restarting the program (although the initialize pulse does clear the scanner). MODCLR and CARRED have the same device code so they can be ored (*ie* 6716) to step the counter and read both it and the carrier status in a single instruction.

### Call Control DC08H

The DC08H has the hardware necessary for the program to place outgoing calls through ACUs (such as the Bell 801) associated with selected modems among those controlled by the DC08F. To implement the call procedure, the DC08H contains two flags and supplies two control signals to the ACU. To place a call, the program turns on Call Request. When the ACU is ready to receive each digit, it turns on Present Next Digit, which sets Digit Request; when the program has a digit ready, it sends the Digit Present signal. Call Status is set by various conditions involving the standard control signals [§8.1] sent by the ACU to the DC08H.

The call control has two device codes, 74 and 75. There are no standard mnemonics defined for instructions that use these codes, but mnemonics consistent with those used for the DC08F are given here. Interrupts are requested by the setting of Digit Request and Call Status.

Although the following eight instructions use the IOP pulses for timing, seven of them are derived from one device code by decoding instruction bits 9–11. Hence, in no case can the codes be ored to combine instruction operations.

**CALLOD          Call Load                                  4.25  $\mu$ s          6754**

Select the ACU specified by AC bits 8–11. The ten ACUs are addressed as 0–11 octal. An address greater than 11 selects no unit.

**CALREQ**      **Call Request**      4.25  $\mu$ s      6756

If power is on in the selected ACU and the associated modem is not already busy, turn on the Call Request signal to that ACU and start its timer. Otherwise (*ie* if Power On is off or Data Line Occupied is on), set Call Status and request an interrupt.

**CALSKP**      **Call Status Skip**      4.25  $\mu$ s      6751

Skip the next instruction in sequence if Call status is clear. This flag is set by the following events.

- ◆ Power On was off or Data Line Occupied was on from the selected ACU when the program gave a CALREQ.
- ◆ Power On went off or Data Set Status came on from the selected ACU while Call Request was on.
- ◆ Abandon Call came on from the selected ACU.

**CALRED**      **Call Status Read**      4.25  $\mu$ s      6755

Clear Call Status and inclusive OR the status of the selected ACU and associated modem into AC bits 0–4 as shown (a 1 in a bit indicates the signal is on).

POWER ON	DATA LINE OCCUPIED	ABANDON CALL	DATA SET STATUS	PRESENT NEXT DIGIT	
0	1	2	3	4	5

**DIGSKP**      **Digit Skip**      4.25  $\mu$ s      6753

Skip the next instruction in sequence if Digit Request is clear.

**DIGCLR**      **Digit Clear**      4.25  $\mu$ s      6741

Clear Digit Request.

**DIGLOD**      **Digit Load**      4.25  $\mu$ s      6757

Send the digit specified by AC bits 8–11 to the selected ACU, and if Call Request is on, turn on Digit Present.

The decimal digits 0–9 are specified by codes 0000–1001; 12 (1100) is the end-of-number code; other configurations of AC bits 8–11 are not used.

To produce a true transfer, this instruction must be preceded by CLA.

**CALCLR**            **Call Clear**                            4.25  $\mu$ s            6752

Turn off Call Request and Digit Present.

To place a call, the program gives CALLOD to select an ACU unit (and therefore a modem and associated communication channel) and CALREQ to turn on Call Request to the selected unit. If either Power On is off or Data Line Occupied is on for the selected unit, the CALREQ fails to turn on Call Request and instead sets Call Status, requesting an interrupt (the program can use CALSKP and CALRED to determine the cause of the interrupt). Otherwise, the ACU starts its timer and waits for a dial tone; reception of the tone turns on Data Line Occupied. As the ACU is ready for each digit it turns on Present Next Digit, which sets Digit Request, requesting an interrupt. The program should respond using DIGSKP, DIGCLR and DIGLOD to provide the digit, clear Digit Request and turn on Digit Present. When the ACU accepts the digit, it turns off Present Next Digit, turning off Digit Present, and restarts the timer. Should the timer run out while waiting for some necessary event in the call procedure, the ACU turns on Abandon Call, which in turn sets Call Status, requesting an interrupt.

After the last digit has been processed, the ACU again turns on Present Next Digit, and the response by the program depends on the particular setup. The program may simply clear Digit Request and let the ACU wait for the call to be answered, at which time the channel is returned to the modem; or the program may have the option of or need to give the end-of-number code, which turns the line over to the modem immediately and the modem waits for the answer. When the modem regains the line, it goes into data mode and turns on Data Set Status, which in turn sets Call Status, requesting an interrupt. If the timer runs out, either before Data Set Status comes on or because the modem going into data mode does not stop it, the ACU generates Abandon Call, setting Call Status and requesting an interrupt.

After Data Set Status comes on, action again depends on the way the ACU is set up. The program can give CALCLR to drop Call Request while the modem waits or the call continues, and the program may use another ACU, although it must eventually terminate the call through the modem control by turning off Data Terminal Ready. Without this option, the program must stay with the call until it is over and terminate it by dropping Call Request.

### 689AG: Part I, Modem Control

The modem control section of the 689AG has two flags that detect ringing signals and carrier changes. The turnon of Ring Indicator on any channel sets Ring Flag; Carrier Detected turning on or off on any channel sets Carrier Flag. The setting of either flag requests an interrupt. The program responds

Note that in no case is Call Status set simply by the passive existence of a condition: rather, it is set always by a specific event. CALREQ being given while Data Line Occupied is on sets Call Status, but Data Line Occupied coming on later (as it must when a dial tone is received) does not. Abandon Call turning on sets Call Status, but if the program chooses to ignore the signal, it simply clears the flag and there is no further interrupt even though Abandon Call may remain on (there would be another interrupt were Abandon Call to go off and then come on again).

If the program gives an end-of-number code but continues to select the same ACU, it can use the timer interrupt as a signal to check that the call has been answered. If the program switches to some other ACU, reselecting the original one later will trigger the interrupt if the timer has already run out.



**RNG Ring Clear** 4.25  $\mu$ s 6742

Clear Ring Flag.

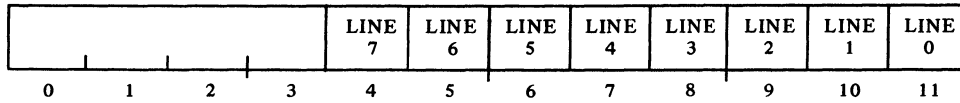
**CARSKP Carrier Skip** 4.25  $\mu$ s 6711

Skip the next instruction in sequence if Carrier Flag is set.

**CARRED Carrier Read** 4.25  $\mu$ s 6714

Inclusive OR the carrier status of the lines in the presently selected group into AC bits 4–11 as shown (a 1 in a bit indicates that Carrier Detected is on from the specified line).

To produce a true transfer, this instruction must be preceded by CLA.

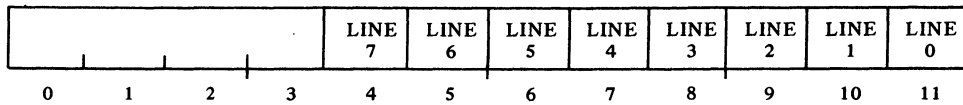


**CARCLR Carrier Clear** 4.25  $\mu$ s 6741

Clear Carrier Flag.

**DTRL0D Data Terminal Ready Load** 4.25  $\mu$ s 6722

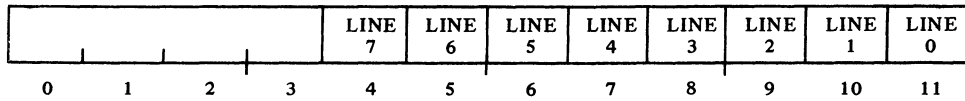
In the presently selected group, turn on the Data Terminal Ready signals to the modems selected by 1s in AC bits 4–11 as shown.



**RTSL0D Request To Send Load** 4.25  $\mu$ s 6732

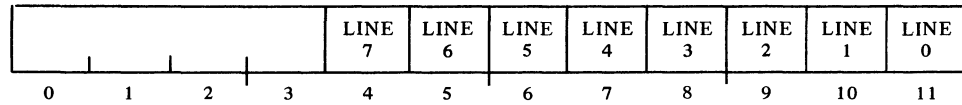
In the presently selected group, turn on the Request To Send signals to the modems selected by 1s in AC bits 4–11 as shown.

With a 103E data station or similar setup, Request To Send can be used for Make Busy.



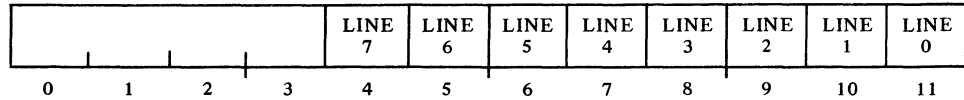
**DTRCLR Data Terminal Ready Clear** 4.25  $\mu$ s 6724

In the presently selected group, turn off the Data Terminal Ready signals to the modems selected by 1s in AC bits 4–11 as shown.



**RTSCLR**            **Request To Send Clear**            4.25  $\mu$ s            6734

In the presently selected group, turn off the Request To Send signals to the modems selected by 1s in AC bits 4–11.



Note that the program cannot select an individual line at random. The lines are divided into groups of eight and to select any particular line the program must clear the group counter and then increment it to the group containing the desired line. Within a given group, the program can read the ring or carrier status of all lines simultaneously and can handle a given control operation (turning Data Terminal Ready or Request To Send on or off) on any desired lines simultaneously.

A malfunction that keeps one Ring Indicator on disables all of them by preventing any ring interrupts. This should not be regarded as extremely unlikely, as simply removing a data set cable holds on the Ring Indicator. Thus, on a low priority basis the program should occasionally check the ring status to see if any Ring Indicator is on that did not interrupt.

The flags are set only by specific events – the turnon of a ring signal and a change in carrier. Hence in response to an interrupt, the program should check all channels in case more than one was ringing or experienced a carrier change.

### 689AG: Part II, Call Control

The call control section of the 689AG has the hardware necessary for the program to place calls through automatic calling units (ACU) associated with modems selected from among those the 689AG controls. To implement the call procedure, the call control contains two flags and supplies two control signals to the ACU. To place a call, the program turns on Call Request. When the ACU is ready to receive each digit it sets Digit Request, and when the program has a digit ready it sends the Digit Present signal. The program must give the end-of-number code, either to end the dialing sequence or later to turn off Call Request. Turnon of the Abandon Call signal sets the Incomplete Call flag.

The call control has three device codes, 75–77. There are no standard mnemonics defined for instructions that use these codes, but mnemonics consistent with those used for the modem control are given here. Interrupts are requested by the setting of Digit Request and Incomplete Call.

<b>CALLOD</b>	<b>Call Load</b>	<b>4.25 <math>\mu</math>s</b>	<b>6752</b>
---------------	------------------	-------------------------------	-------------

Select the ACU specified by AC bits 10–11. The four ACUs are addressed as 0–3 octal.

<b>CALSKP</b>	<b>Call Skip</b>	<b>4.25 <math>\mu</math>s</b>	<b>6761</b>
---------------	------------------	-------------------------------	-------------

Skip the next instruction in sequence if Power On is on and Data Line Occupied is off from the selected ACU. ▲

<b>CALREQ</b>	<b>Call Request</b>	<b>4.25 <math>\mu</math>s</b>	<b>6764</b>
---------------	---------------------	-------------------------------	-------------

If power is on in the selected ACU and the associated modem is not already busy, turn on the Call Request signal to that ACU and start its timer.

<b>DIGSKP</b>	<b>Digit Skip</b>	<b>4.25 <math>\mu</math>s</b>	<b>6751</b>
---------------	-------------------	-------------------------------	-------------

Skip the next instruction in sequence if Digit Request is clear.

<b>DIGCLR</b>	<b>Digit Clear</b>	<b>4.25 <math>\mu</math>s</b>	<b>6762</b>
---------------	--------------------	-------------------------------	-------------

Clear Digit Request.

<b>DIGLOD</b>	<b>Digit Load</b>	<b>4.25 <math>\mu</math>s</b>	<b>6754</b>
---------------	-------------------	-------------------------------	-------------

Send the digit specified by AC bits 5–8 to the selected ACU, and if Call Request is on, turn on Digit Present.

The decimal digits 0–9 are specified by codes 0000–1001; 12 (1100) is the end-of-number code; other configurations of AC bits 5–8 are not used.

<b>INCSKP</b>	<b>Incomplete Call Skip</b>	<b>4.25 <math>\mu</math>s</b>	<b>6771</b>
---------------	-----------------------------	-------------------------------	-------------

Skip the next instruction in sequence if Incomplete Call is clear.

<b>INCCLR</b>	<b>Incomplete Call Clear</b>	<b>4.25 <math>\mu</math>s</b>	<b>6772</b>
---------------	------------------------------	-------------------------------	-------------

Clear Incomplete Call.

To place a call, the program gives CALLOD to select an ACU (and therefore a modem and associated communication channel) and CALSKP to determine whether the selected unit is ready, *ie* power is on and the line is free. If so, CALREQ turns on Call Request for the selected unit, which starts

The dial tone can be sensed by CALSKP, as Data Line Occupied comes on at that time.

At each step in the procedure, the program should give both DIGSKP and INCSKP to distinguish between a digit request and an incomplete call.

If the program chooses to ignore Incomplete Call, it can simply give INCCLR and there is no further interrupt even though Abandon Call may remain on (there would be another interrupt were Abandon Call to go off and then come on again).

If the program continues to select the same ACU after giving an end-of-number code, it can use the timer interrupt as a signal to check that the call has been answered. If the program switches to some other ACU, reselecting the original one later will trigger the interrupt if the timer has already run out.

its timer and waits for a dial tone. As the ACU is ready for each digit, it turns on Present Next Digit, which sets Digit Request, requesting an interrupt. The program should respond using DIGSKP, DIGCLR, and DIGLOD to provide the digit, clear Digit Request and turn on Digit Present. When the ACU processes the digit, it turns off Present Next Digit, turning off Digit Present, and restarts the timer. If the timer runs out while waiting for some necessary event in the call procedure, the ACU turns on Abandon Call, which in turn sets Incomplete Call, requesting an interrupt.

After the last digit has been processed and the ACU has again turned on Present Next Digit, the program must at some point give the end-of-number code, as the only condition that turns off Call Request is Present Next Digit going off while that code is being held in the call control. Suppose the data set can detect the answer tone and can continue the call under modem control (*ie* without Call Request). In this case, the program can clear Digit Request and give the end-of-number code immediately; this causes the ACU to turn the line over to the data set, and the turnoff of Present Next Digit drops both Digit Present and Call Request. The program can then use another ACU while the modem waits for the answer tone and establishes the channel. If the timer is not stopped by Data Set Status, it will run out, causing the ACU to generate Abandon Call; this sets Incomplete Call to request an interrupt if the ACU is still selected.

If the data set cannot detect the answer tone but can do without Call Request, the program should clear Digit Request, wait until the call has been answered and the line turned over to the data set, and then give the end-of-number code anyway. This turns on Digit Present, which will then go off along with Call Request when Present Next Digit goes off. If the timer runs out before Data Set Status comes on, the ACU generates Abandon Call, setting Incomplete Call and requesting an interrupt.

If the data set is configured to terminate when Call Request goes off, the ACU must wait for the answer signal and the program terminates the call by giving the end-of-number code.

### 8.3 DATA LINE SCANNER DC10

The DC10 is a line scanning-multiplexing system for handling transmission and reception of serial data over asynchronous communication channels. The user can select the characteristics of each channel as follows:

Which channel types can actually be selected depends on the communication facilities available. The user can select three frequencies arbitrarily — the rest are dependent on them [see below].

Channel type: full-duplex, full-duplex with local copy, half-duplex, alternate simplex, simplex-receive only, simplex-transmit only

Bit rate: any one of three selected for the 32-line group out of nine available to the system (maximum rate, 100,000 bits per second)

Data bits per character: five or eight

Stop units (bit times) per character: 1, 1.5, 2 (start is one unit)



The choice among the above, once made, is fixed in the hardware so that all transmission and reception over a given channel has the characteristics selected initially. Transfer of data between the DC10 and DECsystem-10 main memory is over the PDP-10 IO bus in single characters, each containing the number of data bits appropriate to the channel.

The entire system can have 64 lines. A given line can be a communication channel but need not be — some lines can be paths for control signals to operate data sets, *ie* a program-controlled modem requires two lines, one for data and one for control. A maximum system can therefore have as many as 64 data lines or as few as 32 data lines with the other 32 lines used for modem control.

The basic unit in the system is a DC10A, which contains the clocks, scanning facilities, and bus interface. Each set of eight data lines requires a DC10B, which contains a transmitter and receiver for each line. These circuits can handle either EIA standard levels or local 20 mA dc signals. To drive a telegraph line, the DC10B transmitter-receiver must be connected to the line via a DC10C telegraph relay assembly. This latter unit also handles eight lines, but it need not be connected to a single DC10B — it can drive any eight lines selected from among the various DC10Bs.

Program control of a data set on a data line through a DC10B requires a modem control line through a DC10E. Each DC10E provides control for eight modems and automatic calling for two of them. There is no necessary correspondence between any DC10E and DC10B; the DC10E lines can be used to control modems associated with any eight data lines connected to the DC10Bs, and any two of these modems can have automatic calling units. In terms of the line structure, a DC10E appears in place of a DC10B. The total number of DC10Bs and DC10Es combined is therefore eight, of which at most four can be DC10Es, as 32 modem controls require 32 data lines.

**DC10A.** This unit contains three clocks that generate nine signaling frequencies. Of these, any three can be made available to the first 32 lines and any three to the second 32 lines (hence, up to six can be used altogether). The user can select three frequencies arbitrarily, and the rest, although not uniquely determined by the three selected, are dependent on them. Within limits, the functional dependency is adjusted to best suit the user's requirements. If no frequencies are specified, the equipment is delivered with clocks for the standard or typical bit rates of Model 28, 33, 35, and 37 teletypewriters and 103 and 202 series Bell data sets (*ie* 75, 110, 150, 300, 600, 1200, and 2400 Hz).

The DC10A also contains a scanner that searches through all the lines and stops to interrupt the program whenever it encounters a line with a flag set. The scanner supplies the number of the line at which it stopped and indicates whether the stop was due to completion of character transmission or reception so the program can respond appropriately. Transmitter and receiver interrupts occur only on data lines, but the scanner checks all lines and also requests an interrupt on encountering a modem control that requires program attention.

Actually, the DC10 can be used solely for modem control with the data going through some other device. Thus, a system with eight DC10Es to handle 64 modem control lines is theoretically possible.

Given three base frequencies  $f_1$ ,  $f_2$ , and  $f_3$ , the nine derived frequencies are  $f_1$ ,  $16f_1$ ,  $f_2$ ,  $16f_2$ ,  $f_3$  and  $2^n f_3$ , where  $n = 1, 2, 3, 4$ . The standard base frequencies supplied respectively are 110, 150, and 75 Hz.

For complete information on installing jumpers, connecting lines, etc., refer to the *DC10 Maintenance Manual*.

A local line is defined as one that is operating in an electrically clean environment and has a loop resistance of less than 50 ohms (eg a terminal connected by 1000 feet of 24 gauge cable).

Current for any relay-buffered line may be supplied externally or by a DC10D power supply, which has a capacity of 2 amperes at  $\pm 125$  volts. A single supply can handle 32 half-duplex or 16 full-duplex lines at 60 mA, or 96 half-duplex or 48 full-duplex lines at 20 mA.

**DC10B.** For each of the eight lines in a DC10B, the user must install jumpers to select the line speed (among the three available), the number of data bits per character, and the number of stop units. The equipment is delivered with the lines configured for full-duplex or alternate-simplex operation, and jumpers must be added to convert any line to full-duplex with local copy or half-duplex (the latter also requires that the DC10C be used). Simplex operation requires no jumpers as it simply implies that either the transmitter or receiver for a given line is not used. Each line must also be connected to the proper connector pins (for EIA or dc signals).

The EIA connections can be used for a Model 37 teletypewriter or similar terminal, a data set controlled through a DC10E, or a data set controlled manually. The dc signals can be used for connection to a relay assembly DC10C for telegraph lines or half-duplex operation, or directly to a local Model 33 or 35 terminal for full-duplex operation with or without local copy. For full-duplex with local copy, the transmitter will not go on until the receiver is idle so as not to garble copy at the teleprinter. The transmitter also waits until the receiver is idle in half-duplex operation.

For output, the program supplies each data character to the desired transmitter, which adds the start and stop bits and sends the character out serially over the line. Each serial character coming in over a line is assembled by a receiver, which strips off the start and stop bits and supplies the data for reading by the program. Completion of transmission or reception of a character sets a corresponding flag for the line. When the scanner encounters the flag, it stops scanning and requests an interrupt. The program responds with a DATAI that reads the number of the line, and if the receiver flag is set, reads the character and clears the flag. If the transmitter flag is set, the program uses the line number to determine what character to send next, and then gives a DATAO to clear the transmitter flag and supply the character for transmission on the line specified by the scanner. In either case, when the interrupt has been serviced, the scanner continues to search for the next flag. The program can select any random line to start transmission, but for further transmission and all reception, the program acts only at an interrupt.

**DC10C.** This unit contains send and receive relays for any eight lines connected to DC10Bs. It can be used to implement a half-duplex channel, to communicate over long dc telegraph lines (up to 22 miles (36 kilometers) of 24 gauge cable), or to connect a Model 35 terminal that is too far away or is in an electrically noisy environment. The unit accommodates polar and neutral circuits of 20–60 mA and has controls for line resistance compensation and send and receive relay bias. Use of the DC10C does not affect programming except in that signaling speed is limited to 300 baud, and in half-duplex operation any character transmitted is also received.

**DC10E.** This unit provides program control over data sets on eight of the DC10B data lines. The program can control one signal to each modem and can monitor two signals from it. The output signal is usually connected to

the Data Terminal Ready lead but can be used for Request To Send; one of the inputs is generally connected to Clear To Send or Carrier Detected (which are equivalent in the CB-CF common configuration of most low-speed modems), and the other generally receives Ring Indicator, Data Set Ready, or Restrained Detected. Besides the standard modem control, the hardware also handles the signals necessary to control automatic calling units (ACU) for placing calls on data lines associated with two of the modems.

There is only one type of interrupt for a DC10E line. Changes in status conditions of interest to the program set a status flag, which the scanner monitors in the same way as a receiver flag on a data line. When the scanner interrupts, a DATAI can be used to read the line number and the status information. If the program desires to read the status of a particular line at any time, it can give a DATAO that deliberately sets the status flag so that the scanner will interrupt the next time it reaches the line.

### Instructions

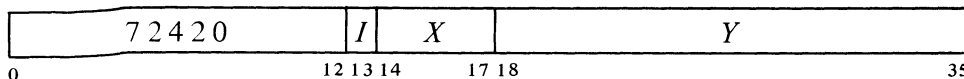
The data line scanner uses the four IO instructions with a single device code. However, the meaning of the information sent by a DATAO or read by a DATAI depends on whether the instruction is dealing with a data line or a modem control line. Hence, the programmer must keep track of whether a DATAI or DATAO is for a DC10B or DC10E. The DC10 device code is 240, mnemonic DLS. A second DC10 would have device code 244.

The lines are numbered 00–77 octal, where the left octal digit indicates the group, the right digit the line within the group. In a DC10E, the ACUs are on lines 6 and 7. The programmer must not only distinguish between the lines for data and the lines for modem control, but in particular he must keep track of which DC10E line controls the modem for a given DC10B line.

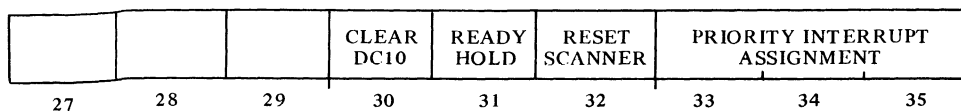
In the DC10E, the three leads are labeled DATA TERM READY, CLR TO SND, and RESTRN DETCTD.

The program can select a line to send out modem control information just as it can to begin data transmission. The device explained here for reading status is necessary because input is available only on a line selected by the scanner.

### CONO DLS, Conditions Out, Data Line Scanner



Assign the interrupt channel specified by bits 33–35 of the effective conditions *E* and perform the function specified by bits 30–32 (a 1 in a bit produces the indicated function, a 0 has no effect).



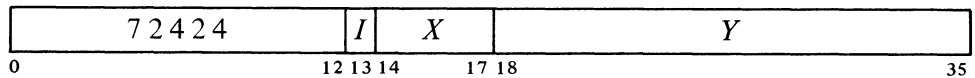
*Notes*

Power turnon and the IO reset signal generated by CONO APR,200000 duplicate this clear function.

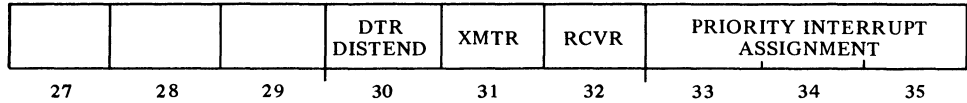
The switch is normally left on, so this bit need not be used, and the program exercises complete control over the Data Terminal Ready signals via the DC10E lines.

- 30 Clear all units in the DC10: clear scanner, receivers, transmitters, flags and control circuits; dismiss PI assignment; and if the DTR distend switch is off, also turn off all transmitters and modem and automatic calling unit control signals.
- 31 Should the DTR distend switch be off, giving a 1 in this bit enables the Data Terminal Ready circuits in the modem controls for over half a second. This allows any modem control to hold on a Data Terminal Ready signal that has been turned on or is turned on within the hold time.
- 32 Set the scanner to line 0.

**CONI DLS, Conditions In, Data Line Scanner**



Read the status of the scanner into bits 30–35 of location *E* as shown.

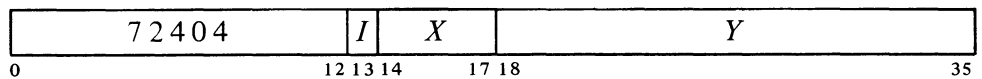


*Notes*

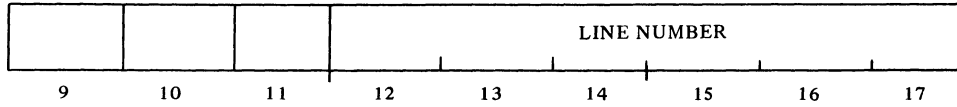
Bits 31 and 32 are valid only when the scanner has stopped and requested an interrupt.

- 30 The DTR distend switch is on. This switch generates Ready Hold continuously (CONO bit 31) so the program need not keep reasserting it, and neither IO reset nor the clear function (CONO bit 30) can turn off Data Terminal Ready signals to the modems.
- 31 The scanner has requested an interrupt for a line whose transmitter flag is on.
- 32 The scanner has requested an interrupt for a line whose receiver flag is on.

**DATAI DLS, Data In, Data Line Scanner**



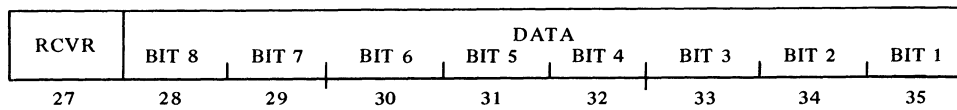
Perform functions for the line selected by the scanner and read the information for it into bits 12–17 and 27–35 of location *E*. The left half word always receives the information shown here.



Bits 12–17 specify the line currently selected by the scanner.

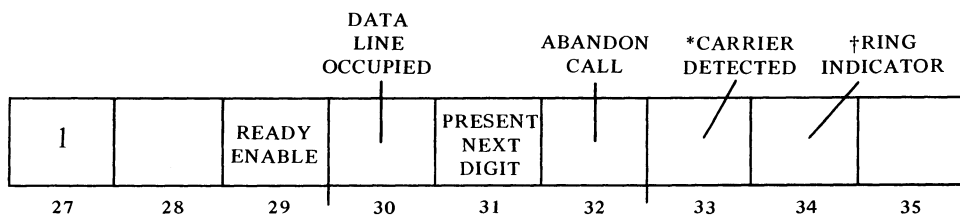
The information received in the right half word and the functions performed depend on whether the selected line is for a DC10B or DC10E as shown below.

*DC10B Line.*



If bit 27 is 1, clear the receiver flag and restart the scanner. The interrupt is for the receiver and bits 28–35 contain the data, with bit 35 received first. An 8-bit character uses all the data bits, a 5-bit character is in bits 31–35 (bits 28–30 are irrelevant). A 0 in bit 27 indicates the interrupt is for the transmitter and bits 28–35 are zero.

*DC10E Line.* Clear the status flag and restart the scanner.



A 1 in bit 29 indicates Ready Enable has been set by a DATAI. This means Data Terminal Ready (or Request To Send) is on provided it has not been turned off within the last 100 ms (and assuming the DTR distend switch is on or the program has given Ready Hold (CONO bit 31) within the last half-second). Bits 33 and 34 are signals from the modem, bits 30–32 are signals from the ACU and are applicable only to lines 6 and 7 (a 1 in a bit indicates the signal is on).

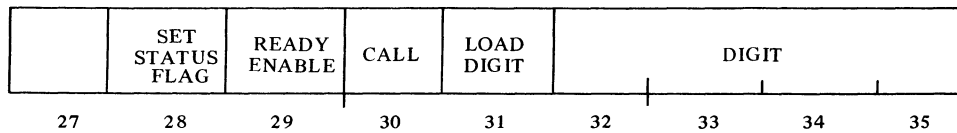
The status flag is set, requesting an interrupt on the assigned channel when the scanner encounters it, by the turnon of any of the signals monitored by bits 31–35 and the turnoff of the signal for bit 33.

The information read is valid only when the scanner has stopped and requested an interrupt.

Alternates:  
 \*Clear To Send  
 †Restrained Detected or Data Set Ready

For switched lines, bits 33 and 34 would likely be used to monitor the signals shown, whereas for unswitched lines they might be used for Clear To Send and Data Set Ready. In 4-row TWX service, bit 34 should be used for Restrained Detected. If no signal is desired, it should be held off by connecting the lead to a negative voltage between -3 and -25 volts.





On the DC10E line specified by bits 12–17 perform the functions specified by bits 28–31 (a 1 in a bit produces the indicated function; in bits 29 and 30 a 0 produces the opposite effect, in bits 28 and 31 a 0 has no effect). Bits 28 and 29 are for the modem, bits 30 and 31 are for the ACU and are applicable only to lines 6 and 7.

#### Notes

- 28 Set the status flag so the next time the scanner checks this line it will stop and request an interrupt on the assigned channel.
- 29 Turn on Data Terminal Ready provided it has not been turned off within the last 100 ms. If this condition is not satisfied, the signal turns on when it does become satisfied. A 0 in this bit turns off Data Terminal Ready.
- 30 Turn on Call Request. A 0 turns off Call Request.
- 31 Make the digit in bits 32–35 available to the ACU, and if Present Next Digit is on (DATAI bit 31), turn on Digit Present.

Whenever Data Terminal Ready is turned off, it is held off for 100 ms to ensure that the data set disconnects properly. This bit can be used instead to control Request To Send, eg with a 103F data set.

#### Data Line Programming

For the data lines per se, the program need give a CONO only initially to select the PI channel. To begin transmission on a line, the program must take initiating action in supplying the first character; then as the transmission of each character is completed, the transmitter sets its flag. Reception requires no initiating action by the program involving the data line, and as each character is assembled the receiver sets its flag. Whenever the scanner encounters a flagged line, it stops to hold the line number and requests an interrupt on the assigned channel. The program can give a CONI (CONSO, CONSZ) to look at both flags for the line, but a DATAI is sufficient: either it has the character, or a 0 in bit 27 indicates that the program must give a DATAO to supply a character to the transmitter specified by DATAI bits 12–17. Either way, the instruction that transfers the character clears the corresponding flag and restarts the scanner. If both flags are set simultaneously, the scanner services the receiver first. In other words, the DATAI reads the received character and clears the receiver flag, but instead of continuing, the scanner immediately requests another interrupt for the transmitter.

Even with a conditions-in check, a DATAI must be given for a receiver or to get the line number for a transmitter.

The program reads input only on an interrupt and from the receiver selected by the scanner. But for output, the program must take several different actions, namely to begin, continue, and terminate transmission. For 8-bit characters, these are distinguished by the configuration of bits 11 and 27 in a DATAO as follows (parentheses indicate equivalent information for 5-bit characters).

	<i>Bit 11</i>	<i>Bit 27 (Bit 30)</i>	<i>Function</i>
If the transmitter selected is already active, the data is garbled.	1	0 (1)	Transmit the character in bits 28–35 over the line selected by bits 12–17. This is used to begin transmission on an idle line. It can be given at any time without affecting the scanner (except to hold it during execution).
	0	0 (1)	Transmit the character in bits 28–35 over the line selected by the scanner, clear the transmitter flag, and restart the scanner. This is used only in a transmitter interrupt when the scanner has stopped.
When transmission is complete, this form must be given to release the scanner.	0	1 (0)	Turn off the transmitter selected by the scanner, clear its flag, and restart the scanner. This is used only in a transmitter interrupt when the scanner has stopped.
This wipes out the character currently being transmitted.	1	1 (0)	Turn off the transmitter selected by bits 12–17. Ordinarily this is not used.

For full-duplex operation with local copy or half-duplex operation, the transmitter does not start sending a character until the receiver for the same line is idle. This is to avoid garbled copy at the teleprinter in the one case, garbled reception in the other. Thus, the program can load a character into a transmitter at any time without affecting the character presently being printed or received respectively, but it may garble the next one.

Characters sent in alternate simplex are also echoed back if the modem is so configured (but there can be no conflict as the channel is one way).

Each character transmitted in half-duplex will subsequently come back to the receiver for the same line, whence the program can check it against the transmitted character. If the two do not match, either a line error has occurred or someone has struck a key at the teletypewriter. The latter action is often used to interrupt program output. Note that every character causes two interrupts, the receiver request being one bit time before that for the transmitter.

**Timing.** A complete scan requires only 20  $\mu$ s and is, therefore, negligible in terms of program response time except at extremely high frequencies (which could be maintained on only a limited number of lines). Where reception is at maximum speed, the program must retrieve a character from the receiver



within  $n + \frac{1}{2}$  bit times, where  $n$  is the number of stop units, to avoid losing data. To keep a transmitter operating at maximum speed, the program must supply a new character in  $n - \frac{1}{2}$  bit times. These times, however, are irrelevant in any reasonable system with a number of lines at different speeds. The program must service every interrupt as quickly as possible to avoid hanging up the scanner, because there is no guarantee that other lines are not already waiting. The time is critical only for reception, where information can be lost, as transmission is simply slowed down. But a slow response by the program to a transmitter interrupt can sink a receiver that is waiting. Even if a receiver is not serviced in time, the program must give the DATAI anyway to release the scanner and clear the receiver flag so the end of the next character can be detected.

### Modem Control Programming

If any DC10E lines are in use, the DTR distend switch should be on. This allows the program to control the Data Terminal Ready signal for each modem individually via its DC10E line and greatly facilitates recovery procedures from a breakdown. When the system is restarted, the IO reset has no effect on the modem control signals. Hence, the software can check through the lines to see which ones are still operating and get the system going again.

To turn on Data Terminal Ready to an individual data set, give a DATAO for the line with a 1 in bit 29. In every DATAO that is ever given for a DC10E line, there must be a 1 in bit 29 unless the programmer specifically wishes to disconnect the modem. When a call is terminated (by a 0 in bit 29), the hardware holds the signal off for 100 ms to ensure that the data set disconnects properly even if the program sets Ready Enable again before that time is up. Generally, the program should turn on Data Terminal Ready on a switched line only to answer a call in response to Ring Indicator or to allow placing of an outgoing call, either manually or through an ACU.

The program can place outgoing calls on the data sets connected to lines 6 and 7 in each DC10E. To determine whether the data set is available, the program can give a DATAO that sets the status flag (bit 28) and then give a DATAI to check Data Line Occupied (bit 30) when the interrupt occurs. To place a call, give a DATAO whose data word selects the line and has a 1 in bit 30 to turn on Call Request and the timer. Reception of a dial tone turns on Data Line Occupied. As the ACU is ready for each digit it turns on Present Next Digit, which sets the status flag. On checking that bit 31 read by a DATAI is on, the program gives a DATAO for the line with the digit in bits 32–35 and a 1 in bit 31 to load the digit into the modem control and turn on Digit Present. When the ACU processes the digit, it turns off Present Next Digit, turning off Digit Present, and restarts the timer. If the timer runs out while waiting for some necessary event in the call procedure, the ACU turns on Abandon Call, which sets the status flag.

With the DTR distend switch off, a system failure causes all data sets to disconnect. To use DC10E lines with the switch off, the program must give a CONO DLS,2P (where  $P$  is the channel assignment) every half-second to keep the data sets connected.

Actually, the hold for the signals is a one-shot that is held on continuously when the switch is on. The circuit is generally set to its maximum time, about .9 second.

To terminate a call and have the line available again as soon as possible (eg to answer any incoming call automatically), the program can give two consecutive DATAOs for the same line, the first having a 0 in bit 29, the second having a 1. The first instruction turns Data Terminal Ready off, and it stays off for 100 ms even though the second instruction sets Ready Enable immediately.

Bits 29 and 30 must also be 1s to hold on Data Terminal Ready and Call Request.

After the last digit has been processed, the ACU again turns on Present Next Digit, and the response by the program depends on the particular setup. The program can simply ignore the interrupt and let the ACU wait for the call to be answered, at which time the channel is returned to the modem; or the program may have the option of or need to give the end-of-number code, which turns the line over to the modem immediately and the modem waits for the answer. If the timer runs out, either before the call is answered or because the modem going into data mode does not stop it, the ACU generates Abandon Call, setting the status flag. After the modem regains the line, the program may be able to drop Call Request if the ACU is so configured, but it is of no concern because the program must eventually give a DATAO anyway to terminate the call by turning off Data Terminal Ready.

If the program gives an end-of-number code, it can use the timer interrupt as a signal to check that the call has been answered.

#### 8.4 SINGLE SYNCHRONOUS LINE UNIT DS10

This unit handles transmission and reception of serial data over one full-duplex, synchronous communication line at speeds of 600 to 20,000 bits per second using EIA standard levels. Although transmission and reception are in the form of a continuous bit stream, this stream is nonetheless handled in terms of characters of six or eight bits in length. Transfer of data between the DS10 and the PDP-10 is in full words via the IO bus. The equipment contains facilities for the program to answer incoming calls. Typical modems used with the DS10 are Bell data sets 201A, 201B, 203, and 205. Timing for reception and transmission is provided by a clock in the data set.

Rates below 2000 require a minor hardware adjustment.

When the character length is six bits, each word has six characters in this order:

FIRST	SECOND	THIRD	FOURTH	FIFTH	SIXTH	
0	5 6	11 12	17 18	23 24	29 30	35

When the character length is eight bits, each word contains thirty-two bits of data comprising four characters in this order:

FIRST	SECOND	THIRD	FOURTH		
0	7 8	15 16	23 24	31 32	35

Transfers over the IO bus are always in 36-bit words, but for 8-bit characters, bits 32–35 are ignored by the transmitter and supplied as 0s by the receiver. In both formats, the least significant bit is transmitted and received first in every character. Communication is initiated by the use of a sync character and should be terminated by an end-of-transmission character (EOT).

Good communication practice dictates the use of a string of sync characters to initiate and a more complex pattern, ending with an EOT, to terminate.

For output, the program selects the character length and supplies the first word with the sync character in the first character position. When the trans-

mitter recognizes the sync character, it sends out the word and signals the program when it is ready for the next one. At the completion of a message, the transmitter can be turned off or the program can cause it to idle, sending the same character over and over, until a new message is ready.

For input, the program must specify the expected character length prior to reception. On detecting a sync character, the receiver assembles the incoming bits into characters of the specified size and assembles the characters into words. As each word is completed, the receiver signals the program to retrieve it.

The equipment contains a number of jumper boards that allow the user to select the polarity of the data signals, the device codes of the equipment, and the configuration of the sync and EOT characters (different sync characters can be selected for transmission and reception). Unless otherwise specified, the equipment is set up for conventional EIA signal polarities, the standard DEC device codes, and the standard ASCII SYN and EOT characters. The sync and end-of-transmission characters respectively are, therefore, 226 and 204 in the 8-bit format, 26 and 04 in the 6-bit format.

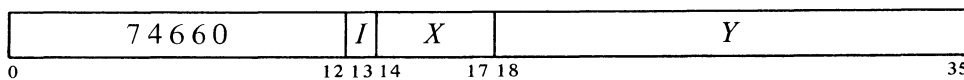
For information on installing jumpers, refer to the *DS10 Maintenance Manual*.

**Instructions**

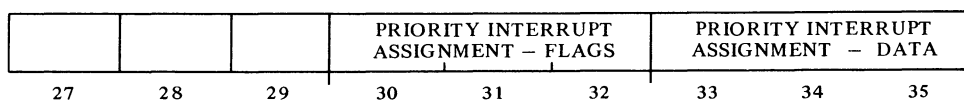
The transmitter and receiver each have a 36-bit word buffer and a shift register for connection to the line for characters of either length. Because the individual characters are handled separately from the buffer, each time the transmitter is free to receive a word from the bus or the receiver has a word for the bus, the program has one entire character time in which to respond.

The DS10 has two device codes, 460 and 464, mnemonics DSS and DSI. Device code 460 is used for both data and conditions; device code 464 is used only for priority interrupt assignments. A second DS10 would have device codes 470 and 474.

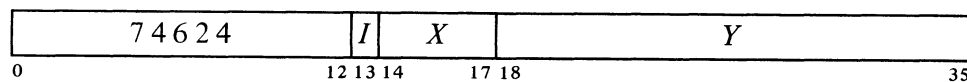
**CONO DSI, Conditions Out, DS10 Interrupt**



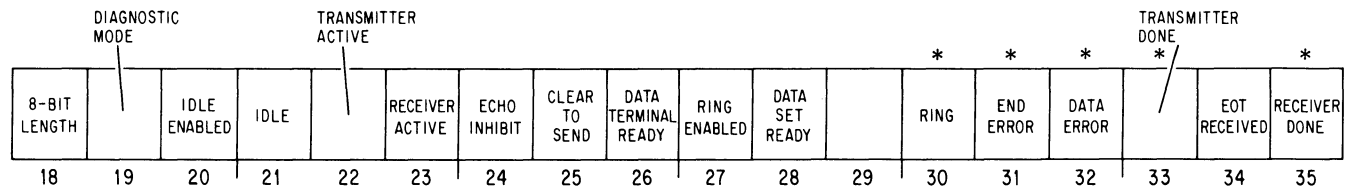
Assign the interrupt channels specified by bits 30–35 of the effective conditions *E* as shown.





**CONI DSS, Conditions In, DS10 Status**


Read the status of the line unit into the right half of location *E* as shown.



DATAI DSS, clears Receiver Done, DATAO DSS, clears Transmitter Done. The setting of Ring, End Error, or Data Error requests an interrupt on the flag channel (assigned by bits 30–32 of the last CONO DSI,). The setting of Transmitter Done or Receiver Done requests an interrupt on the data channel (assigned by bits 33–35 of the last CONO DSI,).

\*These bits cause interrupts.

Bits 25, 26, and 28 are simply the modem control signals (a 1 indicates the signal is on).

- 18 The DS10 is set up to process 8-bit characters; a 0 indicates 6-bit characters.
- 21 The transmitter is idling, *ie* transmitting over and over the first character of the last word supplied by a DATAO.
- 22 The transmitter is now transmitting data. This bit is cleared when the program fails to supply a new word before the last bit of the previous one is transmitted or Clear To Send goes off (bit 25).
- 23 The receiver is now receiving data. This bit is cleared if the program turns off the receiver (CONO DSS, bit 23), carrier is lost, or the receiver clock from the data set goes off.
- 24 If the line is set up for alternate-simplex operation, characters transmitted will not be echoed back to the receiver.
- 30 A ringing signal was received while Ring Enabled (bit 27) was set.
- 31 Either carrier was lost or the receiver clock from the data set went off before an EOT character was received (*ie* while bit 34 was clear).
- 32 The program failed to respond to a receiver interrupt before the next character was assembled. Until a DATAI is given, the buffer will still contain the word previously assembled, but subsequent characters will have been lost.

In the usual fashion, the program can answer by turning on Data Terminal Ready (CONO DSS, bit 26).



channel assigned to data. Within one character time, the program must respond with a DATAO that supplies the next word and clears Transmitter Done. Otherwise, when the transmitter finishes sending the final character in the word, it clears Transmitter Active and Transmitter Done and turns off Request To Send. This means that the program can terminate transmission simply by refusing to respond to the interrupt following the last word – or by dismissing the data PI assignment when giving the last word, so there will be no interrupt at all.

The program may wish to keep a line active even though there are intervals between messages transmitted. To do this give a CONO DSS, with a 1 in bit 20 to enable Idle after loading the last word of the message into the buffer; then at the next interrupt give a DATAO to load a word whose first character is some line idling character previously arranged with the remote station. This DATAO clears Transmitter Done, but it sets Idle so that Transmitter Active remains set indefinitely, causing the transmitter to send out the first character of the word over and over. This idling procedure continues until the program gives a CONO DSS, with a 0 in bit 20 to clear Idle and Idle Enabled, at which time the transmitter sends the rest of the word.

**Input.** While a communication link is maintained, the receiver continuously monitors the incoming bit stream; with the line idle, this is usually continuous marking. Whenever the receiver detects the predefined sync character in the length specified by the program, it synchronizes to that character and sets Receiver Active. From that point on, it assembles incoming bits into characters, and assembles the characters into words in the buffer, taking the first sync character as the first in the first word. As it loads the last character of each word into the buffer it sets Receiver Done, requesting an interrupt on the channel assigned to data. The program must respond within one character time with a DATAI to get the word and clear Receiver Done. If the receiver assembles an entire character before the buffer is read, it simply throws the character away and sets Data Error, requesting an interrupt on the channel assigned to the flags; the last word assembled remains inviolate in the buffer, but all subsequent characters are thrown away until Receiver Done is cleared.

If the receiver detects the predefined end-of-transmission character in the incoming data, it sets EOT Received. The program can determine whether such a character is received (perhaps with each DATAI) by checking for a 1 in bit 34 of the input conditions. It is up to the program to determine whether such a character really represents the end of a message or the end of transmission and is not just the equivalent set of bits, as would quite likely happen if the input is binary data without a true character structure. The program can turn off the receiver at any time by giving a 1 in bit 23 of a CONO DSS. This action or the loss of carrier or clock from the data set clears Receiver Active, halting reception and clearing Receiver Done. If carrier or clock goes off before an EOT character comes in (*ie* while EOT Received is clear), the receiver also sets End Error, requesting an interrupt on the channel assigned to the flags.

*Eg* the program may wish to wait for the remote station to respond to one message before sending another.

The remaining characters in the word may be part of the next message if this is known ahead of time, or they can simply be more idling characters, or sync characters to signal the beginning of the next message.

In alternate-simplex operation, the data transmitted is often echoed back to the receiver from the data set. If no echo is desired, the program can inhibit it by giving a CONO DSS, with a 1 in bit 24; then the input to the receiver is held marking regardless of what is sent out by the transmitter.

**Timing.** Timing depends on various characteristics of the modem, particularly the bit rate. Given the bit rate, the only critical factor for the program is that it must respond to a data interrupt within one character time. Failure of the receiver clock is assumed if no pulses arrive for  $650 \mu\text{s}$  as determined by a timing circuit; this covers speeds of 2000 bits per second and above. For slower speeds, the circuit should be adjusted to some reasonable period greater than the bit time.

**Diagnostic Programming.** Placing the unit in diagnostic mode (by giving a CONO DSS, with a 1 in bit 19) causes it to act in the normal manner but totally divorced from the modem. Instead, the necessary control signals are simulated: Data Terminal Ready produces Data Set Ready, Request To Send produces Clear To Send, and an internal clock supplies both the receiver and transmitter clocks at a frequency of 2500 Hz. With this arrangement, the program can transmit data, starting with a sync character and then responding to Transmitter Done, but the data sent out is fed directly into the receiver, which also functions normally.

Leaving Ring disabled simulates Carrier Detected so the standard logic can monitor both it and the receiver clock for an end error; enabling Ring allows the diagnostic clock to set the Ring flag.

The same circuit is used to detect loss of carrier: it is assumed that carrier is lost if Carrier Detected remains off for the period of the circuit.

The data set need not actually be disconnected.



# Appendices

2

3

4

5



## APPENDIX A

### INSTRUCTIONS AND MNEMONICS

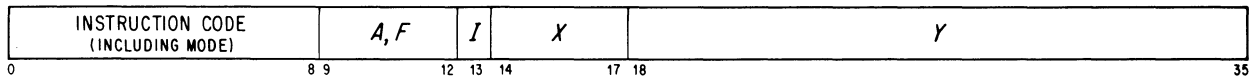
The drawing on the next two pages shows the formats of the various types of words used by the processor (instructions, pointers, operands, etc). The illustration on page A-4 shows the derivation of the instruction mnemonics. Next are two tables that list all instruction mnemonics and their octal codes both numerically and alphabetically. When two mnemonics are given for the same octal code, the first is the preferred form, but the assembler does recognize the second. For completeness, the tables include the MUUOs (indicated by an asterisk) that are recognized by MACRO for communication with the DECsystem-10 Time Sharing Monitor. A double dagger (§) indicates a KI10 instruction code that is unassigned in the KA10.

In-out device codes are included only in the alphabetic listing and are indicated by a dagger (†). Following the tables is a chart that lists the devices with their mnemonic and octal codes and DEC type numbers for both PDP-10 and PDP-6. A device mnemonic ending in the numeral 2 is the recommended form for the second of a given device, but such codes are not recognized by MACRO — they must be defined by the user.

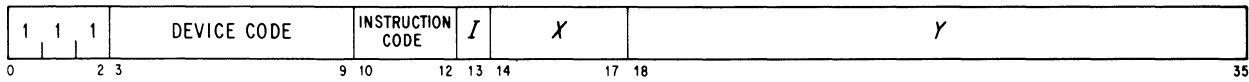
Beginning on page A-13 is a list of all instructions showing their actions in symbolic form. On page A-23 is a table of the positive and negative powers of 2.

Note that 247 is unassigned in the KI10, but 247 and 257 execute as no-ops in the KA10.

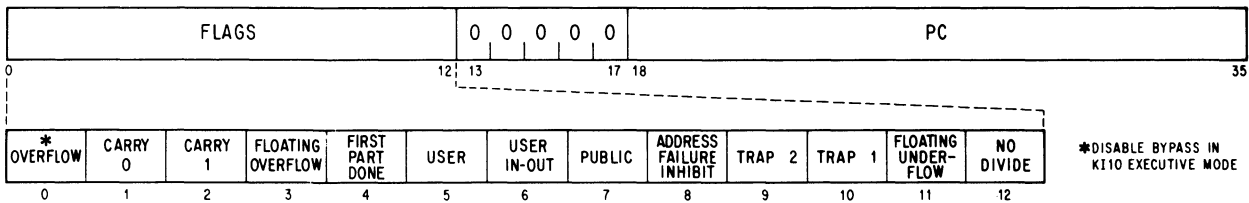
**BASIC INSTRUCTIONS**



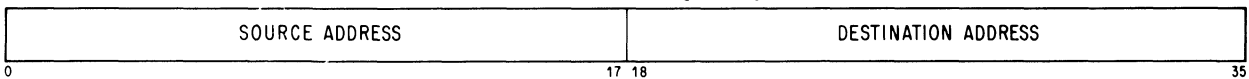
**IN-OUT INSTRUCTIONS**



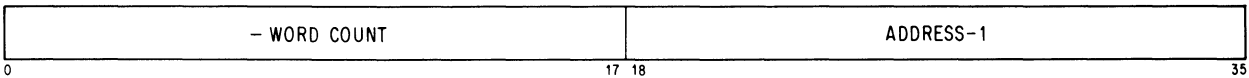
**PC WORD**



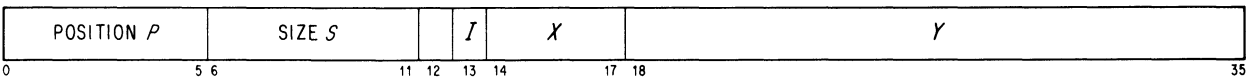
**BLT POINTER {XWD}**



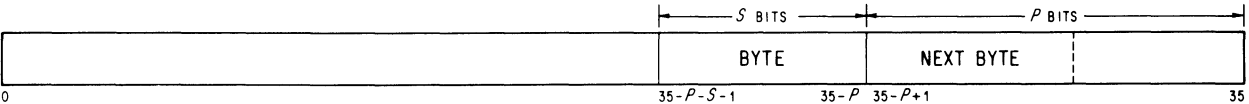
**BLKI / BLKO POINTER, PUSHDOWN POINTER, DATA CHANNEL CONTROL WORD {IOWD}**



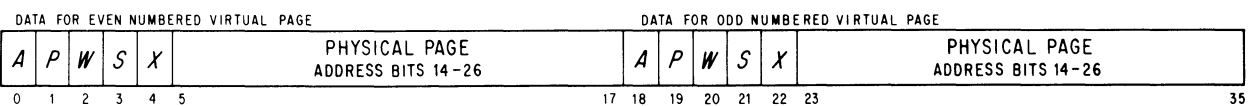
**BYTE POINTER**



**BYTE STORAGE**



**PAGE MAP WORD**



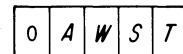
**PAGE FAIL WORD**



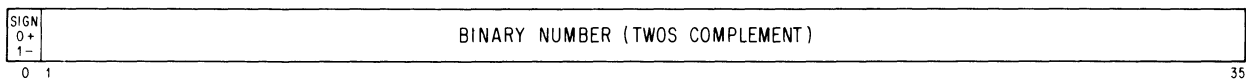
- 20 SMALL USER VIOLATION
- 21 PROPRIETARY VIOLATION

- 22 PAGE REFILL FAILURE
- 23 ADDRESS FAILURE

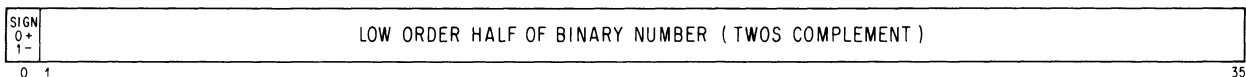
IF BIT 31 IS 0, BITS 31-35 HAVE THIS FORMAT



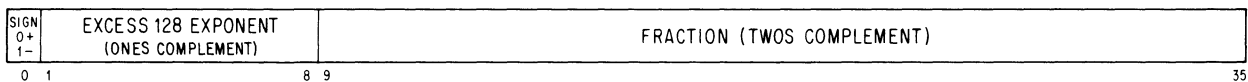
## FIXED POINT OPERANDS (SINGLE PRECISION OR HIGH ORDER WORD)



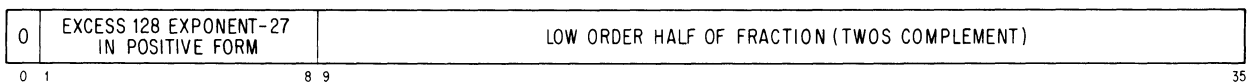
## LOW ORDER WORD IN DOUBLE LENGTH FIXED POINT OPERANDS



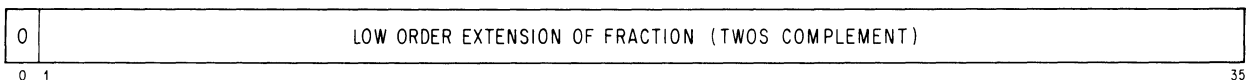
## FLOATING POINT OPERANDS (SINGLE PRECISION OR HIGH ORDER WORD)



## LOW ORDER WORD IN SOFTWARE DOUBLE LENGTH FLOATING POINT OPERANDS



## LOW ORDER WORD IN HARDWARE DOUBLE LENGTH FLOATING POINT OPERANDS



<p>MOV { E Negative Magnitude Swapped }          Half word { Right to Left } to { Right to Left } { no effect Ones Zeros Extend sign }          BLock Transfer          EXCHange AC and memory</p>	<p>ADD          SUBtract          MULTiPLY          Integer MULTiPLY          DIVide          Integer DIVide } { to AC Immediate to AC to Memory to Self }          Floating Add          Floating SuBtract          Floating MultiPLY          Floating DiVide } { and Round } { ~ Immediate to Memory to Both }          Floating SCale          Double Floating Negate          Unnormalized Floating Add          FIX          FIX and Round          FLoaT and Round          Double Floating Add          Double Floating SuBtract          Double Floating MultiPLY          Double Floating DiVide          Double MOV { E Negative } { ~ to Memory }</p>
<p>use present pointer } and { LoaD Byte into AC Increment pointer } and { DePosit Byte in memory }          Increment Byte Pointer</p>	<p>Jump { to SubRoutine and Save Pc and Save AC and Restore AC if Find First One on Flag and CLear it on OVerflow (JFCL 10,) on CaRrY 0 (JFCL 4,) on CaRrY 1 (JFCL 2,) on CaRrY (JFCL 6,) on Floating OVerflow (JFCL 1,) and ReSTore and ReSTore Flags (JRST 2,) and ENable PI channel (JRST 12,) }          HALT (JRST 4,)          PORTAL (JRST 1,)          eXeCuTe</p>
<p>PUSH down } { ~ POP up } { and Jump }</p>	<p>DATA }          BLocK } { In Out }          CONditions { in and Skip if { all masked bits Zero some masked bit One } }</p>
<p>SET to { Zeros Ones AC Memory Complement of AC Complement of Memory }          AND inclusive OR } { ~ with Complement of AC with Complement of Memory Complements of Both }          Inclusive OR eXclusive OR EQuiValence } to { AC AC Immediate Memory Both }</p>	<p>SKIP if memory }          JUMP if AC } { never Less Equal Less or Equal Always Greater Greater or Equal Not equal }          Add One to Subtract One from } { memory and Skip } if { AC and Jump }          Compare AC { Immediate with Memory } and skip if AC          Add One to Both halves of AC and Jump if { Positive Negative }</p>
<p>Arithmetic SHift }          Logical SHift } { ~ ROTate } { Combined }</p>	<p>Test AC { with Direct mask with Swapped mask Right with E Left with E } { No modification set masked bits to Zeros set masked bits to Ones Complement masked bits } and skip { never if all masked bits Equal 0 if Not all masked bits equal 0 Always }</p>

INSTRUCTION MNEMONICS  
 NUMERIC LISTING

000	ILLEGAL	106		162	FMPM
001	} LUUO'S	107		163	FMPB
:		110	‡DFAD	164	FMPR
037		111	‡DFSB	165	FMPRI
040	*CALL	112	‡DFMP	166	FMPRM
041	*INIT	113	‡DFDV	167	FMPRB
042	} RESERVED FOR SPECIAL MONITORS	114		170	FDV
043		115		171	FDVL
044		116		172	FDVM
045		117		173	FDVB
046		120	‡DMOVE	174	FDVR
047	*CALLI	121	‡DMOVN	175	FDVRI
050	*OPEN	122	‡FIX	176	FDVRM
051	*TTCALL	123		177	FDVRB
052	} RESERVED FOR DEC	124	‡DMOVEM	200	MOVE
053		125	‡DMOVNM	201	MOVEI
054		126	‡FIXR	202	MOVEM
055	*RENAME	127	‡FLTR	203	MOVES
056	*IN	130	UFA	204	MOVS
057	*OUT	131	DFN	205	MOVSI
060	*SETSTS	132	FSC	206	MOVSM
061	*STATO	133	IBP	207	MOVSS
062	*STATUS	134	ILDB	210	MOVN
062	*GETSTS	135	LDB	211	MOVNI
063	*STATZ	136	IDPB	212	MOVNM
064	*INBUF	137	DPB	213	MOVNS
065	*OUTBUF	140	FAD	214	MOVMM
066	*INPUT	141	FADL	215	MOVMI
067	*OUTPUT	142	FADM	216	MOVMM
070	*CLOSE	143	FADB	217	MOVMS
071	*RELEAS	144	FADR	220	IMUL
072	*MTAPE	145	FADR1	221	IMULI
073	*UGETF	146	FADRM	222	IMULM
074	*USETI	147	FADRB	223	IMULB
075	*USETO	150	FSB	224	MUL
076	*LOOKUP	151	FSBL	225	MULI
077	*ENTER	152	FSBM	226	MULM
100	*UJEN	153	FSBB	227	MULB
101		154	FSBR	230	IDIV
102		155	FSBRI	231	IDIVI
103		156	FSBRM	232	IDIVM
104		157	FSBRB	233	IDIVB
105		160	FMP	234	DIV
		161	FMPL	235	DIVI

236	DIVM	306	CAIN	367	SOJG
237	DIVB	307	CAIG	370	SOS
240	ASH	310	CAM	371	SOSL
241	ROT	311	CAML	372	SOSE
242	LSH	312	CAME	373	SOSLE
243	JFFO	313	CAMLE	374	SOSA
244	ASHC	314	CAMA	375	SOSGE
245	ROTC	315	CAMGE	376	SOSN
246	LSHC	316	CAMN	377	SOSG
247		317	CAMG	400	SETZ
250	EXCH	320	JUMP	400	CLEAR
251	BLT	321	JUMPL	401	SETZI
252	AOBJP	322	JUMPE	401	CLEARI
253	AOBNJ	323	JUMPLE	402	SETZM
254	JRST	324	JUMPA	402	CLEARM
25404	PORTAL	325	JUMPGE	403	SETZB
25410	JRSTF	326	JUMPN	403	CLEARB
25420	HALT	327	JUMPG	404	AND
25450	JEN	330	SKIP	405	ANDI
255	JFCL	331	SKIPL	406	ANDM
25504	JFOV	332	SKIPE	407	ANDB
25510	JCRY1	333	SKIPLE	410	ANDCA
25520	JCRY0	334	SKIPA	411	ANDCAI
25530	JCRY	335	SKIPGE	412	ANDCAM
25540	JOV	336	SKIPN	413	ANDCAB
256	XCT	337	SKIPG	414	SETM
257	‡MAP	340	AOJ	415	SETMI
260	PUSHJ	341	AOJL	416	SETMM
261	PUSH	342	AOJE	417	SETMB
262	POP	343	AOJLE	420	ANDCM
263	POPJ	344	AOJA	421	ANDCMI
264	JSR	345	AOJGE	422	ANDCMM
265	JSP	346	AOJN	423	ANDCMB
266	JSA	347	AOJG	424	SETA
267	JRA	350	AOS	425	SETAI
270	ADD	351	AOSL	426	SETAM
271	ADDI	352	AOSE	427	SETAB
272	ADDM	353	AOSLE	430	XOR
273	ADDB	354	AOSA	431	XORI
274	SUB	355	AOSGE	432	XORM
275	SUBI	356	AOSN	433	XORB
276	SUBM	357	AOSG	434	IOR
277	SUBB	360	SOJ	434	OR
300	CAI	361	SOJL	435	IORI
301	CAIL	362	SOJE	435	ORI
302	CAIE	363	SOJLE	436	IORM
303	CAILE	364	SOJA	436	ORM
304	CAIA	365	SOJGE	437	IORB
305	CAIGE	366	SOJN	437	ORB



440	ANDCB	521	HLLOI	602	TRNE
441	ANDCBI	522	HLLOM	603	TLNE
442	ANDCBM	523	HLLOS	604	TRNA
443	ANDCBB	524	HRLO	605	TLNA
444	EQV	525	HRLOI	606	TRNN
445	EQVI	526	HRLOM	607	TLNN
446	EQVM	527	HRLOS	610	TDN
447	EQVB	530	HLLE	611	TSN
450	SETCA	531	HLLEI	612	TDNE
451	SETCAI	532	HLLEM	613	TSNE
452	SETCAM	533	HLLES	614	TDNA
453	SETCAB	534	HRLE	615	TSNA
454	ORCA	535	HRLEI	616	TDNN
455	ORCAI	536	HRLEM	617	TSNN
456	ORCAM	537	HRLES	620	TRZ
457	ORCAB	540	HRR	621	TLZ
460	SETCM	541	HRRI	622	TRZE
461	SETCMI	542	HRRM	623	TLZE
462	SETCMM	543	HRRS	624	TRZA
463	SETCMB	544	HLR	625	TLZA
464	ORCM	545	HLRI	626	TRZN
465	ORCMI	546	HLRM	627	TLZN
466	ORCMM	547	HLRS	630	TDZ
467	ORCMB	550	HRRZ	631	TSZ
470	ORCB	551	HRRZI	632	TDZE
471	ORCBI	552	HRRZM	633	TSZE
472	ORCBM	553	HRRZS	634	TDZA
473	ORCBB	554	HLRZ	635	TSZA
474	SETO	555	HLRZI	636	TDZN
475	SETOI	556	HLRZM	637	TSZN
476	SETOM	557	HLRZS	640	TRC
477	SETOB	560	HRRO	641	TLC
500	HLL	561	HRROI	642	TRCE
501	HLLI	562	HRROM	643	TLCE
502	HLLM	563	HRROS	644	TRCA
503	HLLS	564	HLRO	645	TLCA
504	HRL	565	HLROI	646	TRCN
505	HRLI	566	HLROM	647	TLCN
506	HRLM	567	HLROS	650	TDC
507	HRLS	570	HRRE	651	TSC
510	HLLZ	571	HRREI	652	TDCE
511	HLLZI	572	HRREM	653	TSCE
512	HLLZM	573	HRRES	654	TDCA
513	HLLZS	574	HLRE	655	TSCA
514	HRLZ	575	HLREI	656	TDCN
515	HRLZI	576	HLREM	657	TSCN
516	HRLZM	577	HLRES	660	TRO
517	HRLZS	600	TRN	661	TLO
520	HLLO	601	TLN	662	TROE

663	TLOE	673	TSOE	70010	BLKO
664	TROA	674	TDOA	70014	DATAO
665	TLOA	675	TSOA	70020	CONO
666	TRON	676	TDON	70024	CONI
667	TLON	677	TSON	70030	CONSZ
670	TDO	70000	BLKI	70034	CONSO
671	TSO	70004	DATAI		
672	TDOE	70004	RSW		

## INSTRUCTION MNEMONICS

## ALPHABETIC LISTING

†ADC	024	AOSA	354	†CDP	110
ADD	270	AOSE	352	†CDR	114
ADDB	273	AOSG	357	CLEAR	400
ADDI	271	AOSGE	355	CLEARB	403
ADDM	272	AOSL	351	CLEARI	401
AND	404	AOSLE	353	CLEARM	402
ANDB	407	AOSN	356	†CLK	070
ANDCA	410	†APR	000	*CLOSE	070
ANDCAB	413	ASH	240	CONI	70024
ANDCAI	411	ASHC	244	CONO	70020
ANDCAM	412	BLKI	70000	CONSO	70034
ANDCB	440	BLKO	70010	CONSZ	70030
ANDCBB	443	BLT	251	†CPA	000
ANDCBI	441	CAI	300	†CR	150
ANDCBM	442	CAIA	304	DATAI	70004
ANDCM	420	CAIE	302	DATAO	70014
ANDCMB	423	CAIG	307	†DC	200
ANDCMI	421	CAIGE	305	†DCSA	300
ANDCMM	422	CAIL	301	†DCSB	304
ANDI	405	CAILE	303	‡DFAD	110
ANDM	406	CAIN	306	‡DFDV	113
AOBJN	253	*CALL	040	‡DFMP	112
AOBJP	252	*CALLI	047	DFN	131
AOJ	340	CAM	310	‡DFSB	111
AOJA	344	CAMA	314	†DIS	130
AOJE	342	CAME	312	DIV	234
AOJG	347	CAMG	317	DIVB	237
AOJGE	345	CAMGE	315	DIVI	235
AOJL	341	CAML	311	DIVM	236
AOJLE	343	CAMLE	313	†DLB	060
AOJN	346	CAMN	316	†DLC	064
AOS	350	†CCI	014	†DLS	240

‡DMOVE	120	FSBRB	157	HRLS	507
‡DMOVEM	124	FSBRI	155	HRLZ	514
‡DMOVN	121	FSBRM	156	HRLZI	515
‡DMOVNM	125	FSC	132	HRLZM	516
DPB	137	*GETSTS	062	HRLZS	517
†DPC	250	HALT	25420	HRR	540
†DSI	464	HLL	500	HRRE	570
†DSK	170	HLLE	530	HRREI	571
†DSS	460	HLLEI	531	HRREM	572
†DTC	320	HLLEM	532	HRRES	573
†DTS	324	HLLES	533	HRRI	541
*ENTER	077	HLLI	501	HRRM	542
EQV	444	HLLM	502	HRRO	560
EQVB	447	HLLO	520	HRROI	561
EQVI	445	HLLOI	521	HRROM	562
EQVM	446	HLLOM	522	HRROS	563
EXCH	250	HLLOS	523	HRRS	543
FAD	140	HLLS	503	HRRZ	550
FADB	143	HLLZ	510	HRRZI	551
FADL	141	HLLZI	511	HRRZM	552
FADM	142	HLLZM	512	HRRZS	553
FADR	144	HLLZS	513	IBP	133
FADRB	147	HLR	544	IDIV	230
FADRI	145	HLRE	574	IDIVB	233
FADRM	146	HLREI	575	IDIVI	231
FDV	170	HLREM	576	IDIVM	232
FDVB	173	HLRES	577	IDPB	136
FDVL	171	HLRI	545	ILDB	134
FDVM	172	HLRM	546	IMUL	220
FDVR	174	HLRO	564	IMULB	223
FDVRB	177	HLROI	565	IMULI	221
FDVRI	175	HLROM	566	IMULM	222
FDVRM	176	HLROS	567	*IN	056
‡FIX	122	HLRS	547	*INBUF	064
‡FIXR	126	HLRZ	554	*INIT	041
‡FLTR	127	HLRZI	555	*INPUT	066
FMP	160	HLRZM	556	IOR	434
FMPB	163	HLRZS	557	IORB	437
FMPL	161	HRL	504	IORI	435
FMPM	162	HRLE	534	IORM	436
FMPR	164	HRLEI	535	JCRY	25530
FMPRB	167	HRLEM	536	JCRY0	25520
FMPRI	165	HRLES	537	JCRY1	25510
FMPRM	166	HRLI	505	JEN	25460
FSB	150	HRLM	506	JFCL	255
FSBB	153	HRLO	524	JFFO	243
FSBL	151	HRLOI	525	JFOV	25504
FSBM	152	HRLOM	526	JOV	25540
FSBR	154	HRLOS	527	JRA	267

JRST	254	ORCAM	456	SETOM	476
JRSTF	25410	ORCB	470	*SETSTS	060
JSA	266	ORCBB	473	SETZ	400
JSP	265	ORCBI	471	SETZB	403
JSR	264	ORCBM	472	SETZI	401
JUMP	320	ORCM	464	SETZM	402
JUMPA	324	ORCMB	467	SKIP	330
JUMPE	322	ORCMI	465	SKIPA	334
JUMPG	327	ORCMM	466	SKIPE	332
JUMPGE	325	ORI	435	SKIPG	337
JUMPL	321	ORM	436	SKIPGE	335
JUMPLE	323	*OUT	057	SKIPL	331
JUMPN	326	*OUTBUF	065	SKIPL	333
LDB	135	*OUTPUT	067	SKIPN	336
*LOOKUP	076	†PAG	010	SOJ	360
†LPT	124	†PI	004	SOJA	364
LSH	242	†PLT	140	SOJE	362
LSHC	246	POP	262	SOJG	367
‡MAP	257	POPJ	263	SOJGE	365
MOVE	200	PORTAL	25404	SOJL	361
MOVEI	201	†PTP	100	SOJLE	363
MOVEM	202	†PTR	104	SOJN	366
MOVES	203	PUSH	261	SOS	370
MOVMI	214	PUSHJ	260	SOSA	374
MOVMM	215	*RELEAS	071	SOSE	372
MOVMS	217	*RENAME	055	SOSG	377
MOVN	210	†RMC	270	SOSGE	375
MOVNI	211	ROT	241	SOSL	371
MOVNM	212	ROTC	245	SOSLE	373
MOVNS	213	RSW	70004	SOSN	376
MOVSS	204	SETA	424	*STATO	061
*MTAPE	072	SETAB	427	*STATUS	062
†MTC	220	SETAI	425	*STATZ	063
†MTM	230	SETAM	426	SUB	274
†MTS	224	SETCA	450	SUBB	277
MUL	224	SETCAB	453	SUBI	275
MULB	227	SETCAI	451	SUBM	276
MULI	225	SETCAM	452	TDC	650
MULM	226	SETCM	460	TDCA	654
*OPEN	050	SETCMB	463	TDCE	652
OR	434	SETCMI	461	TDCN	656
ORB	437	SETCMM	462	TDN	610
ORCA	454	SETM	414	TDNA	614
ORCAB	457	SETMB	417	TDNE	612
ORCAI	455	SETMI	415	TDNN	616
		SETMM	416	TDO	670
		SETO	474	TDOA	674
		SETOB	477	TDOE	672
		SETOI	475	TDON	676

TDZ	630	TRCA	644	TSO	671
TDZA	634	TRCE	642	TSOA	675
TDZE	632	TRCN	646	TSOE	673
TDZN	636	TRN	600	TSOZ	677
TLC	641	TRNA	604	TSZ	631
TLCA	645	TRNE	602	TSZA	635
TLCE	643	TRNN	606	TSZE	633
TLCN	647	TRO	660	TSZN	637
TLN	601	TROA	664	*TTCALL	051
TLNA	605	TROE	662	UFA	130
TLNE	603	TRON	666	*UGETF	073
TLNN	607	TRZ	620	*UJEN	100
TLO	661	TRZA	624	*USETI	074
TLOA	665	TRZE	622	*USETO	075
TLOE	663	TRZN	626	†UTC	210
TLON	667	TSC	651	†UTS	214
TLZ	621	TSCA	655	XCT	256
TLZA	625	TSCE	653	XOR	430
TLZE	623	TSCN	657	XORB	433
TLZN	627	TSN	611	XORI	431
†TMC	340	TSNA	615	XORM	432
†TMS	344	TSNE	613		
TRC	640	TSNN	617		

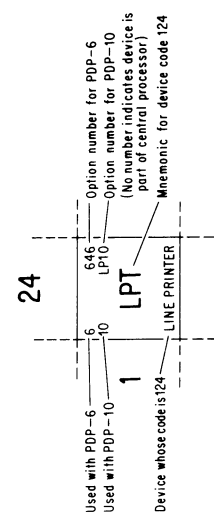
SECOND AND THIRD OCTAL DIGITS		00	04	10	14	20	24	30	34	40	44	50	54	60	64	70	74
FIRST OCTAL DIGIT	0	6,10	10	10	10	10	10	AD10	AD10	10	10	10	10	10	DL10	DK10	DK10
0	APR CENTRAL PROCESSOR	CPA	PI PRIORITY INTERRUPT	PAG* KIT10 PAGING	CCI PDP-8,9 INTERFACE	CCI2 PDP-8,9 INTERFACE	ADC ANALOG-DIGITAL CONVERTER	ADC2 ANALOG-DIGITAL CONVERTER	DIS2 340,6,10 VP10	DIS 646,6,10 LPI0	LPT LINE PRINTER	PLT2 PLOTTER	CR2 CARD READER	DLB PDP-11 DATA LINK	DLC PDP-11 DATA LINK	CLK REAL TIME CLOCK	CLK2 REAL TIME CLOCK
1	PTP PAPER TAPE PUNCH	PTR PAPER TAPE READER	CDR CARD READER	CDP CARD PUNCH	TTY CONSOLE TELETYPE	UTS UTS	MTS MAGNETIC TAPE	MTM1 LINE PRINTER	DIS2 340,6,10 VP10	PLT PLOTTER	PLT2 PLOTTER	CR CARD READER	CR2 CARD READER	DLB2* PDP-11 DATA LINK	DLC2 PDP-11 DATA LINK	DSK DISK/DRUM	DSK2 DISK/DRUM
2	DC DATA CONTROL	DC2 DATA CONTROL	UTC DECTAPE	MTC MTC	DTC DECTAPE	DTS DECTAPE	DTC2 DECTAPE	DTS2 DECTAPE	LPT21 LINE PRINTER	DLS DATA LINE SCANNER	DLS2 DATA LINE SCANNER	DPC DISK PACK SYSTEM	DPC2 DISK PACK SYSTEM	DPC3 DISK PACK SYSTEM	DPC4 DISK PACK SYSTEM	RMC* DATA CONTROL	RMC2 DATA CONTROL
3	DCSA DATA COMMUNICATION	DCSB DATA COMMUNICATION	UTS UTS	MTC MTC	DTC DECTAPE	DTS DECTAPE	DTC2 DECTAPE	DTS2 DECTAPE	TMC MAGNETIC TAPE	TMS MAGNETIC TAPE	TMS2 MAGNETIC TAPE	TMC2 MAGNETIC TAPE	TMS2 MAGNETIC TAPE	TMC2 MAGNETIC TAPE	TMS2 MAGNETIC TAPE	TMC2 MAGNETIC TAPE	TMS2 MAGNETIC TAPE
4																	
5																	
6																	
7																	

KIT10 UNRESTRICTED CODES  
RESERVED FOR DEC

KIT10 UNRESTRICTED CODES  
RESERVED FOR USERS

CODES IN THIS SECTION RESERVED FOR USER SPECIAL DEVICES  
\*IN THE PDP-6 THESE CODES ARE USED FOR OTHER DEVICES †FOR A THIRD LINE PRINTER USE CODE 230

010 DRUM PROCESSOR  
160 PDP-7,8 INTERFACE  
270 DISK FILE (DF)



DEVICE MNEMONICS

## ALGEBRAIC REPRESENTATION

The remaining pages of this Appendix list, in symbolic form, the actual operations performed by the instructions. The grouping, as given below, differs slightly from that used in Chapter 2.

Boolean	A-15	In-out	A-19
Byte manipulation	A-16	Program control	A-19
Fixed point arithmetic	A-16	Pushdown list	A-19
Floating point arithmetic	A-16	Shift and rotate	A-19
Full word data transmission	A-17	Test, arithmetic	A-20
Half word data transmission	A-18	Test, logical	A-21

The terminology and notation used also vary somewhat from that in the body of the manual, as follows.

AC	The accumulator address in bits 9–12 of the instruction word (represented by $A$ in the instruction descriptions).
AC+1	The address one greater than AC, except that AC+1 is 0 if AC is 17.
E	The result of the effective address calculation. E is eighteen bits when used as an address, half word operand, mask or output conditions, but is a signed 9-bit quantity when used as a scale factor or a shift number.
E+1	The address one greater than E, except that E+1 is 0 if E is 777777.
PC	The 18-bit program counter.
( $X$ )	The word contained in register $X$ .
( $X$ ) <sub>L</sub>	The left half of ( $X$ ).
( $X$ ) <sub>R</sub>	The right half of ( $X$ ).
( $X$ ) <sub>S</sub>	The word contained in $X$ with its left and right halves swapped.
$A_n$	The value of bit $n$ of the quantity $A$ .
$A,B$	A 36-bit word with the 18-bit quantity $A$ in its left half and the 18-bit quantity $B$ in its right half (either $A$ or $B$ may be 0).
( $X,Y$ )	The contents of registers $X$ and $Y$ concatenated into a double word operand.
(( $X$ ))	The word contained in the register addressed by ( $X$ ), ie addressed by the word in register $X$ .
$A \rightarrow B$	The quantity $A$ replaces the quantity $B$ ( $A$ and $B$ may be half words, full words or double words). <i>Eg</i> $(AC) + (E) \rightarrow (AC)$ means the word in accumulator AC plus the word in memory location E replaces the word in AC.
(AC) (E)	The word in AC and the word in E.
$\wedge \vee \nabla \sim$	The Boolean operators AND, inclusive OR, exclusive OR, and complement (logical negation).

$+ - \times \div ||$  The arithmetic operators for addition, negation or subtraction, multiplication, division, and absolute value (magnitude).

Square brackets are used occasionally for grouping. With respect to the values of their terms, the equations for a given instruction are in chronological order; eg in the pair of equations

$$(AC) + 1 \rightarrow (AC)$$

$$If (AC) = 0: E \rightarrow (PC)$$

the quantity tested in the second equation is the word in AC after it has been incremented by one.



## Boolean

SETZ	400	$0 \rightarrow (AC)$	SETO	474	$777777777777 \rightarrow (AC)$
SETZI	401	$0 \rightarrow (AC)$	SETOI	475	$777777777777 \rightarrow (AC)$
SETZM	402	$0 \rightarrow (E)$	SETOM	476	$777777777777 \rightarrow (E)$
SETZB	403	$0 \rightarrow (AC) (E)$	SETOB	477	$777777777777 \rightarrow (AC) (E)$
SETA	424	$(AC) \rightarrow (AC)$ [ <i>no-op</i> ]	SETCA	450	$\sim (AC) \rightarrow (AC)$
SETAI	425	$(AC) \rightarrow (AC)$ [ <i>no-op</i> ]	SETCAI	451	$\sim (AC) \rightarrow (AC)$
SETAM	426	$(AC) \rightarrow (E)$	SETCAM	452	$\sim (AC) \rightarrow (E)$
SETAB	427	$(AC) \rightarrow (E)$	SETCAB	453	$\sim (AC) \rightarrow (AC) (E)$
SETM	414	$(E) \rightarrow (AC)$	SETCM	460	$\sim (E) \rightarrow (AC)$
SETMI	415	$0,E \rightarrow (AC)$	SETCMI	461	$\sim [0,E] \rightarrow (AC)$
SETMM	416	$(E) \rightarrow (E)$ [ <i>no-op</i> ]	SETCMM	462	$\sim (E) \rightarrow (E)$
SETMB	417	$(E) \rightarrow (AC) (E)$	SETCMB	463	$\sim (E) \rightarrow (AC) (E)$
AND	404	$(AC) \wedge (E) \rightarrow (AC)$	ANDCA	410	$\sim (AC) \wedge (E) \rightarrow (AC)$
ANDI	405	$(AC) \wedge 0,E \rightarrow (AC)$	ANDCAI	411	$\sim (AC) \wedge 0,E \rightarrow (AC)$
ANDM	406	$(AC) \wedge (E) \rightarrow (E)$	ANDCAM	412	$\sim (AC) \wedge (E) \rightarrow (E)$
ANDB	407	$(AC) \wedge (E) \rightarrow (AC) (E)$	ANDCAB	413	$\sim (AC) \wedge (E) \rightarrow (AC) (E)$
ANDCM	420	$(AC) \wedge \sim (E) \rightarrow (AC)$	ANDCB	440	$\sim (AC) \wedge \sim (E) \rightarrow (AC)$
ANDCMI	421	$(AC) \wedge \sim [0,E] \rightarrow (AC)$	ANDCBI	441	$\sim (AC) \wedge \sim [0,E] \rightarrow (AC)$
ANDCMM	422	$(AC) \wedge \sim (E) \rightarrow (E)$	ANDCBM	442	$\sim (AC) \wedge \sim (E) \rightarrow (E)$
ANDCMB	423	$(AC) \wedge \sim (E) \rightarrow (AC) (E)$	ANDCBB	443	$\sim (AC) \wedge \sim (E) \rightarrow (AC) (E)$
IOR	434	$(AC) \vee (E) \rightarrow (AC)$	ORCA	454	$\sim (AC) \vee (E) \rightarrow (AC)$
IORI	435	$(AC) \vee 0,E \rightarrow (AC)$	ORCAI	455	$\sim (AC) \vee 0,E \rightarrow (AC)$
IORM	436	$(AC) \vee (E) \rightarrow (E)$	ORCAM	456	$\sim (AC) \vee (E) \rightarrow (E)$
IORB	437	$(AC) \vee (E) \rightarrow (AC) (E)$	ORCAB	457	$\sim (AC) \vee (E) \rightarrow (AC) (E)$
ORCM	464	$(AC) \vee \sim (E) \rightarrow (AC)$	ORCB	470	$\sim (AC) \vee \sim (E) \rightarrow (AC)$
ORCMI	465	$(AC) \vee \sim [0,E] \rightarrow (AC)$	ORCBI	471	$\sim (AC) \vee \sim [0,E] \rightarrow (AC)$
ORCMM	466	$(AC) \vee \sim (E) \rightarrow (E)$	ORCBM	472	$\sim (AC) \vee \sim (E) \rightarrow (E)$
ORCMB	467	$(AC) \vee \sim (E) \rightarrow (AC) (E)$	ORCBB	473	$\sim (AC) \vee \sim (E) \rightarrow (AC) (E)$
XOR	430	$(AC) \nabla (E) \rightarrow (AC)$	EQV	444	$\sim [(AC) \nabla (E)] \rightarrow (AC)$
XORI	431	$(AC) \nabla 0,E \rightarrow (AC)$	EQVI	445	$\sim [(AC) \nabla 0,E] \rightarrow (AC)$
XORM	432	$(AC) \nabla (E) \rightarrow (E)$	EQVM	446	$\sim [(AC) \nabla (E)] \rightarrow (E)$
XORB	433	$(AC) \nabla (E) \rightarrow (AC) (E)$	EQVB	447	$\sim [(AC) \nabla (E)] \rightarrow (AC) (E)$

**Byte Manipulation**

IBP	133	<i>Operations on (E) [see page 2-16]</i> <i>If <math>P - S \geq 0</math>: <math>P - S \rightarrow P</math></i> <i>If <math>P - S &lt; 0</math>: <math>Y + 1 \rightarrow Y</math>      <math>36 - S \rightarrow P</math></i>
LDB	135	BYTE IN ((E)) $\rightarrow$ (AC) [see page 2-16]
DPB	137	BYTE IN (AC) $\rightarrow$ BYTE IN ((E)) [see page 2-16]
ILDDB	134	IBP and LDB
IDPB	136	IBP and DPB

**Fixed Point Arithmetic**

ADD	270	(AC) + (E) $\rightarrow$ (AC)	SUB	274	(AC) - (E) $\rightarrow$ (AC)
ADDI	271	(AC) + 0,E $\rightarrow$ (AC)	SUBI	275	(AC) - 0,E $\rightarrow$ (AC)
ADDM	272	(AC) + (E) $\rightarrow$ (E)	SUBM	276	(AC) - (E) $\rightarrow$ (E)
ADDB	273	(AC) + (E) $\rightarrow$ (AC) (E)	SUBB	277	(AC) - (E) $\rightarrow$ (AC) (E)
IMUL	220	(AC) $\times$ (E) $\rightarrow$ (AC)*	MUL	224	(AC) $\times$ (E) $\rightarrow$ (AC,AC+1)
IMULI	221	(AC) $\times$ 0,E $\rightarrow$ (AC)*	MULI	225	(AC) $\times$ 0,E $\rightarrow$ (AC,AC+1)
IMULM	222	(AC) $\times$ (E) $\rightarrow$ (E)*	MULM	226	(AC) $\times$ (E) $\rightarrow$ (E)†
IMULB	223	(AC) $\times$ (E) $\rightarrow$ (AC) (E)*	MULB	227	(AC) $\times$ (E) $\rightarrow$ (AC,AC+1) (E)
IDIV	230	(AC) $\div$ (E) $\rightarrow$ (AC) REMAINDER $\rightarrow$ (AC+1)	DIV	234	(AC,AC+1) $\div$ (E) $\rightarrow$ (AC) REMAINDER $\rightarrow$ (AC+1)
IDIVI	231	(AC) $\div$ 0,E $\rightarrow$ (AC) REMAINDER $\rightarrow$ (AC+1)	DIVI	235	(AC,AC+1) $\div$ 0,E $\rightarrow$ (AC) REMAINDER $\rightarrow$ (AC+1)
IDIVM	232	(AC) $\div$ (E) $\rightarrow$ (E)	DIVM	236	(AC,AC+1) $\div$ (E) $\rightarrow$ (E)
IDIVB	233	(AC) $\div$ (E) $\rightarrow$ (AC) (E) REMAINDER $\rightarrow$ (AC+1)	DIVB	237	(AC,AC+1) $\div$ (E) $\rightarrow$ (AC) (E) REMAINDER $\rightarrow$ (AC+1)

\*The high order word of the product is discarded.

†The low order word of the product is discarded.

**Floating Point Arithmetic**

FAD	140	(AC) + (E) $\rightarrow$ (AC)	FADR	144	(AC) + (E) $\rightarrow$ (AC)
FADL	141	(AC) + (E) $\rightarrow$ (AC,AC+1)	FADR1	145	(AC) + E,0 $\rightarrow$ (AC)
FADM	142	(AC) + (E) $\rightarrow$ (E)	FADRM	146	(AC) + (E) $\rightarrow$ (E)
FADB	143	(AC) + (E) $\rightarrow$ (AC) (E)	FADRB	147	(AC) + (E) $\rightarrow$ (AC) (E)
FSB	150	(AC) - (E) $\rightarrow$ (AC)	FSBR	154	(AC) - (E) $\rightarrow$ (AC)
FSBL	151	(AC) - (E) $\rightarrow$ (AC,AC+1)	FSBRI	155	(AC) - E,0 $\rightarrow$ (AC)
FSBM	152	(AC) - (E) $\rightarrow$ (E)	FSBRM	156	(AC) - (E) $\rightarrow$ (E)
FSBB	153	(AC) - (E) $\rightarrow$ (AC) (E)	FSBRB	157	(AC) - (E) $\rightarrow$ (AC) (E)

FMP	160	$(AC) \times (E) \rightarrow (AC)$	FMPR	164	$(AC) \times (E) \rightarrow (AC)$	
FMPL	161	$(AC) \times (E) \rightarrow (AC, AC+1)$	FMPRI	165	$(AC) \times E, 0 \rightarrow (AC)$	
FMPM	162	$(AC) \times (E) \rightarrow (E)$	FMPRM	166	$(AC) \times (E) \rightarrow (E)$	
FMPB	163	$(AC) \times (E) \rightarrow (AC) (E)$	FMPRB	167	$(AC) \times (E) \rightarrow (AC) (E)$	
FDV	170	$(AC) \div (E) \rightarrow (AC)$	FDVR	174	$(AC) \div (E) \rightarrow (AC)$	
FDVL	171	$(AC) \div (E) \rightarrow (AC)$ REMAINDER $\rightarrow (AC+1)$	FDVRI	175	$(AC) \div E, 0 \rightarrow (AC)$	
FDVM	172	$(AC) \div (E) \rightarrow (E)$	FDVRM	176	$(AC) \div (E) \rightarrow (E)$	
FDVB	173	$(AC) \div (E) \rightarrow (AC) (E)$	FDVRB	177	$(AC) \div (E) \rightarrow (AC) (E)$	
		UFA	130	$(AC) + (E) \rightarrow (AC+1)$	<i>without normalization</i>	
		DFN	131	$-(AC, E) \rightarrow (AC, E)$		
		FSC	132	$(AC) \times 2^E \rightarrow (AC)$		
		FLTR	127	$(E)$ floated, rounded	$\rightarrow (AC)$	
FIX	122	$(E)$ fixed			$\rightarrow (AC)$	
			FIXR	126	$(E)$ fixed, rounded	$\rightarrow (AC)$
		DFAD	110	$(AC, AC+1) + (E, E+1) \rightarrow (AC, AC+1)$		
		DFSB	111	$(AC, AC+1) - (E, E+1) \rightarrow (AC, AC+1)$		
		DFMP	112	$(AC, AC+1) \times (E, E+1) \rightarrow (AC, AC+1)$		
		DFDV	113	$(AC, AC+1) \div (E, E+1) \rightarrow (AC, AC+1)$		
DMOVE	120	$(E, E+1) \rightarrow (AC, AC+1)$	DMOVEM	124	$(AC, AC+1) \rightarrow (E, E+1)$	
DMOVN	121	$-(E, E+1) \rightarrow (AC, AC+1)$	DMOVNM	125	$-(AC, AC+1) \rightarrow (E, E+1)$	

## Full Word Data Transmission

EXCH	250	$(AC) \leftrightarrow (E)$			
BLT	251	Move $E - (AC)_R + 1$ words starting with $((AC)_L) \rightarrow ((AC)_R)$			[see page 2-10]
MOVE	200	$(E) \rightarrow (AC)$	MOVS	204	$(E)_S \rightarrow (AC)$
MOVEI	201	$0, E \rightarrow (AC)$	MOVSI	205	$E, 0 \rightarrow (AC)$
MOVEM	202	$(AC) \rightarrow (E)$	MOVSM	206	$(AC)_S \rightarrow (E)$
MOVES	203	If $AC \neq 0$ : $(E) \rightarrow (AC)$	MOVSS	207	$(E)_S \rightarrow (E)$ If $AC \neq 0$ : $(E) \rightarrow (AC)$
MOVN	210	$-(E) \rightarrow (AC)$	MOVMM	214	$ (E)  \rightarrow (AC)$
MOVNI	211	$-[0, E] \rightarrow (AC)$	MOVMI	215	$0, E \rightarrow (AC)$
MOVNM	212	$-(AC) \rightarrow (E)$	MOVMM	216	$ (AC)  \rightarrow (E)$
MOVNS	213	$-(E) \rightarrow (E)$ If $AC \neq 0$ : $(E) \rightarrow (AC)$	MOVMS	217	$ (E)  \rightarrow (E)$ If $AC \neq 0$ : $(E) \rightarrow (AC)$

## Half Word Data Transmission

HLL	500	$(E)_L \rightarrow (AC)_L$	HLLZ	510	$(E)_L, 0 \rightarrow (AC)$
HLLI	501	$0 \rightarrow (AC)_L$	HLLZI	511	$0 \rightarrow (AC)$
HLLM	502	$(AC)_L \rightarrow (E)_L$	HLLZM	512	$(AC)_L, 0 \rightarrow (E)$
HLLS	503	<i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>	HLLZS	513	$0 \rightarrow (E)_R$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>
HLLO	520	$(E)_L, 777777 \rightarrow (AC)$	HLLE	530	$(E)_L, [(E)_0 \times 777777] \rightarrow (AC)$
HLLOI	521	$0, 777777 \rightarrow (AC)$	HLLEI	531	$0 \rightarrow (AC)$
HLLOM	522	$(AC)_L, 777777 \rightarrow (E)$	HLLEM	532	$(AC)_L, [(AC)_0 \times 777777] \rightarrow (E)$
HLLOS	523	$777777 \rightarrow (E)_R$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>	HLLES	533	$(E)_0 \times 777777 \rightarrow (E)_R$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>
HLR	544	$(E)_L \rightarrow (AC)_R$	HLRZ	554	$0, (E)_L \rightarrow (AC)$
HLRI	545	$0 \rightarrow (AC)_R$	HLRZI	555	$0 \rightarrow (AC)$
HLRM	546	$(AC)_L \rightarrow (E)_R$	HLRZM	556	$0, (AC)_L \rightarrow (E)$
HLRS	547	$(E)_L \rightarrow (E)_R$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>	HLRZS	557	$0, (E)_L \rightarrow (E)$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>
HLRO	564	$777777, (E)_L \rightarrow (AC)$	HLRE	574	$[(E)_0 \times 777777], (E)_L \rightarrow (AC)$
HLROI	565	$777777, 0 \rightarrow (AC)$	HLREI	575	$0 \rightarrow (AC)$
HLROM	566	$777777, (AC)_L \rightarrow (E)$	HLREM	576	$[(AC)_0 \times 777777], (AC)_L \rightarrow (E)$
HLROS	567	$777777, (E)_L \rightarrow (E)$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>	HLRES	577	$[(E)_0 \times 777777], (E)_L \rightarrow (E)$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>
HRR	540	$(E)_R \rightarrow (AC)_R$	HRRZ	550	$0, (E)_R \rightarrow (AC)$
HRRI	541	$E \rightarrow (AC)_R$	HRRZI	551	$0, E \rightarrow (AC)$
HRRM	542	$(AC)_R \rightarrow (E)_R$	HRRZM	552	$0, (AC)_R \rightarrow (E)$
HRRS	543	<i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>	HRRZS	553	$0 \rightarrow (E)_L$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>
HRRO	560	$777777, (E)_R \rightarrow (AC)$	HRRE	570	$[(E)_{18} \times 777777], (E)_R \rightarrow (AC)$
HRROI	561	$777777, E \rightarrow (AC)$	HRREI	571	$[E_{18} \times 777777], E \rightarrow (AC)$
HRROM	562	$777777, (AC)_R \rightarrow (E)$	HRREM	572	$[(AC)_{18} \times 777777], (AC)_R \rightarrow (E)$
HRROS	563	$777777 \rightarrow (E)_L$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>	HRRES	573	$(E)_{18} \times 777777 \rightarrow (E)_L$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>
HRL	504	$(E)_R \rightarrow (AC)_L$	HRLZ	514	$(E)_R, 0 \rightarrow (AC)$
HRLI	505	$E \rightarrow (AC)_L$	HRLZI	515	$E, 0 \rightarrow (AC)$
HRLM	506	$(AC)_R \rightarrow (E)_L$	HRLZM	516	$(AC)_R, 0 \rightarrow (E)$
HRLS	507	$(E)_R \rightarrow (E)_L$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>	HRLZS	517	$(E)_R, 0 \rightarrow (E)$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>

HRLO	524	$(E)_R, 777777 \rightarrow (AC)$	HRLE	534	$(E)_R, [(E)_{18} \times 777777] \rightarrow (AC)$
HRLOI	525	$E, 777777 \rightarrow (AC)$	HRLEI	535	$E, [E_{18} \times 777777] \rightarrow (AC)$
HRLOM	526	$(AC)_R, 777777 \rightarrow (E)$	HRLEM	536	$(AC)_R, [(AC)_{18} \times 777777] \rightarrow (E)$
HRLOS	527	$(E)_R, 777777 \rightarrow (E)$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>	HRLES	537	$(E)_R, [(E)_{18} \times 777777] \rightarrow (E)$ <i>If AC <math>\neq</math> 0: <math>(E) \rightarrow (AC)</math></i>

**In-out**

CONO	70020	$E \rightarrow \text{COMMAND}$	CONSZ	70030	<i>If STATUS<sub>R</sub> <math>\wedge</math> E = 0: skip</i>
CONI	70024	$\text{STATUS} \rightarrow (E)$	CONSO	70034	<i>If STATUS<sub>R</sub> <math>\wedge</math> E <math>\neq</math> 0: skip</i>
DATAO	70014	$(E) \rightarrow \text{DATA}$	DATAI	70004	$\text{DATA} \rightarrow (E)$
BLKO	70010	$(E) + 1000001 \rightarrow (E)^*$	$((E)_R) \rightarrow \text{DATA}$	<i>[see page 2-83]</i>	
BLKI	70000	$(E) + 1000001 \rightarrow (E)^*$	$\text{DATA} \rightarrow ((E)_R)$	<i>[see page 2-83]</i>	

**Program Control**

JSR	264	$\text{FLAGS}, (\text{PC}) \rightarrow (E)$	$E + 1 \rightarrow (\text{PC})$
JSP	265	$\text{FLAGS}, (\text{PC}) \rightarrow (AC)$	$E \rightarrow (\text{PC})$
JRST	254	$E \rightarrow (\text{PC})$	<i>[If AC <math>\neq</math> 0, see page 2-63]</i>
JSA	266	$(AC) \rightarrow (E)$	$E, (\text{PC}) \rightarrow (AC)$ $E + 1 \rightarrow (\text{PC})$
JRA	267	$E \rightarrow (\text{PC})$	$((AC)_L) \rightarrow (AC)$
JFCL	255	<i>If AC <math>\wedge</math> FLAGS <math>\neq</math> 0:</i>	$E \rightarrow (\text{PC})$ $\sim AC \wedge \text{FLAGS} \rightarrow \text{FLAGS}$
XCT	256	<i>Execute (E)</i>	
JFFO	243	<i>If (AC) = 0: <math>0 \rightarrow (AC + 1)</math></i> <i>If (AC) <math>\neq</math> 0: <math>E \rightarrow (\text{PC})</math> [see page 2-61]</i>	
MAP	257	$\text{PHYSICAL MAP DATA} \rightarrow (AC)$	

**Pushdown List**

PUSH	261	$(AC) + 1000001 \rightarrow (AC)^*$	$(E) \rightarrow ((AC)_R)$
POP	262	$((AC)_R) \rightarrow (E)$	$(AC) - 1000001 \rightarrow (AC)^*$
PUSHJ	260	$(AC) + 1000001 \rightarrow (AC)^*$	$\text{FLAGS}, (\text{PC}) \rightarrow ((AC)_R)$ $E \rightarrow (\text{PC})$
POPJ	263	$((AC)_R)_R \rightarrow (\text{PC})$	$(AC) - 1000001 \rightarrow (AC)^*$

**Shift and Rotate**

ASH	240	$(AC) \times 2^E \rightarrow (AC)$	ASHC	244	$(AC, AC+1) \times 2^E \rightarrow (AC, AC+1)$
ROT	241	<i>Rotate (AC) E places</i>	ROTC	245	<i>Rotate (AC, AC+1) E places</i>
LSH	242	<i>Shift (AC) E places</i>	LSHC	246	<i>Shift (AC, AC+1) E places</i>

\*In the KI10, 1 is added to or subtracted from each half separately.

## Arithmetic Testing

AOBJP	252	$(AC) + 1000001 \rightarrow (AC)^*$	$If (AC) \geq 0: E \rightarrow (PC)$		
AOBJN	253	$(AC) + 1000001 \rightarrow (AC)^*$	$If (AC) < 0: E \rightarrow (PC)$		
CAI	300	<i>No-op</i>	CAM	310	<i>No-op</i>
CAIL	301	$If (AC) < E: skip$	CAML	311	$If (AC) < (E): skip$
CAIE	302	$If (AC) = E: skip$	CAME	312	$If (AC) = (E): skip$
CAILE	303	$If (AC) \leq E: skip$	CAMLE	313	$If (AC) \leq (E): skip$
CAIA	304	<i>Skip</i>	CAMA	314	<i>Skip</i>
CAIGE	305	$If (AC) \geq E: skip$	CAMGE	315	$If (AC) \geq (E): skip$
CAIN	306	$If (AC) \neq E: skip$	CAMN	316	$If (AC) \neq (E): skip$
CAIG	307	$If (AC) > E: skip$	CAMG	317	$If (AC) > (E): skip$
JUMP	320	<i>No-op</i>	SKIP	330	$If AC \neq 0: (E) \rightarrow (AC)$
JUMPL	321	$If (AC) < 0: E \rightarrow (PC)$	SKIPL	331	$If AC \neq 0: (E) \rightarrow (AC)$ $If (E) < 0: skip$
JUMPE	322	$If (AC) = 0: E \rightarrow (PC)$	SKIPE	332	$If AC \neq 0: (E) \rightarrow (AC)$ $If (E) = 0: skip$
JUMPLE	323	$If (AC) \leq 0: E \rightarrow (PC)$	SKIPLE	333	$If AC \neq 0: (E) \rightarrow (AC)$ $If (E) \leq 0: skip$
JUMPA	324	$E \rightarrow (PC)$	SKIPA	334	$If AC \neq 0: (E) \rightarrow (AC)$ <i>Skip</i>
JUMPGE	325	$If (AC) \geq 0: E \rightarrow (PC)$	SKIPGE	335	$If AC \neq 0: (E) \rightarrow (AC)$ $If (E) \geq 0: skip$
JUMPN	326	$If (AC) \neq 0: E \rightarrow (PC)$	SKIPN	336	$If AC \neq 0: (E) \rightarrow (AC)$ $If (E) \neq 0: skip$
JUMPG	327	$If (AC) > 0: E \rightarrow (PC)$	SKIPG	337	$If AC \neq 0: (E) \rightarrow (AC)$ $If (E) > 0: skip$
AOJ	340	$(AC) + 1 \rightarrow (AC)$	SOJ	360	$(AC) - 1 \rightarrow (AC)$
AOJL	341	$(AC) + 1 \rightarrow (AC)$ $If (AC) < 0: E \rightarrow (PC)$	SOJL	361	$(AC) - 1 \rightarrow (AC)$ $If (AC) < 0: E \rightarrow (PC)$
AOJE	342	$(AC) + 1 \rightarrow (AC)$ $If (AC) = 0: E \rightarrow (PC)$	SOJE	362	$(AC) - 1 \rightarrow (AC)$ $If (AC) = 0: E \rightarrow (PC)$
AOJLE	343	$(AC) + 1 \rightarrow (AC)$ $If (AC) \leq 0: E \rightarrow (PC)$	SOJLE	363	$(AC) - 1 \rightarrow (AC)$ $If (AC) \leq 0: E \rightarrow (PC)$
AOJA	344	$(AC) + 1 \rightarrow (AC)$ $E \rightarrow (PC)$	SOJA	364	$(AC) - 1 \rightarrow (AC)$ $E \rightarrow (PC)$
AOJGE	345	$(AC) + 1 \rightarrow (AC)$ $If (AC) \geq 0: E \rightarrow (PC)$	SOJGE	365	$(AC) - 1 \rightarrow (AC)$ $If (AC) \geq 0: E \rightarrow (PC)$

\*In the KI10, 1 is added to or subtracted from each half separately.

AOJN	346	$(AC) + 1 \rightarrow (AC)$ <i>If</i> $(AC) \neq 0$ : $E \rightarrow (PC)$	SOJN	366	$(AC) - 1 \rightarrow (AC)$ <i>If</i> $(AC) \neq 0$ : $E \rightarrow (PC)$
AOJG	347	$(AC) + 1 \rightarrow (AC)$ <i>If</i> $(AC) > 0$ : $E \rightarrow (PC)$	SOJG	367	$(AC) - 1 \rightarrow (AC)$ <i>If</i> $(AC) > 0$ : $E \rightarrow (PC)$
AOS	350	$(E) + 1 \rightarrow (E)$ <i>If</i> $(AC) \neq 0$ : $(E) \rightarrow (AC)$	SOS	370	$(E) - 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$
AOSL	351	$(E) + 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) < 0$ : <i>skip</i>	SOSL	371	$(E) - 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) < 0$ : <i>skip</i>
AOSE	352	$(E) + 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) = 0$ : <i>skip</i>	SOSE	372	$(E) - 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) = 0$ : <i>skip</i>
AOSLE	353	$(E) + 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) \leq 0$ : <i>skip</i>	SOSLE	373	$(E) - 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) \leq 0$ : <i>skip</i>
AOSA	354	$(E) + 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>Skip</i>	SOSA	374	$(E) - 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>Skip</i>
AOSGE	355	$(E) + 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) \geq 0$ : <i>skip</i>	SOSGE	375	$(E) - 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) \geq 0$ : <i>skip</i>
AOSN	356	$(E) + 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) \neq 0$ : <i>skip</i>	SOSN	376	$(E) - 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) \neq 0$ : <i>skip</i>
AOSG	357	$(E) + 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) > 0$ : <i>skip</i>	SOSG	377	$(E) - 1 \rightarrow (E)$ <i>If</i> $AC \neq 0$ : $(E) \rightarrow (AC)$ <i>If</i> $(E) > 0$ : <i>skip</i>

## Logical Testing and Modification

TLN	601	<i>No-op</i>	TRN	600	<i>No-op</i>
TLNE	603	<i>If</i> $(AC)_L \wedge E = 0$ : <i>skip</i>	TRNE	602	<i>If</i> $(AC)_R \wedge E = 0$ : <i>skip</i>
TLNA	605	<i>Skip</i>	TRNA	604	<i>Skip</i>
TLNN	607	<i>If</i> $(AC)_L \wedge E \neq 0$ : <i>skip</i>	TRNN	606	<i>If</i> $(AC)_R \wedge E \neq 0$ : <i>skip</i>
TLZ	621	$(AC)_L \wedge \sim E \rightarrow (AC)_L$	TRZ	620	$(AC)_R \wedge \sim E \rightarrow (AC)_R$
TLZE	623	<i>If</i> $(AC)_L \wedge E = 0$ : <i>skip</i> $(AC)_L \wedge \sim E \rightarrow (AC)_L$	TRZE	622	<i>If</i> $(AC)_R \wedge E = 0$ : <i>skip</i> $(AC)_R \wedge \sim E \rightarrow (AC)_R$
TLZA	625	$(AC)_L \wedge \sim E \rightarrow (AC)_L$ <i>skip</i>	TRZA	624	$(AC)_R \wedge \sim E \rightarrow (AC)_R$ <i>skip</i>
TLZN	627	<i>If</i> $(AC)_L \wedge E \neq 0$ : <i>skip</i> $(AC)_L \wedge \sim E \rightarrow (AC)_L$	TRZN	626	<i>If</i> $(AC)_R \wedge E \neq 0$ : <i>skip</i> $(AC)_R \wedge \sim E \rightarrow (AC)_R$

TLC	641	$(AC)_L \forall E \rightarrow (AC)_L$	TRC	640	$(AC)_R \forall E \rightarrow (AC)_R$
TLCE	643	<i>If</i> $(AC)_L \wedge E = 0$ : <i>skip</i> $(AC)_L \forall E \rightarrow (AC)_L$	TRCE	642	<i>If</i> $(AC)_R \wedge E = 0$ : <i>skip</i> $(AC)_R \forall E \rightarrow (AC)_R$
TLCA	645	$(AC)_L \forall E \rightarrow (AC)_L$ <i>skip</i>	TRCA	644	$(AC)_R \forall E \rightarrow (AC)_R$ <i>skip</i>
TLCN	647	<i>If</i> $(AC)_L \wedge E \neq 0$ : <i>skip</i> $(AC)_L \forall E \rightarrow (AC)_L$	TRCN	646	<i>If</i> $(AC)_R \wedge E \neq 0$ : <i>skip</i> $(AC)_R \forall E \rightarrow (AC)_R$
TLO	661	$(AC)_L \vee E \rightarrow (AC)_L$	TRO	660	$(AC)_R \vee E \rightarrow (AC)_R$
TLOE	663	<i>If</i> $(AC)_L \wedge E = 0$ : <i>skip</i> $(AC)_L \vee E \rightarrow (AC)_L$	TROE	662	<i>If</i> $(AC)_R \wedge E = 0$ : <i>skip</i> $(AC)_R \vee E \rightarrow (AC)_R$
TLOA	665	$(AC)_L \vee E \rightarrow (AC)_L$ <i>skip</i>	TROA	664	$(AC)_R \vee E \rightarrow (AC)_R$ <i>skip</i>
TLON	667	<i>If</i> $(AC)_L \wedge E \neq 0$ : <i>skip</i> $(AC)_L \vee E \rightarrow (AC)_L$	TRON	666	<i>If</i> $(AC)_R \wedge E \neq 0$ : <i>skip</i> $(AC)_R \vee E \rightarrow (AC)_R$
TDN	610	<i>No-op</i>	TSN	611	<i>No-op</i>
TDNE	612	<i>If</i> $(AC) \wedge (E) = 0$ : <i>skip</i>	TSNE	613	<i>If</i> $(AC) \wedge (E)_S = 0$ : <i>skip</i>
TDNA	614	<i>Skip</i>	TSNA	615	<i>Skip</i>
TDNN	616	<i>If</i> $(AC) \wedge (E) \neq 0$ : <i>skip</i>	TSNN	617	<i>If</i> $(AC) \wedge (E)_S \neq 0$ : <i>skip</i>
TDZ	630	$(AC) \wedge \sim (E) \rightarrow (AC)$	TSZ	631	$(AC) \wedge \sim (E)_S \rightarrow (AC)$
TDZE	632	<i>If</i> $(AC) \wedge (E) = 0$ : <i>skip</i> $(AC) \wedge \sim (E) \rightarrow (AC)$	TSZE	633	<i>If</i> $(AC) \wedge (E)_S = 0$ : <i>skip</i> $(AC) \wedge \sim (E)_S \rightarrow (AC)$
TDZA	634	$(AC) \wedge \sim (E) \rightarrow (AC)$ <i>skip</i>	TSZA	635	$(AC) \wedge \sim (E)_S \rightarrow (AC)$ <i>skip</i>
TDZN	636	<i>If</i> $(AC) \wedge (E) \neq 0$ : <i>skip</i> $(AC) \wedge \sim (E) \rightarrow (AC)$	TSZN	637	<i>If</i> $(AC) \wedge (E)_S \neq 0$ : <i>skip</i> $(AC) \wedge \sim (E)_S \rightarrow (AC)$
TDC	650	$(AC) \forall (E) \rightarrow (AC)$	TSC	651	$(AC) \forall (E)_S \rightarrow (AC)$
TDCE	652	<i>If</i> $(AC) \wedge (E) = 0$ : <i>skip</i> $(AC) \forall (E) \rightarrow (AC)$	TSCE	653	<i>If</i> $(AC) \wedge (E)_S = 0$ : <i>skip</i> $(AC) \forall (E)_S \rightarrow (AC)$
TDCA	654	$(AC) \forall (E) \rightarrow (AC)$ <i>skip</i>	TSCA	655	$(AC) \forall (E)_S \rightarrow (AC)$ <i>skip</i>
TDCN	656	<i>If</i> $(AC) \wedge (E) \neq 0$ : <i>skip</i> $(AC) \forall (E) \rightarrow (AC)$	TSCN	657	<i>If</i> $(AC) \wedge (E)_S \neq 0$ : <i>skip</i> $(AC) \forall (E)_S \rightarrow (AC)$
TDO	670	$(AC) \vee (E) \rightarrow (AC)$	TSO	671	$(AC) \vee (E)_S \rightarrow (AC)$
TDOE	672	<i>If</i> $(AC) \wedge (E) = 0$ : <i>skip</i> $(AC) \vee (E) \rightarrow (AC)$	TSOE	673	<i>If</i> $(AC) \wedge (E)_S = 0$ : <i>skip</i> $(AC) \vee (E)_S \rightarrow (AC)$
TDOA	674	$(AC) \vee (E) \rightarrow (AC)$ <i>skip</i>	TSOA	675	$(AC) \vee (E)_S \rightarrow (AC)$ <i>skip</i>
TDON	676	<i>If</i> $(AC) \wedge (E) \neq 0$ : <i>skip</i> $(AC) \vee (E) \rightarrow (AC)$	TSON	677	<i>If</i> $(AC) \wedge (E)_S \neq 0$ : <i>skip</i> $(AC) \vee (E)_S \rightarrow (AC)$



POWERS OF TWO

$2^N$   $N$   $2^{-N}$

1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.015625
128	7	0.0078125
256	8	0.00390625
512	9	0.001953125
1024	10	0.0009765625
2048	11	0.00048828125
4096	12	0.000244140625
8192	13	0.0001220703125
16384	14	0.00006103515625
32768	15	0.000030517578125
65536	16	0.0000152587890625
131072	17	0.00000762939453125
262144	18	0.000003814697265625
524288	19	0.0000019073486328125
1048576	20	0.00000095367431640625
2097152	21	0.000000476837158203125
4194304	22	0.0000002384185791015625
8388608	23	0.00000011920928955078125
16777216	24	0.000000059604644775390625
33554432	25	0.0000000298023223876953125
67108864	26	0.00000001490116119384765625
134217728	27	0.000000007450580596923828125
268435456	28	0.0000000037252902984619140625
536870912	29	0.00000000186264514923095703125
1073741824	30	0.000000000931322574615478515625
2147483648	31	0.0000000004656612873077392578125
4294967296	32	0.00000000023283064365386962890625
8589934592	33	0.000000000116415321826934814453125
17179869184	34	0.0000000000582076609134674072265625
34359738368	35	0.00000000002910383045673370361328125
68719476736	36	0.000000000014551915228366851806640625
137438953472	37	0.0000000000072759576141834259033203125
274877906944	38	0.00000000000363797880709171295166015625
549755813888	39	0.00000000000181898940354585647583078125
1099511627776	40	0.0000000000009094947017729282379150390625
2199023255552	41	0.00000000000045474735088646411895751953125
4398046511104	42	0.000000000000227373675443232059478759765625
8796093022208	43	0.000000000000113686837216160297393798828125
17592186044416	44	0.00000000000005684341886080801486968994140625
35184372088832	45	0.000000000000028421709430404007434844970703125
70368744177664	46	0.0000000000000142108547152020037174224853515625
140737488355328	47	0.00000000000000710542735760100185871124267578125
281474976710656	48	0.000000000000003552713678800500929355621337890625
562949953421312	49	0.0000000000000017763568394002504646778106689453125
1125899906842624	50	0.00000000000000088817841970012523233890533447265625
2251799813685248	51	0.000000000000000444089209850062616169452667236328125
4503599627370496	52	0.0000000000000002220446049250313080847263336181640625
9007199254740992	53	0.00000000000000011102230246251565404236316680908203125
18014398509481984	54	0.000000000000000055511151231257827021181583404541015625
36028797018963968	55	0.000000000000000027755575615628913510590791702270578125
72057594037927936	56	0.00000000000000001387778780781445675529539585113525390625
144115188075855872	57	0.000000000000000006938893903907228377647697925567626953125
288230376151711744	58	0.0000000000000000034694469519536141888238489627838134765625
576460752303423488	59	0.00000000000000000173472347597680709441192448139190673828125
1152921504606846976	60	0.000000000000000000867361737988403547205962240695953369140625
2305843009213693952	61	0.0000000000000000004336808689942017736029811203479766845703125
4611686018427387904	62	0.00000000000000000021684043449710088680149056017398834228515625
9223372036854775808	63	0.000000000000000000108420217248550443400745280086994171142578125
18446744073709551616	64	0.0000000000000000000542101086242752217003726400434970855712890625
36893488147419103232	65	0.00000000000000000002710505431213761085018632002174854278564453125
73786976294838206464	66	0.000000000000000000013552527156068805425093160010874271392822265625
147573952589676412928	67	0.000000000000000000006776263578034402712546580005437135696411328125
295147905179352825856	68	0.00000000000000000000338813178901720135627329000271856784820556640625
590295810358705651712	69	0.000000000000000000001694065894508600678136645001359283924102783203125
1180591620717411303424	70	0.0000000000000000000008470329472543003390683225006796419620513916015625
2361183241434822606848	71	0.00000000000000000000042351647362715016953416125033982098102569580078125
4722366482869645213696	72	0.000000000000000000000211758236813575084767080625169910490512847900390625



## APPENDIX B

### INPUT-OUTPUT CODES

The table beginning on the next page lists the complete 1968 ASCII code (ANSI X3.4-1968). The software handles the full character set, and for a program that does not handle lower case, it translates input codes 140-174 into the corresponding upper case codes (100-134) and translates both 175 and 176 into 033, escape. The actual character sets available on different terminals vary greatly, but usually a terminal without lower case will accept lower case codes, printing the corresponding upper case character. The definitions of the control codes are those given by ASCII; most control codes, however, have no effect on the console terminal, and the definitions bear no necessary relation to the use of the codes in conjunction with the DECsystem-10 software. Brackets enclose earlier definitions of control codes (mostly 1963 ASCII). The table includes bit 8 as an even parity bit, the form generally used for paper tape and asynchronous operations; odd parity is generally used for magnetic tape and synchronous operations.

With all line printers, ten control characters are used for format control, and the interface also recognizes null for fill and delete for selecting hidden characters. The 64-character print set includes the figures and upper case; lower case is added for the 96-character set (with the smaller print set, giving a lower case code prints the upper case character). The larger print set includes a character hidden under delete, a feature that is optional on the LP10D and H. The printable characters are generally those defined by ASCII, with little if any variation. The 128-character printer uses the entire set of 7-bit codes for printable characters, with characters hidden under the ten control codes that affect the printer and also under null and delete.

The first two pages of the table of card codes [*pages B-8 to B-12*] list the column punches required to represent characters in the ASCII card code. When reading cards, the software translates the column punch into the octal code shown; when punching cards, it produces the listed column punch when given the corresponding code. There are also a few control hole patterns that the software responds to but does not translate. The next page lists two earlier DEC card codes that have only the figure and upper case character subset, plus a few control punches. The remaining pages of the table show the relationship among the early DEC card codes, the corresponding characters in the ASCII set, and several IBM card punches. Each column punch is produced by a single key on any keypunch for which a character is listed, the character being that which is printed at the top of the card.

Output codes are simply passed on to the terminal as they are, with the expectation that the terminal will ignore irrelevant control codes, and that a terminal that lacks lower case will print the corresponding upper case. A terminal that fails to live up to these assumptions will generally not operate satisfactorily with the DECsystem-10 software.

## ASCII CODE

Even Parity Bit	7-Bit Decimal	7-Bit Octal	Character	Remarks
0	000	000	NUL	Null, tape feed. Control shift P.
1	001	001	SOH	Start of heading [SOM, start of message]. Control A.
1	002	002	STX	Start of text [EOA, end of address]. Control B.
0	003	003	ETX	End of text [EOM, end of message]. Control C.
1	004	004	EOT	End of transmission; shuts off TWX machines and disconnects some data sets. Control D.
0	005	005	ENQ	Enquiry [WRU, "Who are you?"]. Triggers identification ("Here is . . .") at remote station if so equipped. Control E.
0	006	006	ACK	Acknowledge [RU, "Are you . . .?"]. Control F.
1	007	007	BEL	Rings the bell. Control G.
1	008	010	BS	Backspace. Control H.
0	009	011	HT	Horizontal tab. Control I.
0	010	012	LF	Line feed. Control J.
1	011	013	VT	Vertical tab. Control K.
0	012	014	FF	Form feed to top of next page. Control L.
1	013	015	CR	Carriage return to beginning of line. Control M.
1	014	016	SO	Shift out; change character set or change ribbon color to red. Control N.
0	015	017	SI	Shift in; return to standard character set or color. Control O.
1	016	020	DLE	Data link escape [DCO]. Control P.
0	017	021	DC1	Device control 1, turns transmitter (reader) on. Control Q (X ON).
0	018	022	DC2	Device control 2, turns punch or auxiliary on. Control R (TAPE, AUX ON).
1	019	023	DC3	Device control 3, turns transmitter (reader) off. Control S (X OFF).
0	020	024	DC4	Device control 4 (stop), turns punch or auxiliary off. Control T (TAPE, AUX OFF).
1	021	025	NAK	Negative acknowledge [ERR, error]. Control U.
1	022	026	SYN	Synchronous idle [SYNC]. Control V.
0	023	027	ETB	End of transmission block [LEM, logical end of medium]. Control W.
0	024	030	CAN	Cancel [S <sub>0</sub> ]. Control X.
1	025	031	EM	End of medium [S <sub>1</sub> ]. Control Y.
1	026	032	SUB	Substitute [S <sub>2</sub> ]. Control Z.
0	027	033	ESC	Escape, prefix [S <sub>3</sub> ]. Control shift K.
1	028	034	FS	File separator [S <sub>4</sub> ]. Control shift L.
0	029	035	GS	Group separator [S <sub>5</sub> ]. Control shift M.
0	030	036	RS	Record separator [S <sub>6</sub> ]. Control shift N.
1	031	037	US	Unit separator [S <sub>7</sub> ]. Control shift O.

Figures				Upper Case				Lower Case			
Even Parity Bit	7-Bit Decimal	7-Bit Octal	Character	Even Parity Bit	7-Bit Decimal	7-Bit Octal	Character	Even Parity Bit	7-Bit Decimal	7-Bit Octal	Character <sup>10</sup>
1	032	040	SP <sup>1</sup>	1	064	100	@ <sup>4</sup>	0	096	140	~ <sup>11</sup>
0	033	041	!	0	065	101	A	1	097	141	a
0	034	042	"	0	066	102	B	1	098	142	b
1	035	043	# <sup>2</sup>	1	067	103	C	0	099	143	c
0	036	044	\$	0	068	104	D	1	100	144	d
1	037	045	%	1	069	105	E	0	101	145	e
1	038	046	&	1	070	106	F	0	102	146	f
0	039	047	^ <sup>3</sup>	0	071	107	G	1	103	147	g
0	040	050	(	0	072	110	H	1	104	150	h
1	041	051	)	1	073	111	I	0	105	151	i
1	042	052	*	1	074	112	J	0	106	152	j
0	043	053	+	0	075	113	K	1	107	153	k
1	044	054	,	1	076	114	L	0	108	154	l
0	045	055	-	0	077	115	M	1	109	155	m
0	046	056	.	0	078	116	N	1	110	156	n
1	047	057	/	1	079	117	O	0	111	157	o
0	048	060	∅	0	080	120	P	1	112	160	p
1	049	061	1	1	081	121	Q	0	113	161	q
1	050	062	2	1	082	122	R	0	114	162	r
0	051	063	3	0	083	123	S	1	115	163	s
1	052	064	4	1	084	124	T	0	116	164	t
0	053	065	5	0	085	125	U	1	117	165	u
0	054	066	6	0	086	126	V	1	118	166	v
1	055	067	7	1	087	127	W	0	119	167	w
1	056	070	8	1	088	130	X	0	120	170	x
0	057	071	9	0	089	131	Y	1	121	171	y
0	058	072	:	0	090	132	Z	1	122	172	z
1	059	073	;	1	091	133	[ <sup>5</sup>	0	123	173	{
0	060	074	<	0	092	134	\ <sup>6</sup>	1	124	174	<sup>12</sup>
1	061	075	=	1	093	135	] <sup>7</sup>	0	125	175	} <sup>13</sup>
1	062	076	>	1	094	136	^ <sup>8</sup>	0	126	176	~ <sup>14</sup>
0	063	077	?	0	095	137	_ <sup>9</sup>	1	127	177	DEL <sup>15</sup>

<sup>1</sup>Space.

<sup>2</sup>£ on some (non-DEC) units.

<sup>3</sup>Accent acute or apostrophe – ' before 1965, but used until recently on DEC units.

<sup>4</sup>^ 1965–67, but never on DEC units.

<sup>5</sup>Shift K.

<sup>6</sup>~ 1965–67, but never on DEC units. Shift L.

<sup>7</sup>Shift M.

<sup>8</sup>Circumflex – ^ before 1965, but used until recently on DEC units.

<sup>9</sup>Underscore – \_ before 1965, but used until recently on DEC units.

<sup>10</sup>Codes 140–173 first defined in 1965. For a full ASCII character set the Monitor accepts codes 140–176 as lower case. For a character set that lacks lower case, the Monitor translates input codes 140–174 into the corre-

sponding upper case codes (100–134) and translates both 175 and 176 into 033, escape. Early versions of the Monitor used 175 as the escape code and translated both 176 and 033 to it.

<sup>11</sup>Accent grave – @ 1965–67, but never on DEC units.

<sup>12</sup>Control character ACK before 1965; | 1965–67, but never on DEC units. Vertical bar may or may not have gap depending on font design, but generally does not on DEC units.

<sup>13</sup>Unassigned control character (usually ALT MODE) before 1965. Code generated by ALT MODE key on most DEC units.

<sup>14</sup>Control character ESC before 1965; | 1965–67, but never on DEC units. Code generated by ESC key on some DEC units.

<sup>15</sup>Delete, rub out (not part of lower case set).

**LINE PRINTER CODE: LP10A, B, C, D, E**  
**Basic Character Set**

Control				Figures				Upper Case			
Hex	Decimal	Octal	Character	Hex	Decimal	Octal	Character	Hex	Decimal	Octal	Character
09	009	011	HT	20	032	040	SP	40	064	100	@
0A	010	012	LF	21	033	041	!	41	065	101	A
0B	011	013	VT	22	034	042	"	42	066	102	B
0C	012	014	FF	23	035	043	#	43	067	103	C
0D	013	015	CR	24	036	044	\$	44	068	104	D
				25	037	045	%	45	069	105	E
10	016	020	DLE	26	038	046	&	46	070	106	F
11	017	021	DC1	27	039	047	'	47	071	107	G
12	018	022	DC2	28	040	050	(	48	072	110	H
13	019	023	DC3	29	041	051	)	49	073	111	I
14	020	024	DC4	2A	042	052	*	4A	074	112	J
				2B	043	053	+	4B	075	113	K
00	000	000	NUL	2C	044	054	,	4C	076	114	L
7F	127	177	DEL	2D	045	055	-	4D	077	115	M
				2E	046	056	.	4E	078	116	N
				2F	047	057	/	4F	079	117	O
				30	048	060	0	50	080	120	P
				31	049	061	1	51	081	121	Q
				32	050	062	2	52	082	122	R
				33	051	063	3	53	083	123	S
				34	052	064	4	54	084	124	T
				35	053	065	5	55	085	125	U
				36	054	066	6	56	086	126	V
				37	055	067	7	57	087	127	W
				38	056	070	8	58	088	130	X
				39	057	071	9	59	089	131	Y
				3A	058	072	:	5A	090	132	Z
				3B	059	073	;	5B	091	133	[
				3C	060	074	<	5C	092	134	\
				3D	061	075	=	5D	093	135	]
				3E	062	076	>	5E	094	136	↑
				3F	063	077	?	5F	095	137	←

## Additional Characters – 95, 96 and 128 Character Sets

LP10D, E				LP10E			
Hex	Decimal	Octal	Character	Hex	Decimal	Octal	Character
60	096	140	˘	7F/00	127/000	177/000	• <i>NUL</i>
61	097	141	a	01	001	001	↓ <i>SOH</i>
62	098	143	b	02	002	002	α <i>STX</i>
63	099	143	c	03	003	003	β <i>ETX</i>
64	100	144	d	04	004	004	Λ <i>EOT</i>
65	101	145	e	05	005	005	¬ <i>ENQ</i>
66	102	146	f	06	006	006	ε <i>ACK</i>
67	103	147	g	07	007	007	π <i>BEL</i>
68	104	150	h	08	008	010	λ <i>BS</i>
69	105	151	i	7F/09	127/009	177/011	γ <i>HT</i>
6A	106	152	j	7F/0A	127/010	177/012	δ <i>LF</i>
6B	107	153	k	7F/0B	127/011	177/013	∫ <i>VT</i>
6C	108	154	l	7F/0C	127/012	177/014	± <i>FF</i>
6D	109	155	m	7F/0D	127/013	177/015	⊕ <i>CR</i>
6E	110	156	n	0E	014	016	∞ <i>SO</i>
6F	111	157	o	0F	015	017	∂ <i>SI</i>
70	112	160	p	7F/10	127/016	177/020	⊂ <i>DLE</i>
71	113	161	q	7F/11	127/017	177/021	⊃ <i>DC1</i>
72	114	162	r	7F/12	127/018	177/022	∩ <i>DC2</i>
73	115	163	s	7F/13	127/019	177/023	∪ <i>DC3</i>
74	116	164	t	7F/14	127/020	177/024	∇ <i>DC4</i>
75	117	165	u	15	021	025	∃ <i>NAK</i>
76	118	166	v	16	022	026	⊗ <i>SYN</i>
77	119	167	w	17	023	027	↔ <i>ETB</i>
78	120	170	x	18	024	030	^ <i>CAN</i>
79	121	171	y	19	025	031	→ <i>EM</i>
7A	122	172	z	1A	026	032	_ <i>SUB</i>
7B	123	173	{	1B	027	033	≠ <i>ESC</i>
7C	124	174		1C	028	034	≤ <i>FS</i>
7D	125	175	}	1D	029	035	≥ <i>GS</i>
7E	126	176	~	1E	030	036	≡ <i>RS</i>
7F/7F	127/127	177/177	␣ <i>DEL</i>	1F	031	037	√ <i>US</i>

Code pairs indicate hidden characters. For characters after the 95th, corresponding ASCII control characters are given in italics to facilitate generating codes at a keyboard.

**LINE PRINTER CODE: LP10F and H**  
**Basic Character Set**

Control				Figures				Upper Case			
Hex	Decimal	Octal	Character	Hex	Decimal	Octal	Character	Hex	Decimal	Octal	Character
09	009	011	HT	20	032	040	SP	40	064	100	@
0A	010	012	LF	21	033	041	!	41	065	101	A
0B	011	013	VT	22	034	042	"	42	066	102	B
0C	012	014	FF	23	035	043	#	43	067	103	C
0D	013	015	CR	24	036	044	\$	44	068	104	D
				25	037	045	%	45	069	105	E
10	016	020	DLE	26	038	046	&	46	070	106	F
11	017	021	DC1	27	039	047	'	47	071	107	G
12	018	022	DC2	28	040	050	(	48	072	110	H
13	019	023	DC3	29	041	051	)	49	073	111	I
14	020	024	DC4	2A	042	052	*	4A	074	112	J
				2B	043	053	+	4B	075	113	K
00	000	000	NUL	2C	044	054	,	4C	076	114	L
7F	127	177	DEL	2D	045	055	-	4D	077	115	M
				2E	046	056	.	4E	078	116	N
				2F	047	057	/	4F	079	117	O
				30	048	060	0 [Ø]	50	080	120	P
				31	049	061	1	51	081	121	Q
				32	050	062	2	52	082	122	R
				33	051	063	3	53	083	123	S
				34	052	064	4	54	084	124	T
				35	053	065	5	55	085	125	U
				36	054	066	6	56	086	126	V
				37	055	067	7	57	087	127	W
				38	056	070	8	58	088	130	X
				39	057	071	9	59	089	131	Y
				3A	058	072	:	5A	090	132	Z [Z]
				3B	059	073	;	5B	091	133	[
				3C	060	074	<	5C	092	134	\
				3D	061	075	=	5D	093	135	]
				3E	062	076	>	5E	094	136	^
				3F	063	077	?	5F	095	137	_

Table gives EDP character set of LP10FE and HE. Brackets enclose substitutions for scientific set, LP10FF and HF.



## Additional Characters – LP10H

Hex	Decimal	Octal	Character	Hex	Decimal	Octal	Character
60	096	140		70	112	160	p
61	097	141	a	71	113	161	q
62	098	142	b	72	114	162	r
63	099	143	c	73	115	163	s
64	100	144	d	74	116	164	t
65	101	145	e	75	117	165	u
66	102	146	f	76	118	166	v
67	103	147	g	77	119	167	w
68	104	150	h	78	120	170	x
69	105	151	i	79	121	171	y
6A	106	152	j	7A	122	172	z
6B	107	153	k	7B	123	173	{
6C	108	154	l	7C	124	174	
6D	109	155	m	7D	125	175	}
6E	110	156	n	7E	126	176	~
6F	111	157	o	7F/7F	127/127	177/177	← DEL

Character with code 177 is hidden under delete.

## ASCII CARD CODE

Octal	Character	Column Punch	Octal	Character	Column Punch	Octal	Character	Column Punch	Octal	Character	Column Punch
000	NUL	12 0 9 8 1	040	SP	<i>None</i>	100	@	8 4	140	~	8 1
001	SOH	12 9 1	041	!	12 8 7	101	A	12 1	141	a	12 0 1
002	STX	12 9 2	042	"	8 7	102	B	12 2	142	b	12 0 2
003	ETX	12 9 3	043	#	8 3	103	C	12 3	143	c	12 0 3
004	EOT	9 7	044	\$	11 8 3	104	D	12 4	144	d	12 0 4
005	ENQ	0 9 8 5	045	%	0 8 4	105	E	12 5	145	e	12 0 5
006	ACK	0 9 8 6	046	&	12	106	F	12 6	146	f	12 0 6
007	BEL	0 9 8 7	047	'	8 5	107	G	12 7	147	g	12 0 7
010	BS	11 9 6	050	(	12 8 5	110	H	12 8	150	h	12 0 8
011	HT	12 9 6	051	)	11 8 5	111	I	12 9	151	i	12 0 9
012	LF	0 9 5	052	*	11 8 4	112	J	11 1	152	j	12 11 1
013	VT	12 9 8 3	053	+	12 8 6	113	K	11 2	153	k	12 11 2
014	FF	12 9 8 4	054	,	0 8 3	114	L	11 3	154	l	12 11 3
015	CR	12 9 8 5	055	-	11	115	M	11 4	155	m	12 11 4
016	SO	12 9 8 6	056	.	12 8 3	116	N	11 5	156	n	12 11 5
017	SI	12 9 8 7	057	/	0 1	117	O	11 6	157	o	12 11 6
020	DLE	12 11 9 8 1	060	0	0	120	P	11 7	160	p	12 11 7
021	DC1	11 9 1	061	1	1	121	Q	11 8	161	q	12 11 8
022	DC2	11 9 2	062	2	2	122	R	11 9	162	r	12 11 9
023	DC3	11 9 3	063	3	3	123	S	0 2	163	s	11 0 2
024	DC4	9 8 4	064	4	4	124	T	0 3	164	t	11 0 3
025	NAK	9 8 5	065	5	5	125	U	0 4	165	u	11 0 4
026	SYN	9 2	066	6	6	126	V	0 5	166	v	11 0 5
027	ETB	0 9 6	067	7	7	127	W	0 6	167	w	11 0 6
030	CAN	11 9 8	070	8	8	130	X	0 7	170	x	11 0 7
031	EM	11 9 8 1	071	9	9	131	Y	0 8	171	y	11 0 8
032	SUB	9 8 7	072	:	8 2	132	Z	0 9	172	z	11 0 9
033	ESC	0 9 7	073	;	11 8 6	133	[	12 8 2	173	{	12 0
034	FS	11 9 8 4	074	<	12 8 4	134	\	0 8 2	174		12 11
035	GS	11 9 8 5	075	=	8 6	135	]	11 8 2	175	}	11 0
036	RS	11 9 8 6	076	>	0 8 6	136	^	11 8 7	176	~	11 0 1
037	US	11 9 8 7	077	?	0 8 7	137	_	0 8 5	177	DEL	12 9 7

When reading or punching cards, the software translates between the octal codes and column punches listed here. The software also recognizes the following control punches.

*Binary*           7 9  
*Mode Switch*    12 0 2 4 6 8  
*End of File*     12 11 0 1 6 7 8 9

Column Punch	Character	Column Punch	Character	Column Punch	Character	Column Punch	Character
None	SP	11 8	Q	12 0 4	d	11 8 3	\$
12	&	11 9	R	12 0 5	e	11 8 4	*
11	-	0 1	/	12 0 6	f	11 8 5	)
0	0	0 2	S	12 0 7	g	11 8 6	;
1	1	0 3	T	12 0 8	h	11 8 7	^
2	2	0 4	U	12 0 9	i	0 9 5	LF
3	3	0 5	V	12 9 1	SOH	0 9 6	ETB
4	4	0 6	W	12 9 2	STX	0 9 7	ESC
5	5	0 7	X	12 9 3	ETX	0 8 2	\
6	6	0 8	Y	12 9 5	HT	0 8 3	,
7	7	0 9	Z	12 9 7	DEL	0 8 4	%
8	8	9 2	SYN	12 8 2	[	0 8 5	_
9	9	9 7	EOT	12 8 3	.	0 8 6	>
12 11	:	8 1	~	12 8 4	<	0 8 7	?
12 0	{	8 2	:	12 8 5	(	9 8 4	DC4
12 1	A	8 3	#	12 8 6	+	9 8 5	NAK
12 2	B	8 4	@	12 8 7	!	9 8 7	SUB
12 3	C	8 5	'	11 0 1	~	12 9 8 3	VT
12 4	D	8 6	=	11 0 2	s	12 9 8 4	FF
12 5	E	8 7	"	11 0 3	t	12 9 8 5	CR
12 6	F	12 11 1	j	11 0 4	u	12 9 8 6	SO
12 7	G	12 11 2	k	11 0 5	v	12 9 8 7	SI
12 8	H	12 11 3	l	11 0 6	w	11 9 8 1	EM
12 9	I	12 11 4	m	11 0 7	x	11 9 8 4	FS
11 0	}	12 11 5	n	11 0 8	y	11 9 8 5	GS
11 1	J	12 11 6	o	11 0 9	z	11 9 8 6	RS
11 2	K	12 11 7	p	11 9 1	DC1	11 9 8 7	US
11 3	L	12 11 8	q	11 9 2	DC2	0 9 8 5	ENQ
11 4	M	12 11 9	r	11 9 3	DC3	0 9 8 6	ACK
11 5	N	12 0 1	a	11 9 6	BS	0 9 8 7	BEL
11 6	O	12 0 2	b	11 9 8	CAN	12 11 9 8 1	DLE
11 7	P	12 0 3	c	11 8 2	]	12 0 9 8 1	NUL

7 9

12 0 2 4 6 8

12 11 0 1 6 7 8 9

*Binary**Mode Switch**End of File*

EARLY DEC CARD CODES

Character	7-Bit Octal	DEC 029	DEC 026	Character	7-Bit Octal	DEC 029	DEC 026
Space	040	None	None	@	100	8 4	8 4
!	041	11 8 2*	12 8 7	A	101	12 1	12 1
"	042	8 7	0 8 5	B	102	12 2	12 2
#	043	8 3	0 8 6	C	103	12 3	12 3
\$	044	11 8 3	11 8 3	D	104	12 4	12 4
%	045	0 8 4	0 8 7	E	105	12 5	12 5
&	046	12	11 8 7	F	106	12 6	12 6
'	047	8 5	8 6	G	107	12 7	12 7
(	050	12 8 5	0 8 4	H	110	12 8	12 8
)	051	11 8 5	12 8 4	I	111	12 9	12 9
*	052	11 8 4	11 8 4	J	112	11 1	11 1
+	053	12 8 6	12	K	113	11 2	11 2
,	054	0 8 3	0 8 3	L	114	11 3	11 3
-	055	11	11	M	115	11 4	11 4
.	056	12 8 3	12 8 3	N	116	11 5	11 5
/	057	0 1	0 1	O	117	11 6	11 6
0	060	0	0	P	120	11 7	11 7
1	061	1	1	Q	121	11 8	11 8
2	062	2	2	R	122	11 9	11 9
3	063	3	3	S	123	0 2	0 2
4	064	4	4	T	124	0 3	0 3
5	065	5	5	U	125	0 4	0 4
6	066	6	6	V	126	0 5	0 5
7	067	7	7	W	127	0 6	0 6
8	070	8	8	X	130	0 7	0 7
9	071	9	9	Y	131	0 8	0 8
:	072	8 2	11 8 2 or 11 0†	Z	132	0 9	0 9
;	073	11 8 6	0 8 2	[	133	12 8 2	11 8 5
<	074	12 8 4	12 8 6	\	134	11 8 7*	8 7
=	075	8 6	8 3	]	135	0 8 2*	12 8 5
>	076	0 8 6	11 8 6	^	136	12 8 7*	8 5
?	077	0 8 7	12 8 2 or 12 0†	_	137	0 8 5	8 2

Binary 7 9  
 Mode Switch 12 0 2 4 6 8  
 End of File 12 11 0 1 6 7 8 9

†The Monitor accepts either punch for input but outputs only the triple punch.

These two DEC card codes provide a representation for the figure and upper case character subset. DEC 029 is not available in all programs, but it is almost identical to the ASCII subset, differing only in the four column punches indicated by asterisks as follows:

DEC 029	11 8 2	11 8 7	0 8 2	12 8 7
ASCII	12 8 7	0 8 2	11 8 2	11 8 7

The next two pages show the relationship among the various character sets for the column punches listed

Column Punch	Character	Column Punch	Character
<i>None</i>	<i>Space</i>	12 9	I
0	0	11 1	J
1	1	11 2	K
2	2	11 3	L
3	3	11 4	M
4	4	11 5	N
5	5	11 6	O
6	6	11 7	P
7	7	11 8	Q
8	8	11 9	R
9	9	0 1	/
12 1	A	0 2	S
12 2	B	0 3	T
12 3	C	0 4	U
12 4	D	0 5	V
12 5	E	0 6	W
12 6	F	0 7	X
12 7	G	0 8	Y
12 8	H	0 9	Z

Column Punch	026 Data Processing	026 Fortran	029	DEC 026	DEC 029	ASCII
12	&	+	&	+	&	&
11	-	-	-	-	-	-
12 0				?		
11 0				:		
8 2			:	-	:	:
8 3	#	=	#	=	#	#
8 4	@	-	@	@	@	@
8 5			'	^	'	'
8 6			=	'	=	=
8 7			"	\	"	"
12 8 2			φ	?	[	[
12 8 3			.	.	.	.

above, and where they exist, the corresponding single-key punch configurations and printed characters for several IBM key punches.

Column Punch	026 Data Processing	026 Fortran	029	DEC 026	DEC 029	ASCII
12 8 4	□	)	<	)	<	<
12 8 5			(	]	(	(
12 8 6			+	<	+	+
12 8 7				!	^	!
11 8 2			!	:	!	]
11 8 3	\$	\$	\$	\$	\$	\$
11 8 4	*	*	*	*	*	*
11 8 5			)	[	)	)
11 8 6			;	>	;	;
11 8 7			┘	&	\	^
0 8 2			<i>See note</i>	;	]	\
0 8 3	,	,	,	,	,	,
0 8 4	%	(	%	(	%	%
0 8 5			←	"	—	—
0 8 6			>	#	>	>
0 8 7			?	%	?	?
7 9				<i>Binary</i>	<i>Binary</i>	EOT
12 0 2 4 6 8				<i>Mode Switch</i>	<i>Mode Switch</i>	
12 11 0 1 6 7 8 9				<i>End of File</i>	<i>End of File</i>	

NOTE: There is a single key for the 0 8 2 punch on the 029 but printing is suppressed.

## APPENDIX C

### IO BIT ASSIGNMENTS

The drawings on the following pages define the meanings of the bits in the half words and full words of control and status information handled by the IO instructions (bits that cause interrupts are indicated by asterisks). First are the KI10 processor, the KA10 processor, and the console IO devices [pages C-2, C-6 and C-8]. The rest of the appendix is devoted to the peripheral devices, which are in order by type number, and the pagination also uses those numbers. Besides the bit layouts for conditions and status, the section on a given device also lists all other IO instructions for that device, showing the operations performed in symbolic form using the conventions defined for the representation of the processor instructions in Appendix A [see page A-13].

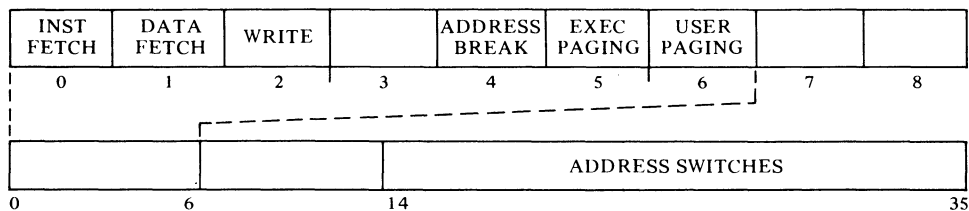
**KI10 PROCESSOR**

Console APR 000 PI 004 PTR 104

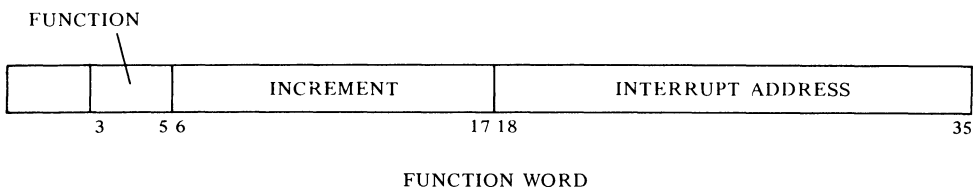
DATAI APR, 70004 (DS) → (E) (RSW)

DATAO PI, 70054 If MI PROG DIS = 0: (E) → (MI)  
1 → PROGRAM DATA

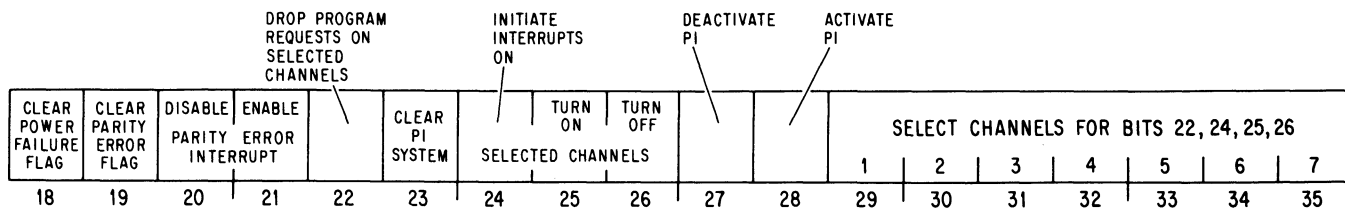
DATAO PTR, 71054



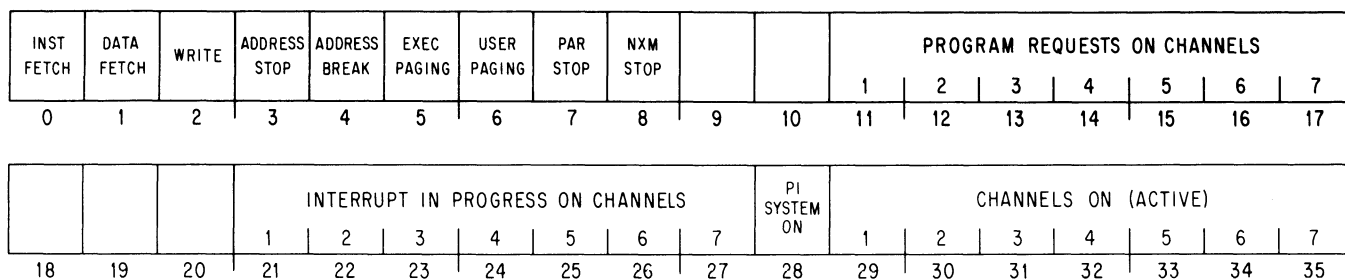
Priority Interrupt PI 004



CONO PI, 70060



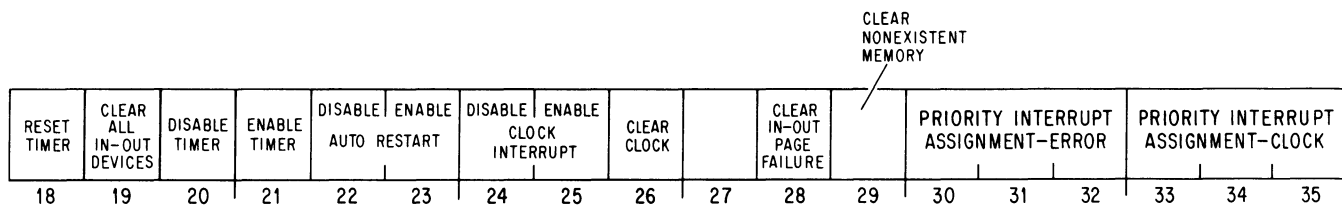
CONI PI, 70064



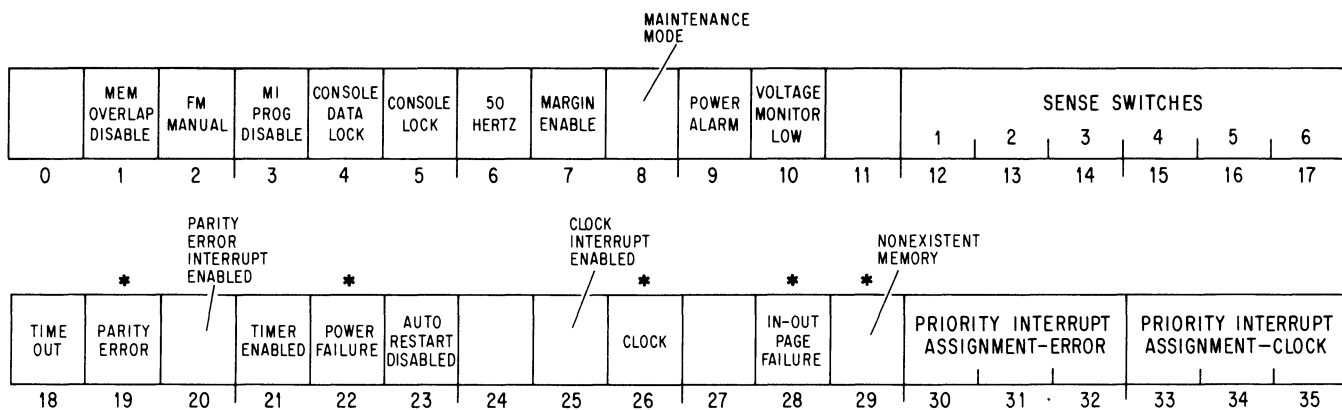


Processor Conditions APR 000

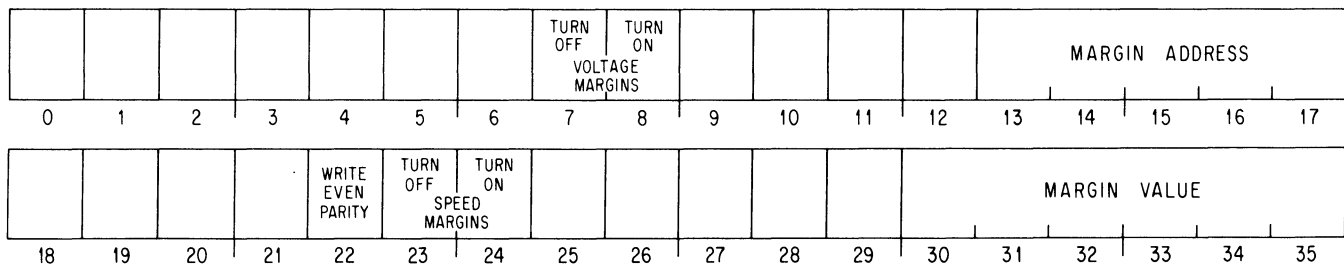
CONO APR, 70020



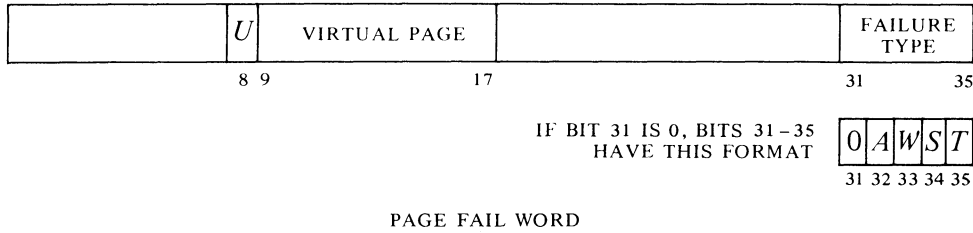
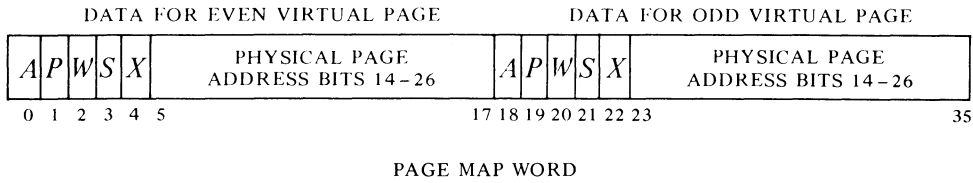
CONI APR, 70024



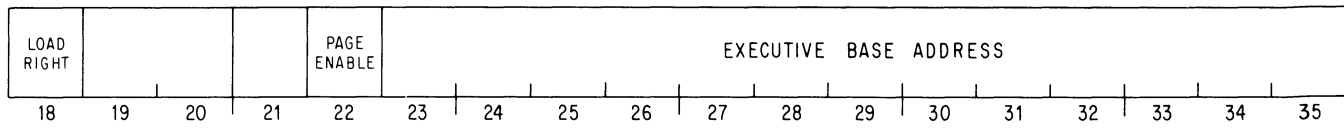
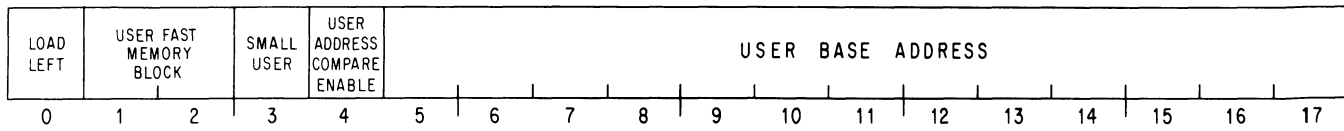
DATAO APR, 70014



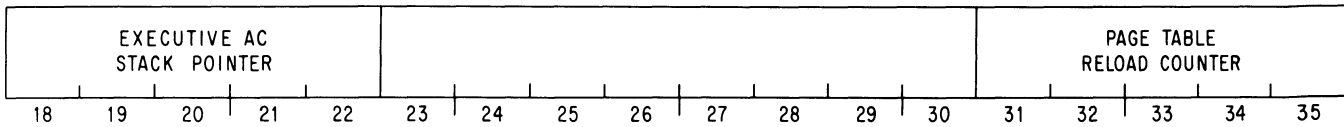
Memory Management PAG 010



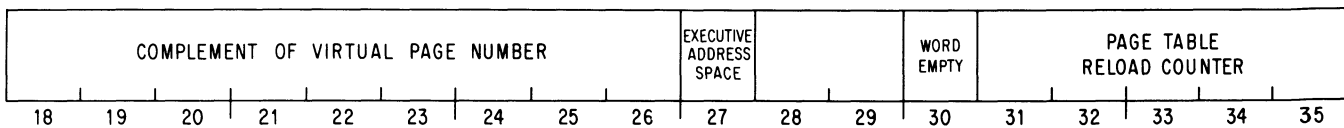
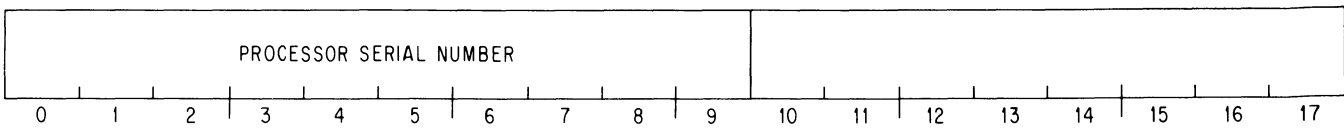
DATAO PAG, 70114 0 → ASSOCIATIVE MEMORY  
 DATAI PAG, 70104



CONO PAG, 70120

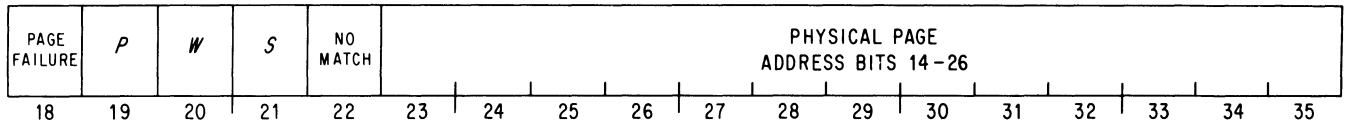


CONI PAG, 70124



MAP

257



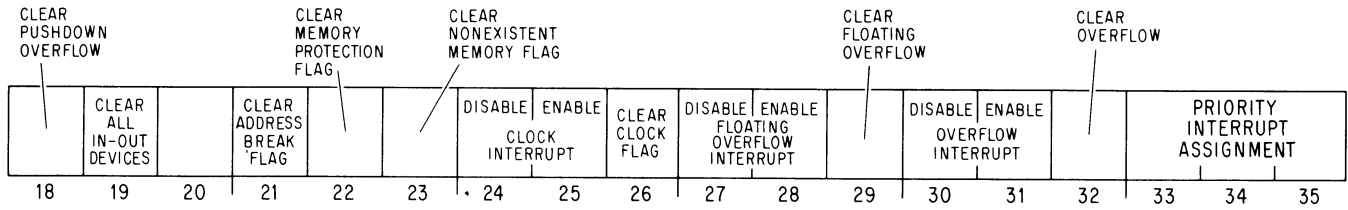
KA10 PROCESSOR

Console APR 000

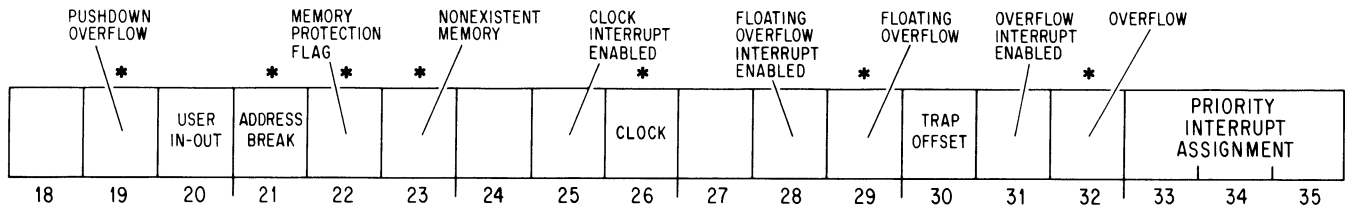
DATAI APR, 70004 (DS) → (E) (RSW)

Processor Conditions APR 000

CONO APR, 70020

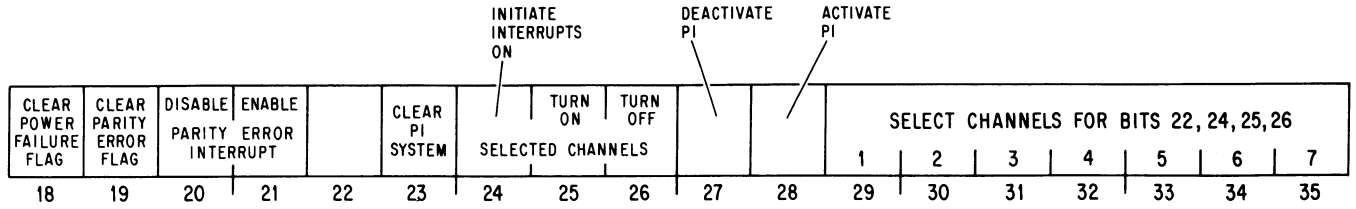


CONI APR, 70024

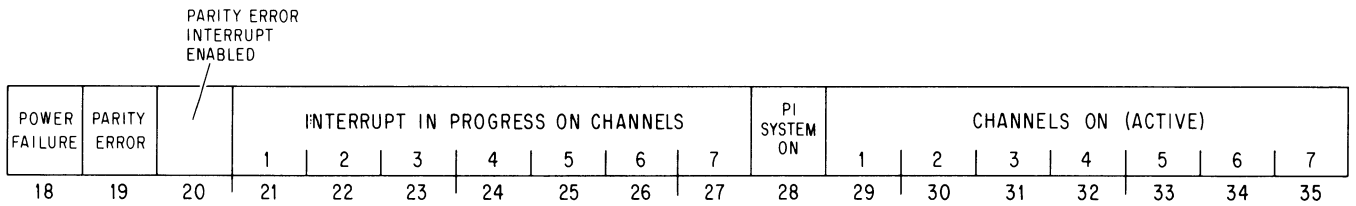


Priority Interrupt PI 004

CONO PI, 70060

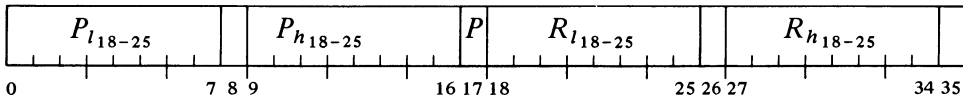


CONI PI, 70064



Memory Management APR 000

DATAO APR, 70014



CONSOLE IO

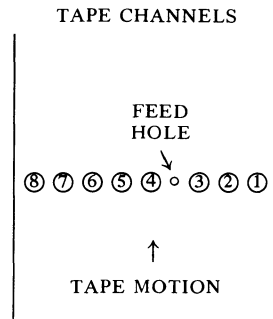
Reader PTR 104

CONO PTR, 71060

			BINARY	BUSY	DONE	PRIORITY INTERRUPT ASSIGNMENT		
27	28	29	30	31	32	33	34	35

CONI PTR, 71064

TAPE			BINARY	BUSY	DONE	PRIORITY INTERRUPT ASSIGNMENT		
27	28	29	30	31	32	33	34	35



DATAI PTR, 71044 (BUFFER) → (E)  
 0 → DONE  
 1 → BUSY

Punch PTP 100

CONO PTP, 71020

			BINARY	BUSY	DONE	PRIORITY INTERRUPT ASSIGNMENT		
27	28	29	30	31	32	33	34	35

CONI PTP, 71024

		NO TAPE	BINARY	BUSY	DONE	PRIORITY INTERRUPT ASSIGNMENT		
27	28	29	30	31	32	33	34	35

DATAO PTP, 71014 (E)<sub>28-35</sub> → (BUFFER)  
 0 → DONE  
 1 → BUSY

Console Terminal TTY 120

CONO TTY, 71220

TEST	CLEAR INPUT BUSY	CLEAR INPUT DONE	CLEAR OUTPUT BUSY	CLEAR OUTPUT DONE	SET INPUT BUSY	SET INPUT DONE	SET OUTPUT BUSY	SET OUTPUT DONE	PRIORITY INTERRUPT ASSIGNMENT		
24	25	26	27	28	29	30	31	32	33	34	35

CONI TTY, 71224

TEST					INPUT BUSY	INPUT DONE	OUTPUT BUSY	OUTPUT DONE	PRIORITY INTERRUPT ASSIGNMENT		
24	25	26	27	28	29	30	31	32	33	34	35

DATAO TTY, 71214 (E)<sub>28-35</sub> → (BUFFER)  
 0 → OUTPUT DONE  
 1 → OUTPUT BUSY

DATAI TTY, 71204 (BUFFER) → (E)<sub>28-35</sub>  
 0 → INPUT DONE

CP10

IO BIT ASSIGNMENTS

CARD PUNCH CP10  
CDP 110

CONO CDP, 71120

		CLEAR PUNCH	OFFSET CARD		EJECT CARD	DISABLE TROUBLE INTERRUPTS	ENABLE TROUBLE INTERRUPTS	CLEAR ERROR	DISABLE END OF CARD	ENABLE END OF CARD	CLEAR END OF CARD	SET PUNCH ON	CLEAR DATA REQUEST	SET DATA REQUEST	PRIORITY INTERRUPT ASSIGNMENT		
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

CONI CDP, 71124

		HOPPER LOW	NEED OPERATOR SERVICE	PICK FAILURE-STACK FAILURE	EJECT FAILURE	TROUBLE *	TROUBLE INTERRUPT ENABLED	ERROR	CARD IN PUNCH	END OF CARD ENABLED	END OF CARD *	PUNCH ON	BUSY	DATA REQUEST *	PRIORITY INTERRUPT ASSIGNMENT		
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

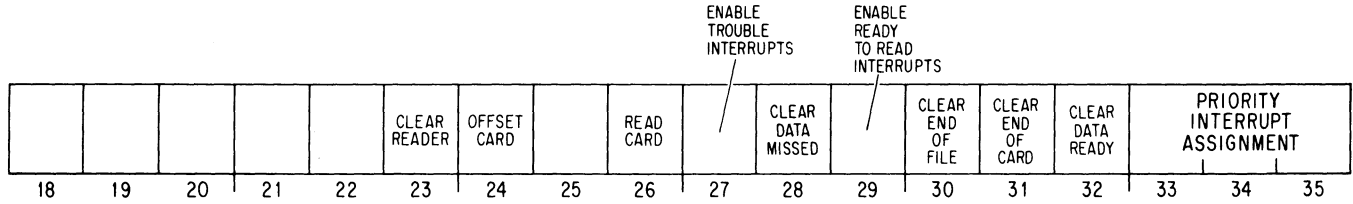
DATAO CDP, 71114    0 → DATA REQUEST  
                          1 → PUNCH ON, BUSY

							ROW 12	ROW 11	ROW 0	ROW 1	ROW 2	ROW 3	ROW 4	ROW 5	ROW 6	ROW 7	ROW 8	ROW 9
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	

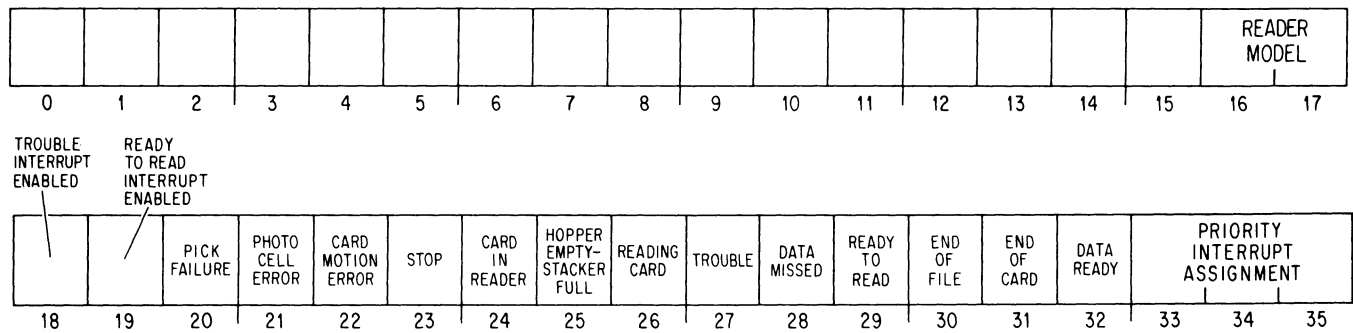


**CARD READER CR10**  
CR 150

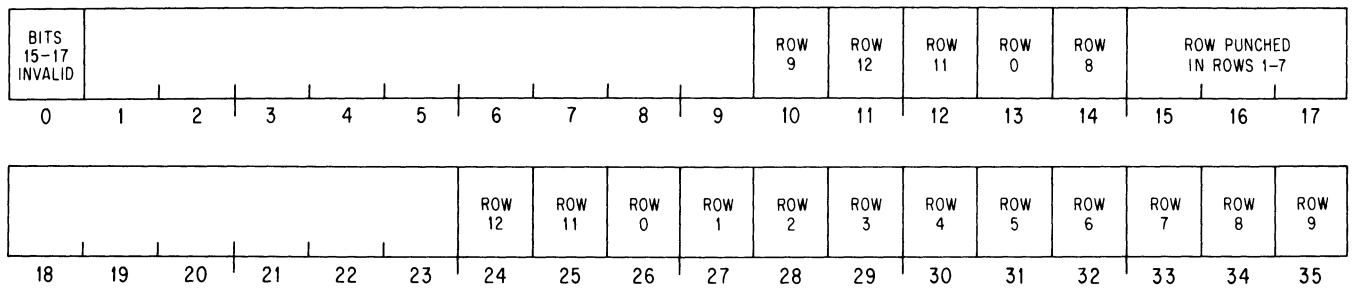
CONO CR, 71520



CONI CR, 71524



DATAI CR, 71504 0 → DATA READY



2

3

4

5

**TWELVE- AND EIGHTEEN-BIT COMPUTER INTERFACE DA10**  
*CCI 014*

CONO CCI,        70160

						DISABLE SELF-CHECK	ENABLE	CLEAR FROM TEN FULL	SET FROM TEN FULL	CLEAR FROM TEN EMPTY	SET FROM TEN EMPTY	CLEAR TO TEN FULL	SET TO TEN FULL	CLEAR TO TEN EMPTY	SET TO TEN EMPTY			PRIORITY INTERRUPT ASSIGNMENT
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	

CONI CCI,        70164

							SELF CHECK ENABLED		FROM TEN FULL		FROM TEN EMPTY		TO TEN FULL		TO TEN EMPTY			PRIORITY INTERRUPT ASSIGNMENT
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	

DATAO CCI,        70154    (E) → (FROM TEN)

DATAI CCI,        70144    (TO TEN) → (E)

*Twelve-bit Computer 35, 36, 37*

- 6361    *If TO TEN EMPTY = 1: skip*
- 6371    *If FROM TEN FULL = 1: skip*
- 6351    0 → (TO TEN)
- 6354    (AC) → (TO TEN)<sub>0-11</sub>
- 6364    (AC) → (TO TEN)<sub>12-23</sub>
- 6374    (AC) → (TO TEN)<sub>24-35</sub>
- 6352    (TO TEN)<sub>0-11</sub> ∨ (AC) → (AC)
- 6362    (TO TEN)<sub>12-23</sub> ∨ (AC) → (AC)
- 6372    (TO TEN)<sub>24-35</sub> ∨ (AC) → (AC)

*Eighteen-bit Computer 22*

- 702221    *If TO TEN EMPTY = 1: skip*
- 702241    *If FROM TEN FULL = 1: skip*
- 702201    0 → (TO TEN)
- 702204    (AC) → (TO TEN)<sub>0-17</sub>
- 702224    (AC) → (TO TEN)<sub>18-35</sub>
- 702212    (TO TEN)<sub>0-17</sub> → (AC)
- 702232    (TO TEN)<sub>18-35</sub> → (AC)

DC10-1

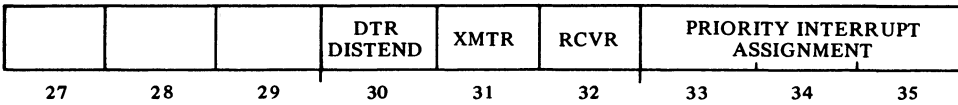
IO BIT ASSIGNMENTS

DATA LINE SCANNER DC10  
DLS 240

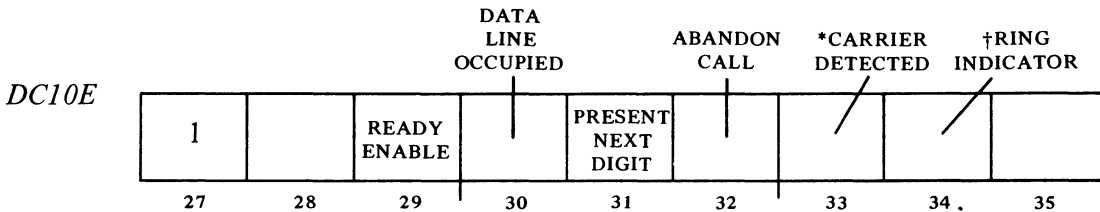
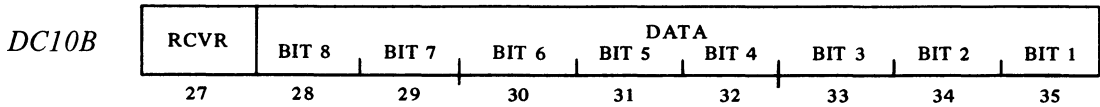
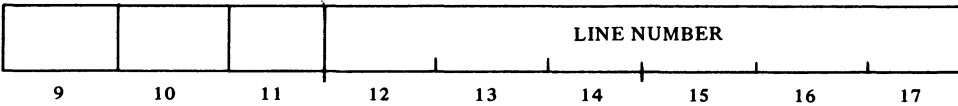
CONO DLS, 72420



CONI DLS, 72424



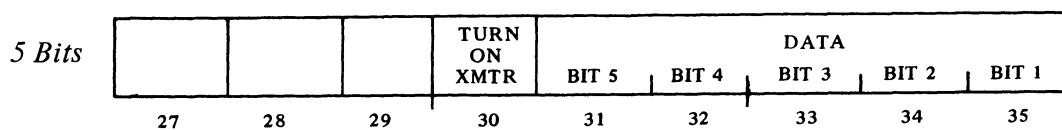
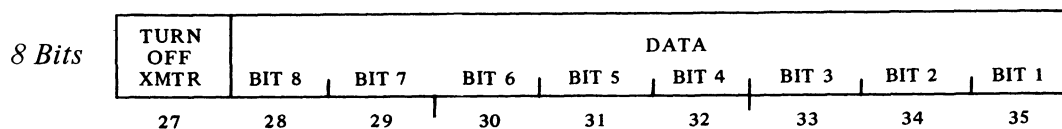
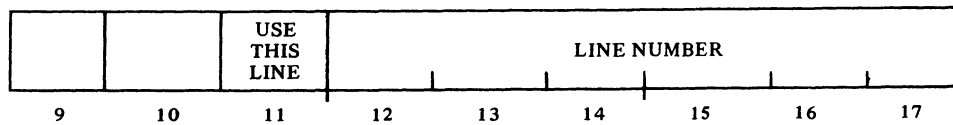
DATAI DLS, 72404



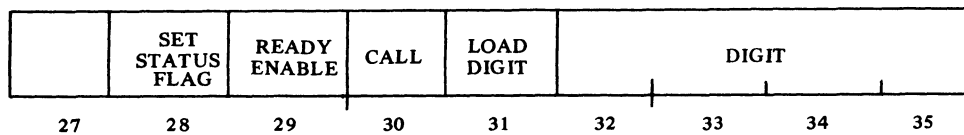
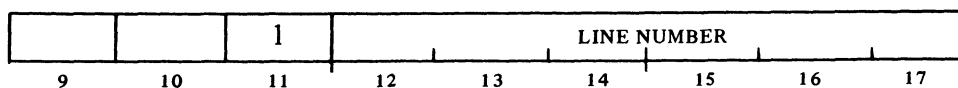
Alternates: \*CLEAR TO SEND    †RESTRAIN DETECTED  
DATA SET READY

DATAO DLS, 72414

DC10B

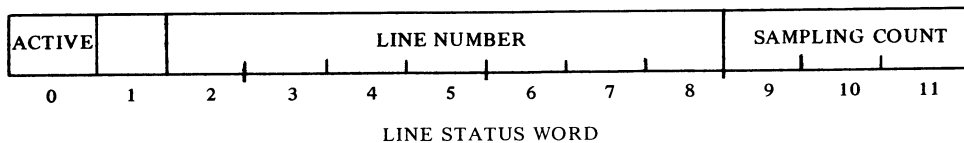


DC10E



## DATA COMMUNICATION SYSTEM DC68A

## DC08A Serial Line Multiplexer 40-47



	<i>Clock 1</i>	<i>Clock 2</i>	<i>Clock 3</i>	<i>Clock 4</i>	
	6424	6434	6444	6454	<i>Enable</i> CLOCK
	6421	6431	6441	6451	<i>If</i> CLOCK FLAG = 1: <i>skip</i>
	6422	6432	6442	6452	<i>Disable</i> CLOCK
6411	0 → (LSR)		6401		(LSR) + 1 → (LSR)
6412	(AC) <sub>5-11</sub> ∨ (LSR) → (LSR)		6414		(LSR) ∨ (AC) <sub>5-11</sub> → (AC) <sub>5-11</sub>
6404	(AC) <sub>11</sub> → LINE (AC) <sub>i</sub> → (AC) <sub>i+1</sub> 0 → (AC) <sub>0</sub>				
6471	0 → (LCC)		6461		(LCC) + 1 → (LCC)
6472	(AC) <sub>5-11</sub> ∨ (LCC) → (LCC)		6464		(LCC) ∨ (AC) <sub>5-11</sub> → (AC) <sub>5-11</sub>
6402	(LSW) <sub>2-8</sub> → (LSR) <i>If</i> [(LSW) <sub>0</sub> = 0] ∨ [LINE HOLD = 1]: ∼LINE → (LSW) <sub>0</sub> <i>If</i> [(LSW) <sub>0</sub> = 1] ∧ [LINE HOLD = 0]: (LSW) <sub>9-11</sub> + 1 → (LSW) <sub>9-11</sub> <i>If</i> (LSW) <sub>9-11</sub> = 2: <i>if</i> LINE HOLD = 0: (CAW) <sub>i</sub> → (CAW) <sub>i+1</sub> LINE → (CAW) <sub>0</sub> <i>If</i> (LSW) <sub>9-11</sub> ≠ 2: <i>skip</i> 2  <i>If</i> (CAW) <sub>11</sub> = 1: <i>if</i> (LCC) = 0: 1 → LINE HOLD <i>skip</i> <i>if</i> (LCC) ≠ 0: (CAW) → (AC) 0 → LINE HOLD <i>If</i> (CAW) <sub>11</sub> = 0: <i>skip</i>				

## DC08F Modem Control 70-73

6704	<i>Enable</i> flags <i>Select</i> modem by (AC <sub>5-11</sub> )
6712	<i>Disable</i>

- 6701 *If* RING FLAG = 1: *skip*
- 6702 (RING SCANNER)  $\vee$  (AC)<sub>5-11</sub>  $\rightarrow$  (AC)<sub>5-11</sub>
- 6734 0  $\rightarrow$  RING FLAG    *Enable* RING SCANNER
- 6711 *If* CARRIER FLAG = 1: *skip*
- 6714 (CARRIER SCANNER)  $\vee$  (AC)<sub>5-11</sub>  $\rightarrow$  (AC)<sub>5-11</sub>  
CARRIER DETECTED  $\vee$  (AC)<sub>0</sub>  $\rightarrow$  (AC)<sub>0</sub>
- 6724 0  $\rightarrow$  CARRIER FLAG    *Enable* CARRIER SCANNER  
0  $\rightarrow$  CARRIER CHANGE *for line selected by* (AC)<sub>5-11</sub>
- 6731 1  $\rightarrow$  DATA TERMINAL READY *for line selected by* (AC)<sub>5-11</sub>
- 6732 1  $\rightarrow$  REQUEST TO SEND *for line selected by* (AC)<sub>5-11</sub>
- 6721 0  $\rightarrow$  DATA TERMINAL READY *for line selected by* (AC)<sub>5-11</sub>
- 6722 0  $\rightarrow$  REQUEST TO SEND *for line selected by* (AC)<sub>5-11</sub>

*DC08H Call Control 74,75*

- 6754 *Select ACU by* (AC)<sub>8-11</sub>)
- 6756 *If* [POWER ON = 1]  $\wedge$  [DATA LINE OCCUPIED = 0]: 1  $\rightarrow$  CALL REQUEST  
*If* [POWER ON = 0]  $\vee$  [DATA LINE OCCUPIED = 1]: 1  $\rightarrow$  CALL STATUS
- 6751 *If* CALL STATUS = 1: *skip*
- 6755 0  $\rightarrow$  CALL STATUS    STATUS  $\vee$  (AC)<sub>0-4</sub>  $\rightarrow$  (AC)<sub>0-4</sub>

POWER ON	DATA LINE OCCUPIED	ABANDON CALL	DATA SET STATUS	PRESENT NEXT DIGIT	
0	1	2	3	4	5

- 6753 *If* DIGIT REQUEST = 1: *skip*
- 6741 0  $\rightarrow$  DIGIT REQUEST
- 6757 (AC)<sub>8-11</sub>  $\rightarrow$  ACU  
*If* CALL REQUEST = 1: 1  $\rightarrow$  DIGIT PRESENT
- 6752 0  $\rightarrow$  CALL REQUEST    0  $\rightarrow$  DIGIT PRESENT





DATA CHANNEL DF10C/DF10  
Control Words

*DF10C 22-bit Address*

<i>Normal transfer</i>	— WORD COUNT	INITIAL DATA ADDRESS — 1
	0 13 14	35
<i>Discard input</i>	— WORD COUNT	ZERO
	0 13 14	35
<i>Jump</i>	ZERO	CONTROL WORD ADDRESS
	0 13 14	35
<i>Halt</i>	ZERO	ZERO
	0 13 14	35

*DF10 18-bit Address*

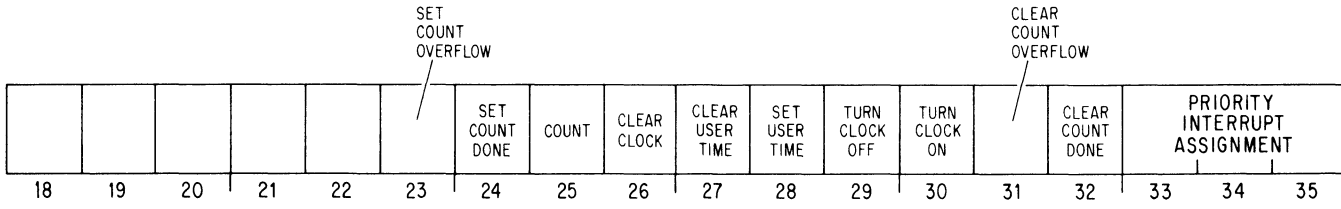
<i>Normal transfer</i>	— WORD COUNT	INITIAL DATA ADDRESS — 1
	0 17 18	35
<i>Discard input</i>	— WORD COUNT	ZERO
	0 17 18	35
<i>Jump</i>	ZERO	CONTROL WORD ADDRESS
	0 17 18	35
<i>Halt</i>	ZERO	ZERO
	0 17 18	35

DK10

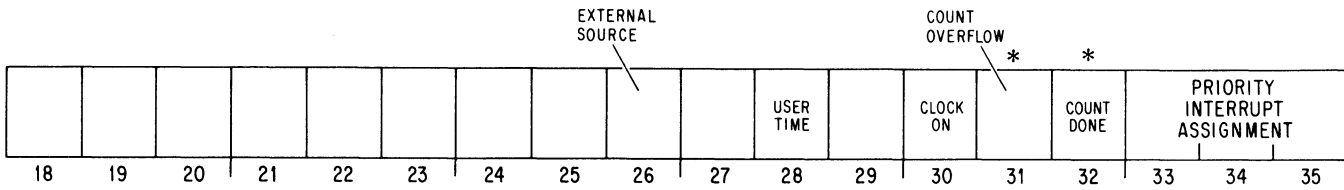
IO BIT ASSIGNMENTS

REAL TIME CLOCK DK10  
CLK 070

CONO CLK, 70720



CONI CLK, 70724



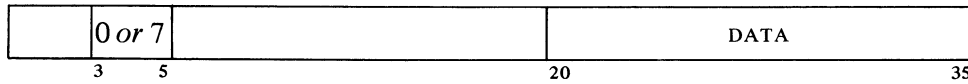
DATAO CLK, 70714 (E)<sub>R</sub> → (INTERVAL)

DATAI CLK, 70704 (COUNTER) → (E)<sub>R</sub>

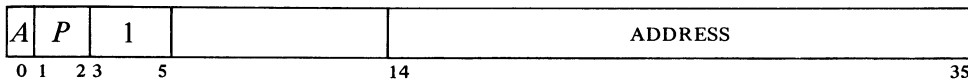
**PDP-11 DATA LINK DL10**  
DLB 060 DLC 064

*Data Formats*

*Immediate mode*

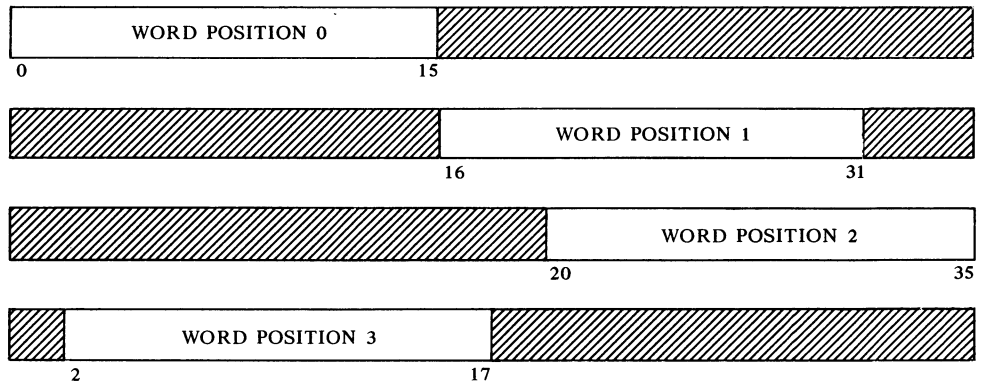


*Indirect mode*

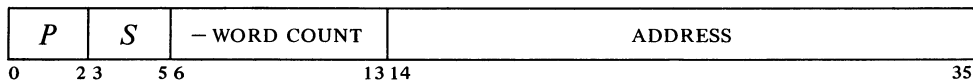


- A* Access  
 0 Read or write  
 1 Read only

*Word Position Specified by P*

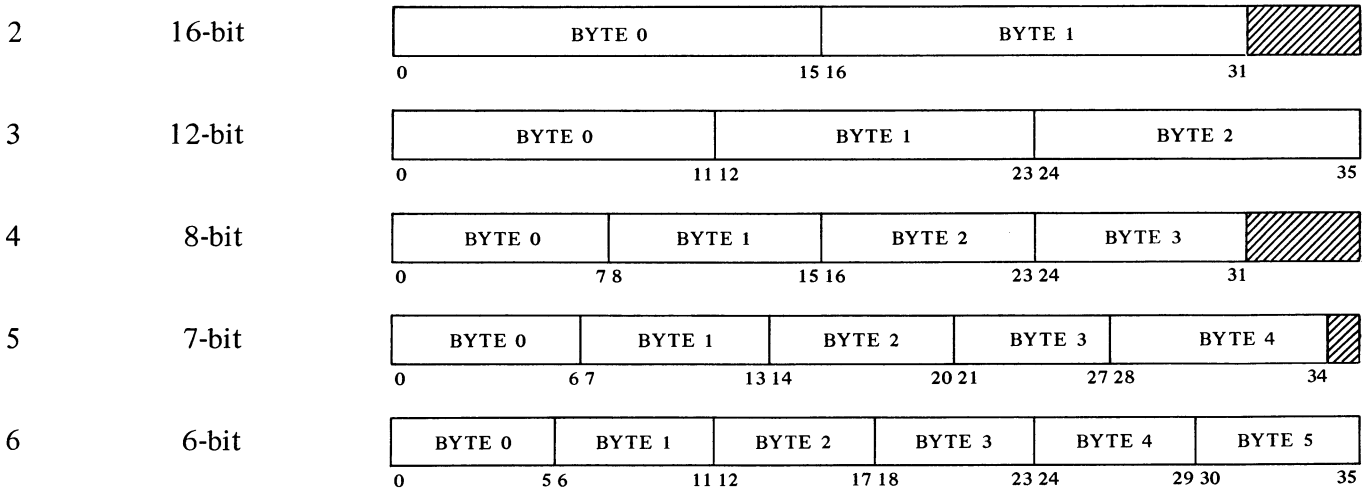


*Pointer modes*



*S* *Byte Mode*

*Byte Positions Specified by P*



*PDP-11 Control and Status Registers*

DATO 100000 (Conditions out)

SET 11 INTERRUPT	CLEAR	SET 10 INTERRUPT	CLEAR	SET NONEX MEMORY	CLEAR	SET PARITY ERROR	CLEAR	SET WORD COUNT OVERFLOW	CLEAR			ERROR INTERRUPT ENABLE	11 INTERRUPT ENABLE	INTERRUPT ASSIGNMENT
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1 0

DATI 100000 (Status)

*		*†		*		*		*		*									
11 INTERRUPT		10 INTERRUPT		NONEX MEMORY		PARITY ERROR		WORD COUNT OVERFLOW		THIS PORT ENABLED		ERROR INTERRUPT ENABLE	11 INTERRUPT ENABLE	INTERRUPT ASSIGNMENT					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1 0					

† Interrupts PDP-10.

*PDP-10 Standard Instructions*

CONO DLB, 70620

BASE ADDRESS (12 HIGH BITS)												MASK FOR SIZE OF POINTER BLOCK				PDP-11	
8K OPTION (9 BITS)																	
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

CONO DLC, 70660

CLEAR DL10	LOCK DL10	ACTION OF IS: 0 CLEAR 1 SET	PDP-11-3			PDP-11-2			PDP-11-1			PDP-11-0			PRIORITY INTERRUPT ASSIGNMENT		
			11 INT'RUPT	PORT ENABLE	10 INT'RUPT	11 INT'RUPT	PORT ENABLE	10 INT'RUPT	11 INT'RUPT	PORT ENABLE	10 INT'RUPT	11 INT'RUPT	PORT ENABLE	10 INT'RUPT	33	34 35	
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

CONI DLC, 70664

DIAG	DIAG MSYN	DIAG C0	DIAG C1	DIAG R2	DIAG R1	DIAG INH CYC		11-3 8K OPTION	11-2 8K OPTION	11-1 8K OPTION	11-0 8K OPTION	IOC LOCK ON	~IOC LOCK DELAY ON	~IOC LOCK 1 OUT	~IOC LOCK 0 OUT	18-BIT ADDRESS	STANDARD INTERRUPT
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
			*†	*	*	*†	*	*†	*	*†	*	*†	*				
	GOT DL10 LOCKED		PDP-11-3			PDP-11-2			PDP-11-1			PDP-11-0			PRIORITY INTERRUPT ASSIGNMENT		
			11 INT'RUPT	PORT ENABLE	10 INT'RUPT	11 INT'RUPT	PORT ENABLE	10 INT'RUPT	11 INT'RUPT	PORT ENABLE	10 INT'RUPT	11 INT'RUPT	PORT ENABLE	10 INT'RUPT	33	34 35	
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

† Interrupts PDP-11.

DATAO DLC, 70654

CLEAR SELECTED LEFT BITS	SET		PDP-11-3			PDP-11-2			PDP-11-1			PDP-11-0					
0	1	2	NONEX MEMORY	PARITY ERROR	WORD COUNT OVERFLO	NONEX MEMORY	PARITY ERROR	WORD COUNT OVERFLO	NONEX MEMORY	PARITY ERROR	WORD COUNT OVERFLO	NONEX MEMORY	PARITY ERROR	WORD COUNT OVERFLO	15	16	17
			PDP-11-3			PDP-11-2			PDP-11-1			PDP-11-0					
18	19	20	STOP	START		STOP	START		STOP	START		STOP	START		33	34	35

STANDARD INTERRUPT

DATAI DLC, 70644 (All interrupts to PDP-11)

			PDP-11-3			PDP-11-2			PDP-11-1			PDP-11-0					11-3 NO REMOTE CONSOLE
0	1	2	NONEX MEMORY	PARITY ERROR	WORD COUNT OVERFLO	NONEX MEMORY	PARITY ERROR	WORD COUNT OVERFLO	NONEX MEMORY	PARITY ERROR	WORD COUNT OVERFLO	NONEX MEMORY	PARITY ERROR	WORD COUNT OVERFLO	15	16	17
			PDP-11-3			PDP-11-2			PDP-11-1			PDP-11-0					
11-2 NO REMOTE CONSOLE	11-1 NO REMOTE CONSOLE	11-0 NO REMOTE CONSOLE	PORT EXISTS	RUN DELAY	REMOTE POWER	PORT EXISTS	RUN DELAY	REMOTE POWER	PORT EXISTS	RUN DELAY	REMOTE POWER	PORT EXISTS	RUN DELAY	REMOTE POWER	33	34	35

PDP-10 Diagnostic Instructions

DATAO DLB, 70614

DIAG	DIAG MSYN	DIAG CØ	DIAG C1	SELECT INPUT REGISTER		DIAG INH CYC												PULSE CLOCK		
0	1	2	3	4	5	6	7	8										33	34	35

DATAI DLB, 70604 R = 01: (MB) → (E)  
R = 23: As shown

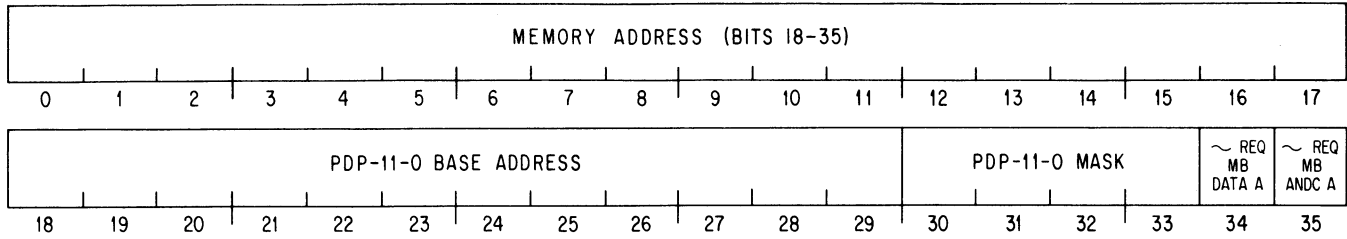
CLOCK READY	~CLOCK SYNC	UNIBUS DATA - UBX																		
0	1	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0			
SHIFT 1632 A	~SHIFT 32 A	~SHIFT 16 A	~SHIFT 4812 A	~SHIFT 12 A	~SHIFT 8 A	~SHIFT 4 A	SHIFT 2 A	~SHIFT 2 A	~SHIFT UBXM 1 A	~SHIFT UBXM Ø A	~SHIFT MB1 A	~SHIFT MB Ø A	REQ HIGH BYTE	REQ LOW BYTE	REQ DATOX A	~REQ UBX DATA A	~REQ P Ø A			
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35			

DL10-4

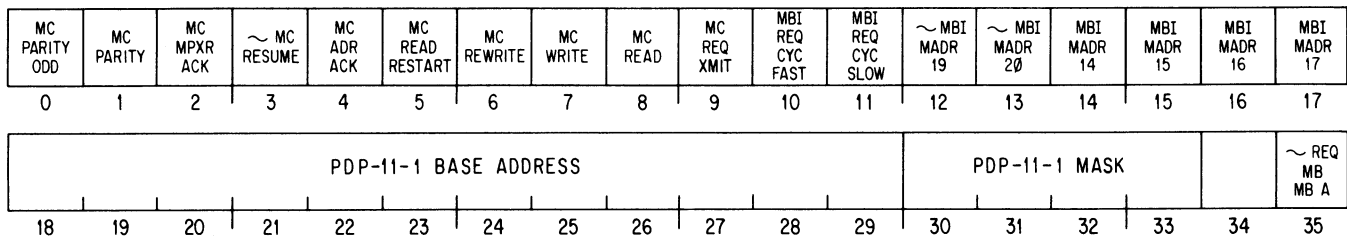
IO BIT ASSIGNMENTS

CONI DLB, 70624

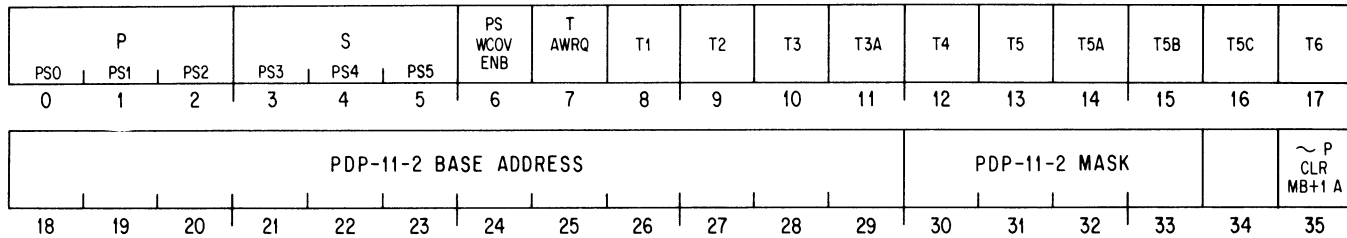
R = 0



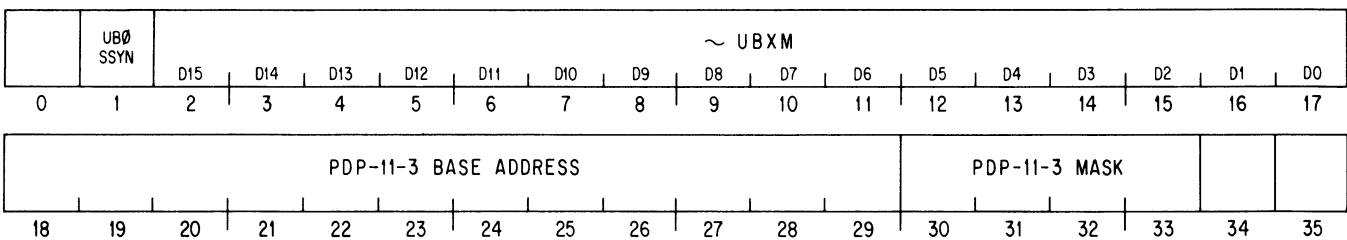
R = 1



R = 2

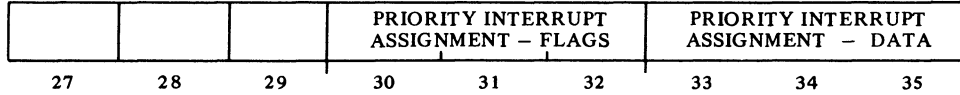


R = 3

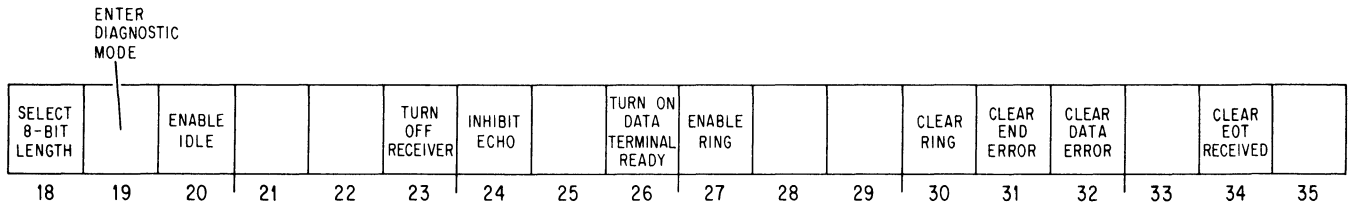


**SINGLE SYNCHRONOUS LINE UNIT DS10**  
*DSS 460 DSI 464*

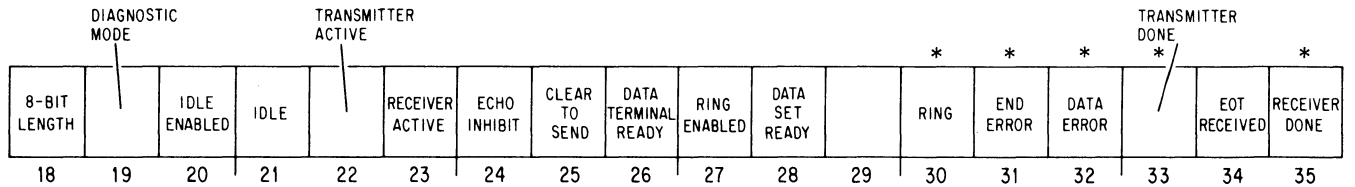
CONO DSI, 74660  
 CONI DSI, 74664



CONO DSS, 74620



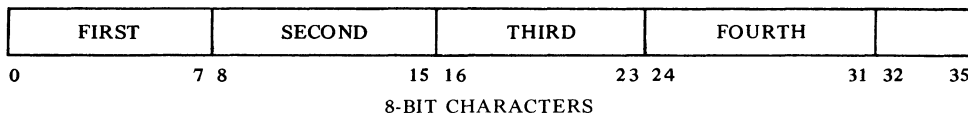
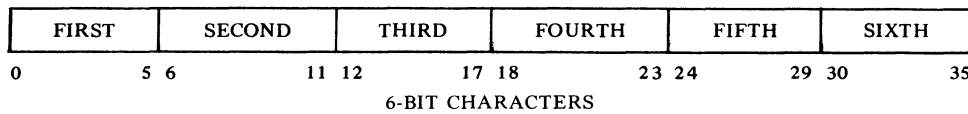
CONI DSS, 74624



DATAO DSS, 74614 (E) → (TRANSMITTER BUFFER)  
 0 → TRANSMITTER DONE

DATAI DSS, 74604 (RECEIVER BUFFER) → (E)  
 0 → (RECEIVER BUFFER)  
 0 → RECEIVER DONE

*Data formats*



MULTIPLE LINE SYNCHRONOUS INTERFACE DS11

Register addresses

775xx0 Receiver status  
 775xx2 Receiver data  
 775xx4 Transmitter status  
 775xx6 Transmitter data  
 Line 0 775400-775406  
 ⋮ ⋮  
 Line 15 775570-775576

DATO 775xx0 (Receiver conditions out)

CLEAR RING	CLEAR BIT OVERRUN	CLEAR CHARACTER OVERRUN	CLEAR CARRIER LOST	SYNC STATE			DATA TERMINAL READY			CHARACTER LENGTH	DONE INTERRUPT ASSIGNMENT		RING ENABLE	RECEIVER ENABLE	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

DATI 775xx0 (Receiver status)

*	*	*	*	*†											
RING	BIT OVERRUN	CHARACTER OVERRUN	CARRIER LOST	SYNC STATE		CARRIER DETECTED	DATA TERMINAL READY	DONE	DATA SET READY	CHARACTER LENGTH	DONE INTERRUPT ASSIGNMENT		RING ENABLE	RECEIVER ENABLE	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

†Sync state interrupts at 01.

DATO 775xx4 (Transmitter conditions out)

	CLEAR BIT OVERRUN	CLEAR CHARACTER OVERRUN	CLEAR CLEAR TO SEND LOST				DATA TERMINAL READY	SET DONE		CHARACTER LENGTH	DONE INTERRUPT ASSIGNMENT		IDLE	REQUEST TO SEND	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

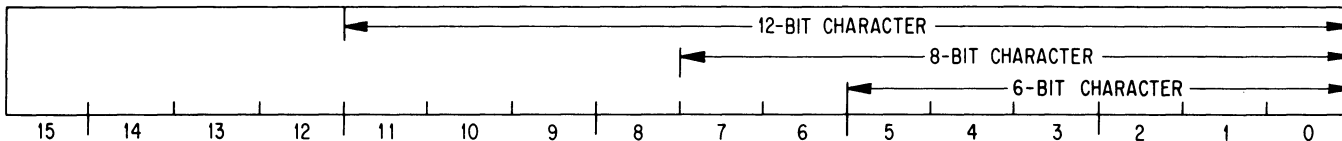
DATI 775xx4 (Transmitter status)

	*	*	*				*								
	BIT OVERRUN	CHARACTER OVERRUN	CLEAR TO SEND LOST			CLEAR TO SEND	DATA TERMINAL READY	DONE	DATA SET READY	CHARACTER LENGTH	DONE INTERRUPT ASSIGNMENT		IDLE	REQUEST TO SEND	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

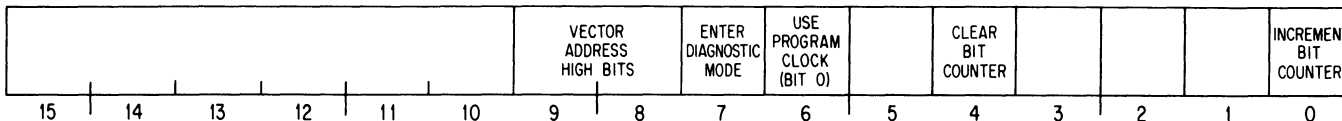


DATI 775xx2 *Read* RECEIVER DATA  
 0 → RECEIVER DONE

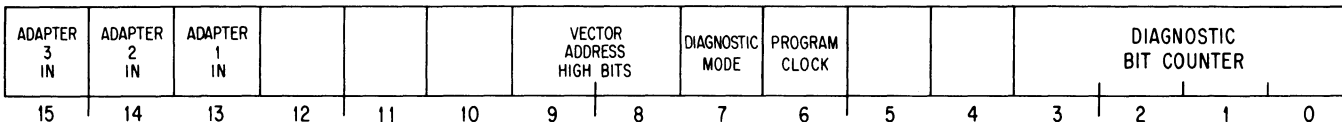
DATO 775xx2 *Load* TRANSMITTER DATA  
 0 → TRANSMITTER DONE



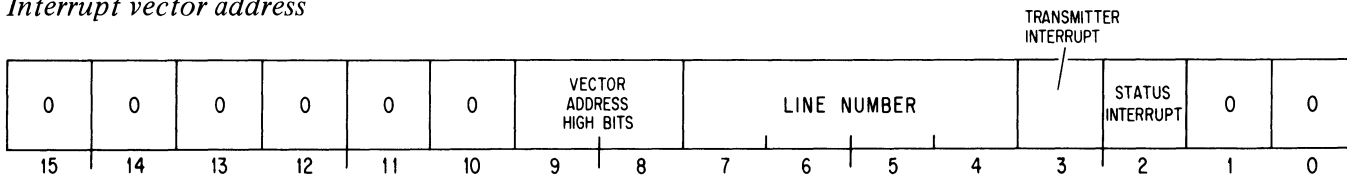
DATO 775600 (*Interface conditions out*)



DATI 775600 (*Interface status*)



*Interrupt vector address*



LP10

IO BIT ASSIGNMENTS

LINE PRINTER LP10  
LPT 124

CONO LPT, 71260

	CLEAR PRINTER			BUSY	DONE	PRIORITY INTERRUPT ASSIGNMENT - ERROR			PRIORITY INTERRUPT ASSIGNMENT - DONE		
24	25	26	27	28	29	30	31	32	33	34	35

CONI LPT, 71264

128	96		ERROR	BUSY	DONE	PRIORITY INTERRUPT ASSIGNMENT - ERROR			PRIORITY INTERRUPT ASSIGNMENT - DONE		
24	25	26	27	28	29	30	31	32	33	34	35

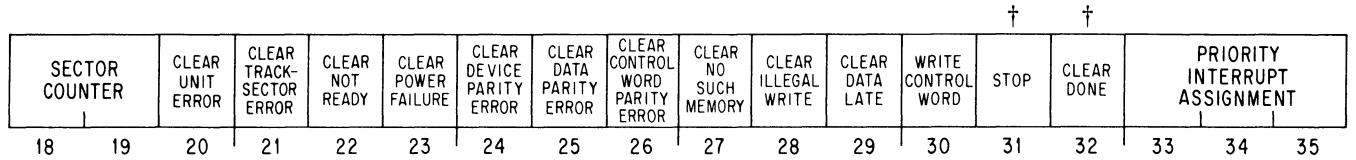
DATAO LPT, 71254 (E) → (BUFFER)  
 0 → DONE  
 1 → BUSY

FIRST	SECOND	THIRD	FOURTH	FIFTH	
0	6 7	13 14	20 21	27 28	34

DATA FORMAT

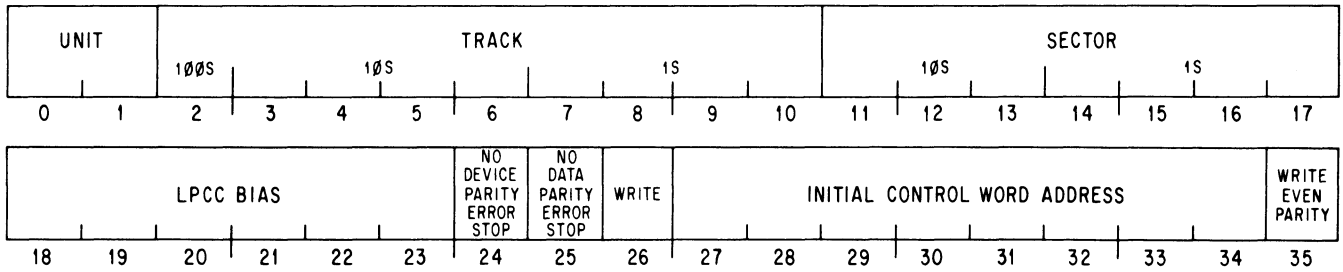
DISK/DRUM SYSTEM RC10  
DSK 170

CONO DSK, 71720

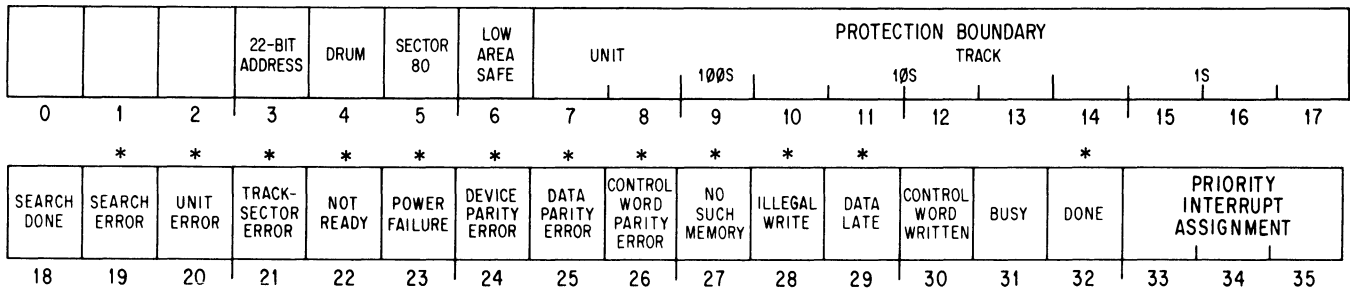


†Bits 31 and 32 both 1: Clear disk system

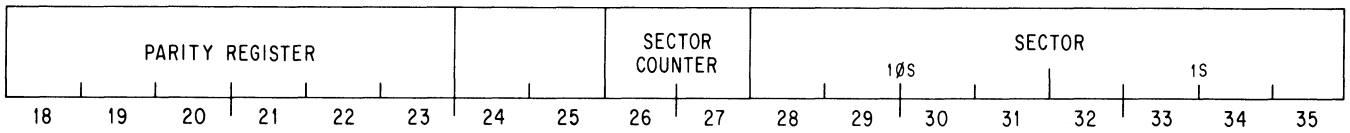
DATAO DSK, 71714 If BUSY = 0: 0 → Flags



CONI DSK, 71724

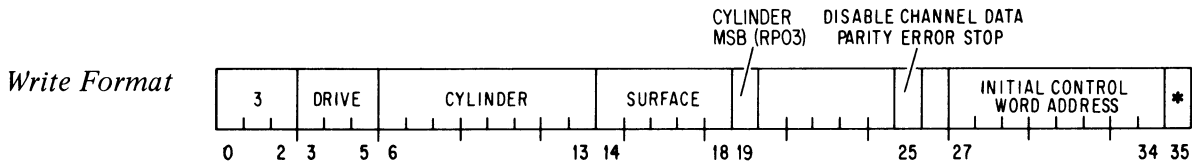
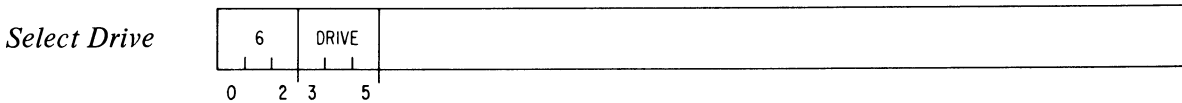
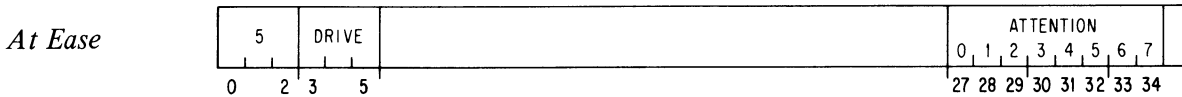
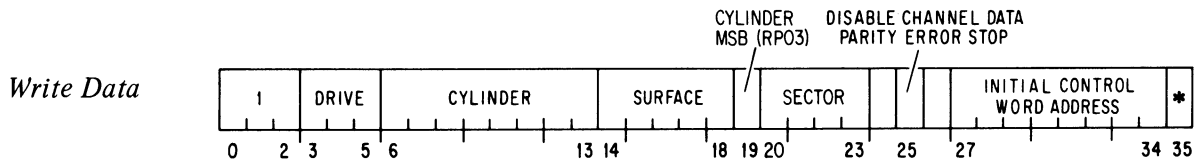
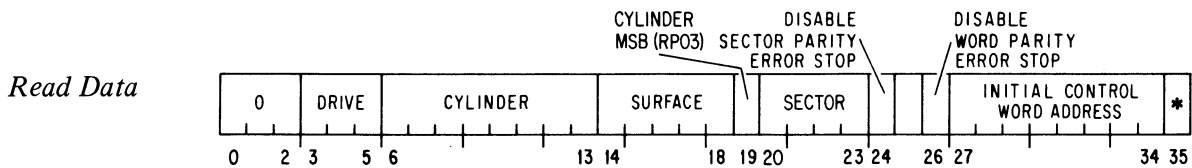
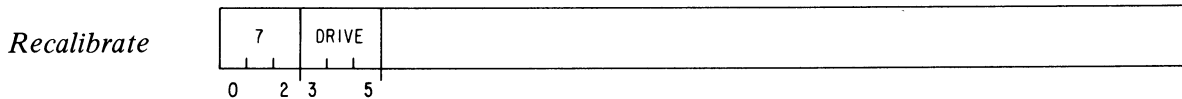
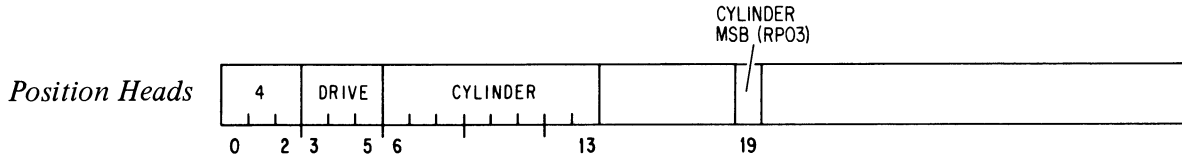
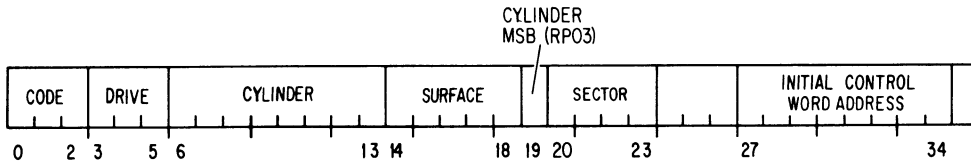


DATAI DSK, 71704



**DISK PACK SYSTEM RP10**  
*DPC 250*

DATAO DPC, 72514 *If* BUSY = 0: 0 → *Flags*



\*Write even parity in memory.

CONO DPC, 72520

		CLEAR POWER FAILURE	CLEAR SEARCH ERROR	CLEAR DATA LATE	CLEAR NO SUCH MEMORY	CLEAR PARITY ERROR FLAGS	CLEAR DISK SYSTEM	CLEAR ILLEGAL WRITE	CLEAR ILLEGAL DATAO	CLEAR SECTOR ADDRESS ERROR	CLEAR SURFACE ADDRESS ERROR	WRITE CONTROL WORD	STOP	CLEAR DONE	PRIORITY INTERRUPT ASSIGNMENT		
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

DATAI DPC, 72504

DRIVE			CYLINDER								POSITION FAILURE	HEADS IN POSITION	DISK ON LINE	FILE UNSAFE	NO SUCH DRIVE	READ ONLY	WRITE HEADER LOCKOUT
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
								*	*	*	*	*	*	*	*	*	*
SECTOR				CYLINDER MSB (RPO3)	RPO3	BAD SPOT	ATTENTION										
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

CONI DPC, 72524

												22-BIT ADDRESS	CONTROL WORD PARITY ERROR	SECTOR PARITY ERROR	CHANNEL DATA PARITY ERROR	DISK WORD PARITY ERROR	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
SEARCH DONE	END OF CYLINDER	POWER FAILURE	SEARCH ERROR	DATA LATE	NO SUCH MEMORY	PARITY ERROR	NOT READY	ILLEGAL WRITE	ILLEGAL DATAO	SECTOR ADDRESS ERROR	SURFACE ADDRESS ERROR	CONTROL WORD WRITTEN	BUSY	INTERRUPT	PRIORITY INTERRUPT ASSIGNMENT		
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

**DECTAPE TD10**  
DTC 320 DTS 324

CONO DTC, 73220 0 → *Flags*

STOP	GO FORWARD	GO REVERSE	INHIBIT START DELAY	SELECT	DESELECT	UNIT			FUNCTION			PRIORITY INTERRUPT ASSIGNMENT-DATA			PRIORITY INTERRUPT ASSIGNMENT-FLAGS		
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

CONI DTC, 73224

STOP	FORWARD	REVERSE		TAPE SELECTED	NO SELECTION	UNIT			FUNCTION			PRIORITY INTERRUPT ASSIGNMENT-DATA			PRIORITY INTERRUPT ASSIGNMENT-FLAGS		
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

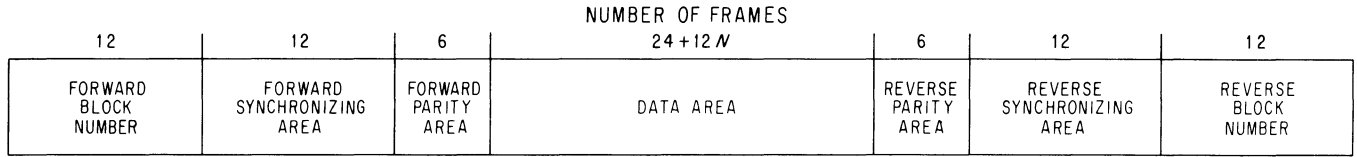
CONO DTS, 73260

PARITY ERROR	DATA MISSED	JOB DONE	ILLEGAL OPERATION	END ZONE	BLOCK MISSED		CLEAR DECTAPE SYSTEM								STOP ALL TAPES	SET FUNCTION STOP	
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

CONI DTS, 73264

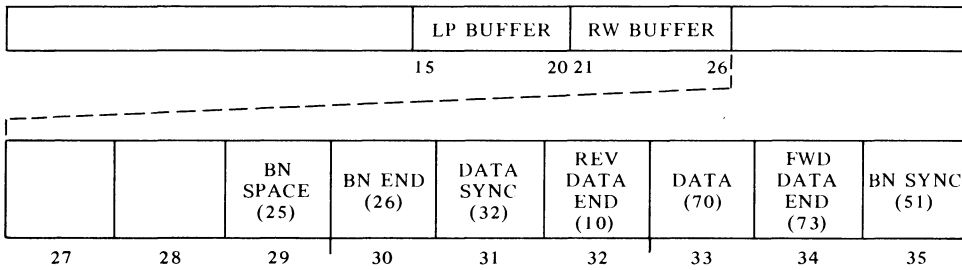
PARITY ERROR	DATA MISSED	JOB DONE	ILLEGAL OPERATION	END ZONE	BLOCK MISSED	DELAY IN PROGRESS	ACTIVE	UP TO SPEED	BLOCK NUMBER	LEADING PARITY	DATA	FINAL	TRAILING PARITY	IDLE	BLOCK NUMBER READ	FUNCTION STOP	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

*	*	*	ILLEGAL OPERATION	*	*	*			INCOMPLETE BLOCK		*	*				*	*
PARITY ERROR	DATA MISSED	JOB DONE	END ZONE	BLOCK MISSED	WRITE LOCK	WRITE MARK SWITCH			MARK TRACK ERROR	SELECT ERROR						FLAG REQUEST	DATA REQUEST
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35



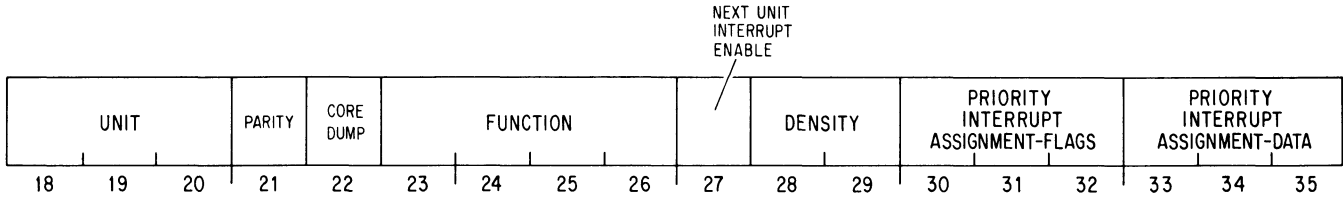
BLOCK FORMAT

- DATAO DTC,     73214   (E) → (BUFFER)  
                          0 → DATA REQUEST
- DATAI DTC,     73204   (BUFFER) → (E)  
                          0 → DATA REQUEST
- DATAO DTS,     73254   Clear time:   TPO   If (E)<sub>21</sub> = 1: 1 → TUPS0  
                          Set time:    TP1   (E)<sub>35</sub> → MARK TRACK SHIFT
- DATAI DTS,     73244   (0 indicates mark)

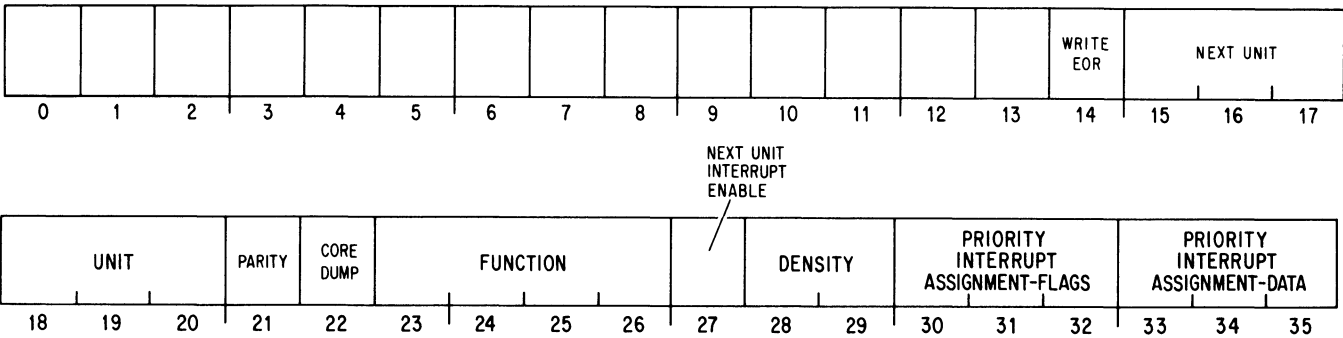


STANDARD MAGNETIC TAPE TM10  
TMC 340 TMS 344

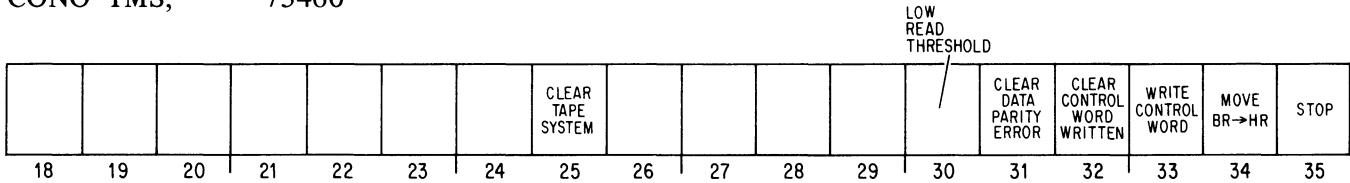
CONO TMC, 73420 0 → BR 0 → Flags



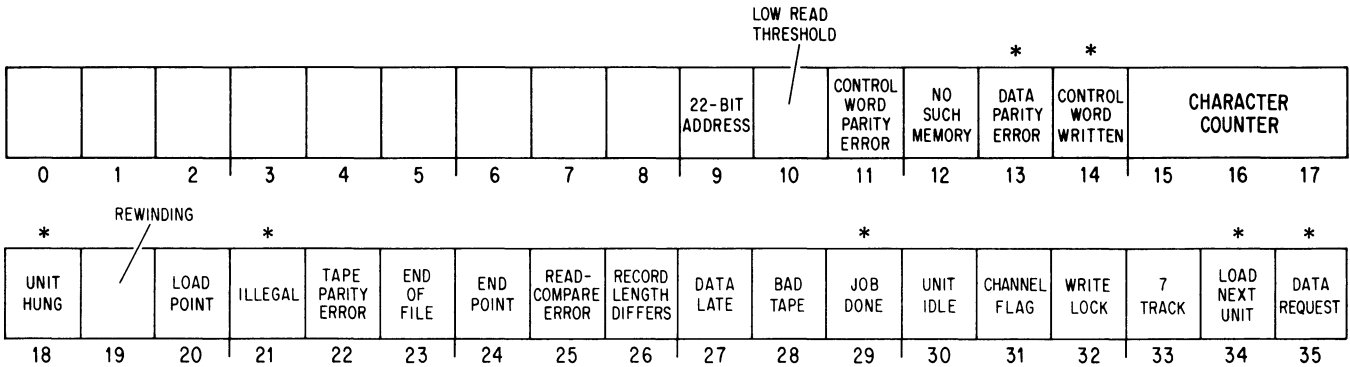
CONI TMC, 73424



CONO TMS, 73460



CONI TMS, 73464







XY10

IO BIT ASSIGNMENTS

**PLOTTER XY10**  
*PLT 140*

CONO PLT, 71420

				BUSY	DONE	PRIORITY INTERRUPT ASSIGNMENT		
27	28	29	30	31	32	33	34	35

CONI PLT, 71424

			POWER ON	BUSY	DONE	PRIORITY INTERRUPT ASSIGNMENT		
27	28	29	30	31	32	33	34	35

DATAO PLT, 71414

RAISE PEN	LOWER PEN	$-\Delta X$ (DRUM UP)	$+\Delta X$ (DRUM DOWN)	$+\Delta Y$ (CARRIAGE LEFT)	$-\Delta Y$ (CARRIAGE RIGHT)
30	31	32	33	34	35

## APPENDIX D

### TIMING

This appendix contains tables and charts for determining the time taken by instructions in PDP-10 processors. But some advice is in order before the reader turns to the timing information. It is quite likely that inspecting tables and charts for the fastest instructions takes more time than is saved using them. Moreover there are other considerations of far greater import to good programming than speed. The primary objective should be to generate code that is correct, clear and concise. Only after producing code having these qualities should the programmer concern himself with the timing; and even then there are general principles whose employment is considerably more valuable than searching through charts and tables.

The most important single factor in writing fast code is to pick a fast algorithm. Here are some guidelines.

◆ *Loops are slow.*

Always try to use one big loop instead of many small ones.

Put loops in subroutines, rather than subroutine calls in loops.

Set up data structures to minimize searching; instead index directly into tables. If direct indexing is not feasible, divide tables into sublists wherever possible.

Avoid testing inside loops for conditions that are constant throughout the loop. Avoid recomputing constants.

Use pointers rather than moving blocks of data.

◆ *UUOs are slow.* If an extra ten instructions will save you from doing a UUO half the time, use them.

◆ *IO is very slow.* Keep information in core as much as possible. When it is not possible, move data to disk, but always keep in core that part most likely to be needed next. In any event always keep track of what information is where; nothing wastes time more than bringing in the same data twice. It is also a good idea to keep track of what is in your buffers if there is any chance you will have to back up.

After finding the best procedure for doing a particular job, try to minimize the number of instructions, and following that the number of memory references. These two are obviously related, as every instruction requires at least the memory reference to fetch it. Generally speaking the fewer instructions, the faster the code will go; and there is the added benefit of fewer places for coding errors. But such rules should never be followed blindly, as there are various tradeoffs that must be taken into consideration. It is seldom useful to decrease the number of instructions or references by 10%, only to double the storage requirement in the process. For an exposition of such tradeoffs, refer to the discussion of parity procedures given in §2.11.

The reason that minimizing the number of instructions and/or memory references tends to minimize time is that for most of the simpler instructions – data handling, Boolean, test – memory access is the dominant factor in instruction time. This is not true however for those instructions that contain extensive repetitive procedures, such as multiply and divide. A single MUL may entail a dozen additions, but it requires no more memory time than a single ADD. Substituting an MUL for two or three Boolean or test instructions rarely saves time. A good rule of thumb for approximating program time is to figure about a microsecond per memory reference, and then add a factor for shift operations. The KA10 requires 150 nanoseconds per shift; the KI10 requires 110 nanoseconds, but right shifting is done two places at a time (in other words, 55 nanoseconds per place shifted right). Remember that shifting occurs not only in shift-rotate, multiply and divide instructions, but also in JFFO, byte manipulation, unnormalizing one operand with respect to the other in floating add and subtract, fixing and floating a number, and normalizing a floating point result. Note also that multiplication and division require addition or subtraction combined with shifting.

After coding using the above guidelines and suggestions, run SNOOPY and TATTLE to determine where the program is really spending its time and whether further local optimization is worthwhile. It usually is not, but where it is, use the information given in the following pages. A note of caution, however, concerning that information. No instruction times are measured. The timing charts are constructed from delay times and published times for various circuits used in the hardware. Specific instruction times listed in tables and elsewhere are calculated from the timing charts. There are therefore manifest sources of inaccuracy in the information without considering the obvious fact that no two machines ever run exactly alike. Be especially leery of attaching any significance to the last digit.

Do not assume that the fastest algorithm on one type of PDP-10 processor will be the fastest on another, or that instruction times given for one processor can be regarded as a relative indication of the times for another. Each new processor is faster than its predecessors, but different instructions are speeded up different amounts. Since the shorter instructions are dependent almost entirely on memory cycle time, it is generally the slowest instructions that get speeded up the most in a new machine. In all processors the fastest no-op is JFCL, and the fastest jump is JRST; but the fastest absolute skip is CAIA on a KA10, TRNA on a KI10. One thing a programmer can safely assume is that with each new machine, equivalent instructions will tend toward taking the same amount of time. Suppose we wish to change a single bit in the right half of AC: in the KA10 an XORI is faster than a TRC, but in the KI10 they are the same.

## KI10 INSTRUCTION TIMES

The table on the next two pages lists the processor execution time in microseconds for each instruction beginning with its address calculation. The times do not include the instruction fetch (.89 microsecond), as this is overlapped with the preceding instruction execution; in each case the processor time needed to complete the instruction fetch depends upon the extent of the overlap, a factor that varies from one instruction to another. The time listed is that required for direct addressing without indexing (*ie* with no effective address calculation), and assuming *E* addresses an ME10 or MF10 core location (1  $\mu$ s cycle), except in DFN and UFA, which are most frequently used with *E* equal to *A*+1. It is further assumed that no conflicts develop in memory access – in other words there are a number of interleaved memories, and data blocks are kept in separate memories from instructions, so the processor need never wait for operand access while a given memory completes a cycle from an instruction fetch.

To arrive at more complete execution times for various circumstances, make the following adjustments to the figures given in the table. For indexing add .06; for indirect addressing add 1.02 for each address cycle without indexing, 1.08 for each with indexing. If the final address cycle includes indexing, add .12 to JRA, POP and POPJ, and add .11 to any instruction that does not fetch a memory operand. If memory operand storage is in fast memory, subtract .08 unless there is also storage in a second accumulator, in which case add .03.

Following the table is a chart (with intervals in nanoseconds) that can be used for calculating the instruction time in almost any circumstances, with any memory, etc. However neither table nor chart includes any information about interrupts, page failures, bus conflicts between interrupts and IO instructions, or other special situations.

Memory access by the processor is divided into three parts: page check, request setup, and the actual access cycle over the memory bus. In an instruction fetch, the first two of these can be overlapped with operand storage, but not the third. The effect of this on instruction fetch time is as follows. If an instruction does not store a memory operand (either because it has no operand or stores the result in an accumulator), probably the next instruction fetch will be overlapped entirely: hence the second instruction will be ready by the time the first is done. If an instruction stores a memory operand, there is no overlap on the bus, but most likely the page check and request setup will already have been performed (these show up as the 175 preceding each read access in the chart). After the write access is complete an instruction has 147 nanoseconds to go, during which period the read access for the next instruction can begin. Finally some instructions put off triggering the next instruction fetch until near the very end, but even in the worst case (BLT) there is a minimum overlap of 87 nanoseconds, which is enough for the page check (81).

For more esoteric considerations: add 1.11 for each page refill cycle; add .09 to every page check in an instruction executed by an executive XCT, from the console, or in an interrupt; and add .20 per read access if the machine is running with the parity stop switch on.

The time given for MAP assumes no page failure.

## KI10 INSTRUCTION TIMES

		Full Words				Half Words			
EXCH	1.61	MOVE	MOVMS	1.26	Basic		1.26		
		MOVEI	MOVSI	.45	Immediate		.45		
BLT	1.35	MOVEM	MOVSM	1.06	Memory, no action		1.72		
+ <i>per word</i>		MOVES	MOVSS	1.61	Memory, some action		1.06		
Memory → memory	1.59	MOVN	MOVMS	1.32	Self		1.61		
AC → memory	1.21	MOVNI	MOVMI	.51					
Memory → AC	1.42	MOVNM	MOVMM	1.12					
		MOVNS	MOVMS	1.67					
Byte		Pushdown		Arithmetic Test		In-out			
IBP	1.90	PUSH	1.94	AOBJP	AOBJN	.57	BLKO	1.51 + DATAO	
LDB	2.87–6.72	POP	2.16	CAI		.62	BLKI	1.51 + DATAI	
DPB	3.12–4.99	PUSHJ	1.12	CAM		1.43		<i>Fast Slow</i>	
ILDB	3.54–7.39	POPJ	1.43	JUMP		.56	DATAO	3.13 4.12	
IDPB	3.80–5.67			AOJ	SOJ	.62	DATAI	CONI 2.05 3.37	
				SKIP		1.37	CONO	2.32 3.31	
				AOS	SOS	1.78	CONSO	CONSZ 1.55 2.87	
Program Control				Logical Test					
JSR	.95	TLN	.45	TRN	.34	TDN	1.15	TSN	1.26
JSP	.45	TLNE	.62	TRNE	.51	TDNE	1.32	TSNE	1.43
JRST	.34	TLNA	.56	TRNA	.45	TDNA	1.26	TSNA	1.37
JEN	.34	TLNN	.62	TRNN	.51	TDNN	1.32	TSNN	1.43
PORTAL	.34	TLZ	.56	TRZ	.45	TDZ	1.26	TSZ	1.37
JRSTF	.45	TLZE	.73	TRZE	.62	TDZE	1.43	TSZE	1.54
JSA	1.06	TLZA	.67	TRZA	.56	TDZA	1.37	TSZA	1.48
JRA	1.59	TLZN	.73	TRZN	.62	TDZN	1.43	TSZN	1.54
JFCL	.34	TLC	.56	TRC	.45	TDC	1.26	TSC	1.37
JFFO	.79–2.66	TLCE	.73	TRCE	.62	TDCE	1.43	TSCE	1.54
XCT	.34	TLCA	.67	TRCA	.56	TDCA	1.37	TSCA	1.48
LUUO	1.06	TLCN	.73	TRCN	.62	TDCN	1.43	TSCN	1.54
MUUO	2.83	TLO	.67	TRO	.56	TDO	1.32	TSO	1.48
MAP	.60	TLOE	.73	TROE	.62	TDOE	1.43	TSOE	1.54
		TLOA	.67	TROA	.56	TDOA	1.37	TSOA	1.48
		TLON	.73	TRON	.62	TDON	1.43	TSON	1.54

	Boolean				Shift-rotate			
	SETZ SETA	SETO SETCA	AND ANDCB XOR	ANDCA SETM EQV	ANDCM SETCM	IOR ORCA ORCM ORCB	Left Right Left long Right long	1.14-5.10 1.19-3.17 1.14-9.06 1.19-5.15
Basic	.45		1.26			1.37		
Immediate	.45		.45			.56		
Memory, Both	.95		1.61			1.72		

## Fixed Point Arithmetic

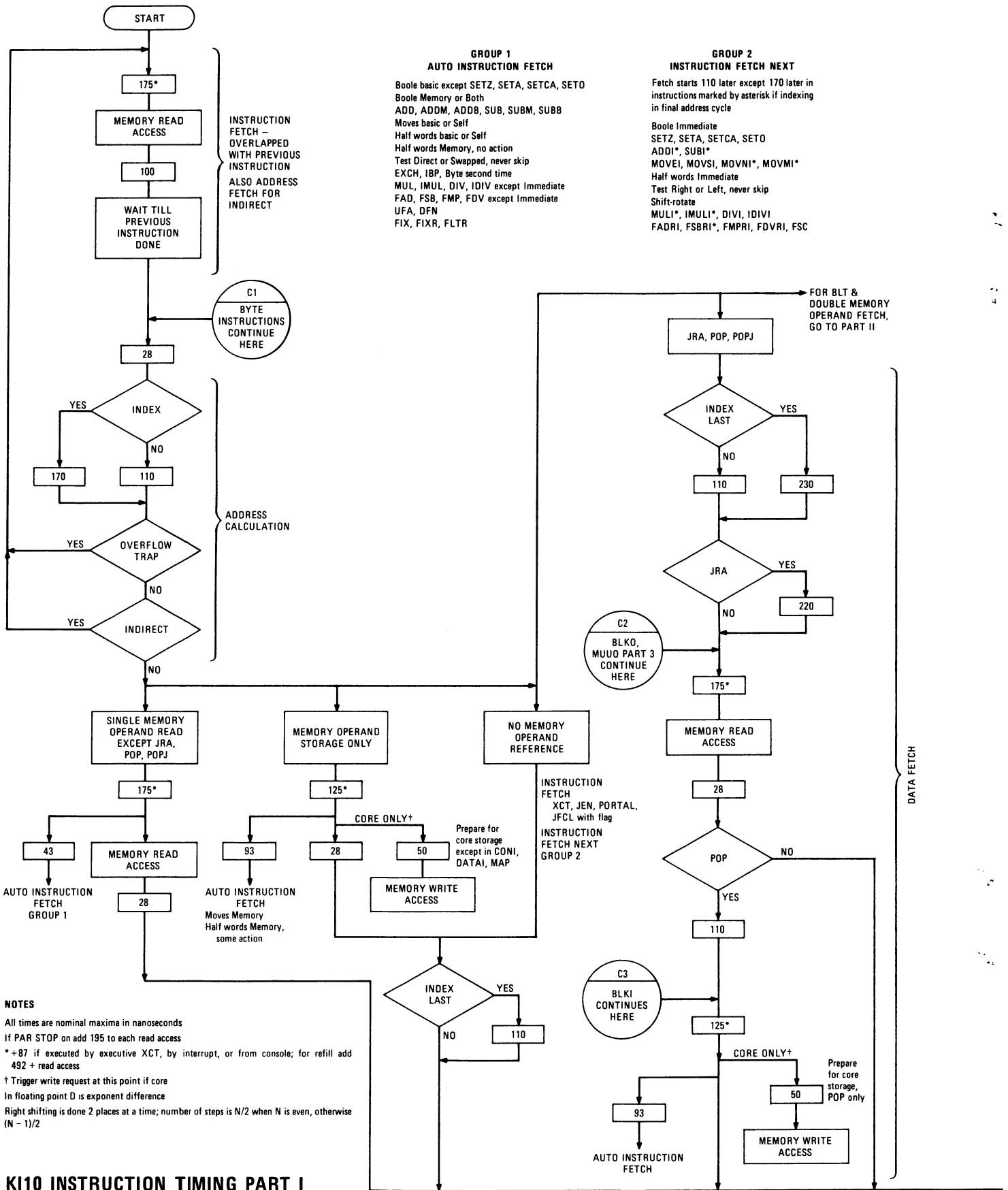
ADD SUB	1.32	MUL	3.63-7.14	DIV	8.10-8.51		
ADDI SUBI	.51	MULI	2.82-6.33	DIVI	7.29-7.70		
ADDM SUBM	1.67	MULM	3.76-7.27	DIVM	8.23-8.64	<i>No Divide</i>	
ADDB SUBB	1.67	MULB	3.87-7.38	DIVB	8.34-8.75	Immediate	.90-1.08
		IMUL	3.47-6.38	IDIV	8.16-8.45	Other	1.71-1.89
		IMULI	2.66-5.57	IDIVI	7.35-7.64		
		IMULM	3.82-6.73	IDIVM	8.29-8.58		
		IMULB	3.82-6.73	IDIVB	8.40-8.69		

## Single Precision Floating Point Arithmetic

FAD	2.45-6.20	FSB	2.62-6.37	FMP	3.65-4.83
FADL	2.79-6.54	FSBL	2.96-6.71	FMPL	3.99-5.17
FADM FADB	2.80-6.55	FSBM FSBB	2.97-6.73	FMPM FMPB	4.00-5.18
FADR	2.45-6.26	FSBR	2.62-6.43	FMPR	3.65-4.89
FADRI	1.75-5.56	FSBRI	1.98-5.79	FMPRI	2.95-3.59
FADRM FADRB	2.80-6.61	FSBRM FSBRB	2.98-6.79	FMPRM FMPRB	4.00-5.24
DFN	1.50	FDV	7.12-7.75		
UFA	1.91-3.86	FDVL	7.77-8.52	<i>No Divide</i>	
FSC	1.02, 1.19	FDVM FDVB	7.47-8.10	FDVL	2.11, 2.29
FIX FIXR	1.72-3.31	FDVR	7.49-7.95	FDVRI	1.24, 1.30
FLTR	2.10-6.07	FDVRI	6.79-7.25	Other	1.94, 2.00
		FDVRM FDVRB	7.84-8.30		

## Double Precision Floating Point Arithmetic

DFAD	2.59-7.00	DFMP	6.89-10.59	DMOVE	1.73
DFSB	2.59-7.19	DFDV	14.88-15.24	DMOVEM	1.85
		<i>No Divide</i>	2.62, 2.70	DMOVN	2.07, 2.13
				DMOVNM	2.31



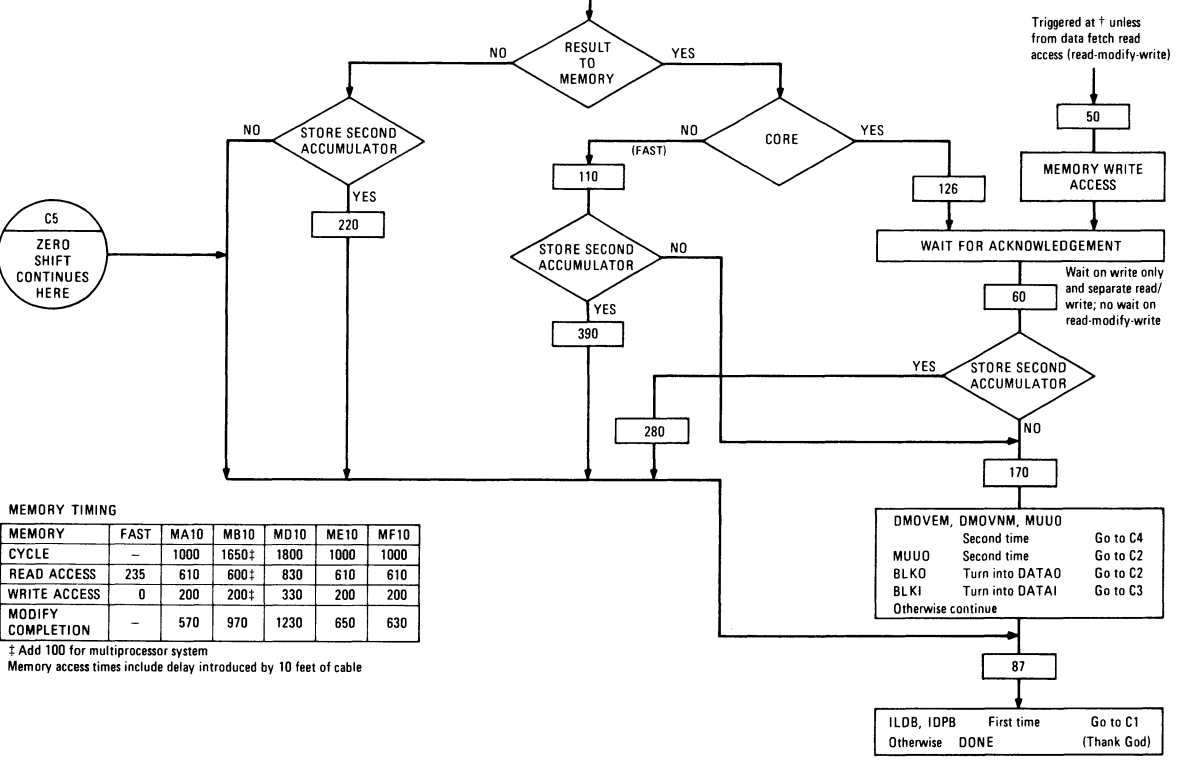
KI10 INSTRUCTION TIMING PART I



C4  
MUOU PART 2,  
DMOVEM, DMOVNM  
CONTINUE  
HERE

JRST, JEN, PORTAL, JFCL, XCT, TDN, TRN	110		
Boole except all modes IOR, ORCA, ORCM, ORCB; Half words except Memory; MOVE, MOVVS except Memory; EXCH, JSR, JSP, JRA, JRSTF; TDC, TDNA, TDZ, TLN, TRC, TRNA, TRZ, TSN; MAP		INSTRUCTION FETCH	
ADD, SUB, POPJ; MOVN, MOVNM except Memory; TDNE, TDNN, TRNE, TRNN	220	At End	
IOR, ORCA, ORCM, ORCB; MOVEM, MOVSM; Half words Memory; JSA, SKIP, JUMP; TDCA, TDO, TDDA, TDZA, TLC, TLNA, TLZ, TRCA, TRO, TROA, TRZA, TSC, TSNA, TSZ	280	JSR, JSP JRSTF AOBJP, AOBJN JFCL without flag POPJ	
AOBJP, AOBJN	330	CAM, CAI	
MOVNM, MOVMM; CAM, CAI, AOS, SOS, AOJ, SOJ; TDCE, TDCN, TDCE, TDON, TDZE, TDZN, TLNE, TLNN, TRCE, TRCN, TROE, TRON, TRZE, TRZN, TSNE, TSNN	340	SKIP, AOS, SOS JUMP, AOJ, SOJ MAP	
POP	390	110 Before End	
TLCA, TLO, TLOA, TLZA, TSCA, TSO, TSOA, TSZA	418	JSA, JRA	
TLCE, TLCN, TLOE, TLOA, TLZE, TLZN, TSCE, TSCN, TSOE, TSON, TSZE, TSZN	440	Test except never skip	
	500		
JFFO	560 + 110 × number leading 0s mod 18	Instruction fetch after 390	
PUSH	170 + 125* +† 248	Auto instruction fetch 93 after †	
PUSHJ	170 + 125* +† 248	Instruction fetch at end	
LUUU	110 + 125* +† 248	Instruction fetch 110 before end	
MUOU	First time 110 + 125* +† 248 Second time 110 + 125* +† 138 Third time 220	Instruction fetch at end	
IBP	510		
LDB, DPB	First time 270	Go to C1	
ILDB, IDPB	First time 510		
LDB, ILDB	Second time 610 + 110 × position count/2		
DPB, IDPB	Second time 170 + 110 × position count +† 330		
Shift-rotate	Left: $\begin{cases} 800 E \leq 72 \\ 740 \text{ otherwise} \end{cases}$ } Zero shift - go to C5 Nonzero: + 110 + 110 × number places Right: 740 } Zero shift - go to C5 Nonzero: + 220 + 110 × number places/2		
DMOVEM	First time 220 Second time 170 + 125* +† 248	Auto instruction fetch 93 after †	
DMOVNM	First time 620 Second time 170 + 125* +† 308	Auto instruction fetch 93 after †	
MUL	2150 + 60 per addition +† 220		
Maximum except MULI	3450 (18 adds) + 60 negative multiplier		
Maximum MULI	2910 (9 adds)		
IMUL	MUL + 60		
DIV	6280 + 180 negative dividend +† 560 + 60 negative dividend + 170 negative quotient		
No divide	670 + 180 negative dividend		
IDIV	60 - 120 negative dividend + DIV		
DFN	560		
UFA	1070 + maybe 60 to negate D + (110 + 110 per 2-place unnormalize shift) $0 < D \leq 64 + 230$ nonzero result + 170 negative result		
FAD, FADR	960 + 110 Immediate + maybe 60 to negate D + (110 + 110 per 2-place unnormalize shift) $0 < D \leq 64 + 110$ per normalize shift + (larger of 170 negative result or 230 must round) +† 110 + 230 nonzero result + 120 nonzero Long result		
FSB, FSBR	170 - 60 Immediate + FAD		
FMP, FMPI	2270 + 110 Immediate + 60 per addition + 110 unnormalized + (larger of 170 negative product or 230 must round) +† 110 + 230 nonzero product + 120 nonzero Long product		
Maximum add time except FMPI	840 (14 adds) + 60 negative multiplier		
Maximum add time FMPI	300 (5 adds)		
FSC	560 + 170 negative result + 230 nonzero result		
FLTR	830 + 110 per normalize shift + (larger of 170 negative operand or 230 must round) + 230 nonzero operand		
FIX, FIXR	340 + 110 no overflow + 230 no shift + (400 + 110 per shift) left < 9 + (280 + 110 per 2-place shift) right < 28		
FDVR	6110 + 110 Immediate + 120 negative dividend + 170  divisor  ≤  dividend  + 170 negative quotient +† 110 + 230 nonzero quotient		
No Divide	900 + 110 Immediate + 60 negative dividend		
FDV except FDVL	5740 + 120 negative dividend + 170  divisor  ≤  dividend  + 340 negative quotient +† 110 + 230 nonzero quotient		
No Divide	900 + 60 negative dividend		
FDVL	6050 + 240 negative dividend + 170  divisor  ≤  dividend  + 340 negative quotient + 460 nonzero quotient		
No Divide	1070 + 180 negative dividend		
BLKO, BLKI	340 - 60 First Part Done		
CONO, DATAO	110 +† 1870 Fast +† 110 Slow	Instruction fetch CONSO, CONSZ at end; otherwise 110 before †	
CONI, DATAI, CONSO, CONSZ	110 +† 990 Fast +† 2310 Slow		
		† Wait till 700 since last bus discharge	
		‡ Start bus discharge	

INSTRUCTION EXECUTION



DATA STORE

MEMORY TIMING

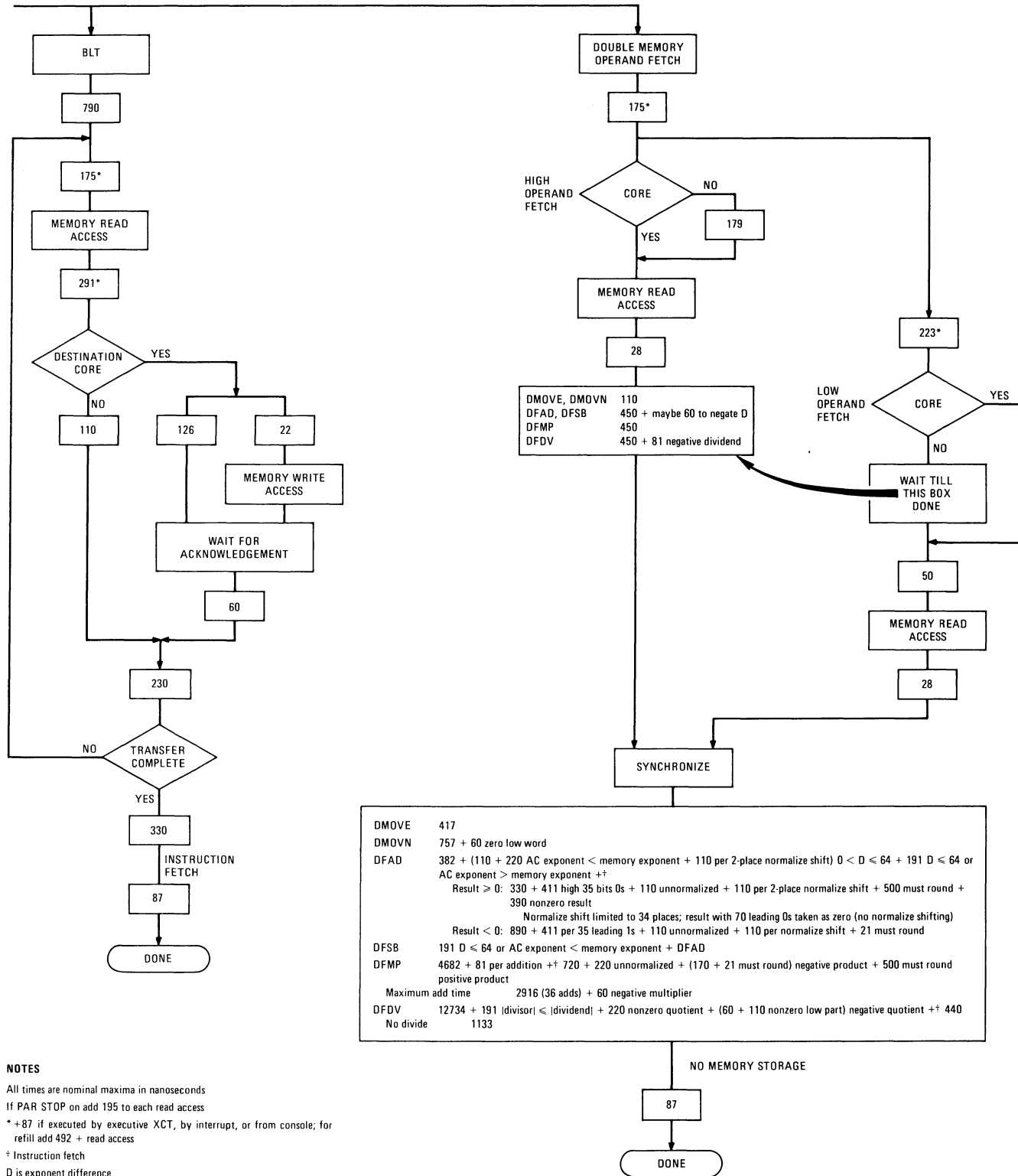
MEMORY	FAST	MA10	M810	MD10	ME10	MF10
CYCLE	-	1000	1650‡	1800	1000	1000
READ ACCESS	235	610	600‡	830	610	610
WRITE ACCESS	0	200	200‡	330	200	200
MODIFY COMPLETION	-	570	970	1230	650	630

‡ Add 100 for multiprocessor system  
Memory access times include delay introduced by 10 feet of cable

DMOVEM, DMOVNM, MUOU  
Second time Go to C4  
MUOU Second time Go to C2  
BLKO Turn into DATAO Go to C2  
BLKI Turn into DATAI Go to C3  
Otherwise continue

ILDB, IDPB First time Go to C1  
Otherwise DONE (Thank God)

FROM ADDRESS CALCULATION IN PART I



NOTES

All times are nominal maxima in nanoseconds

If PAR STOP on add 195 to each read access

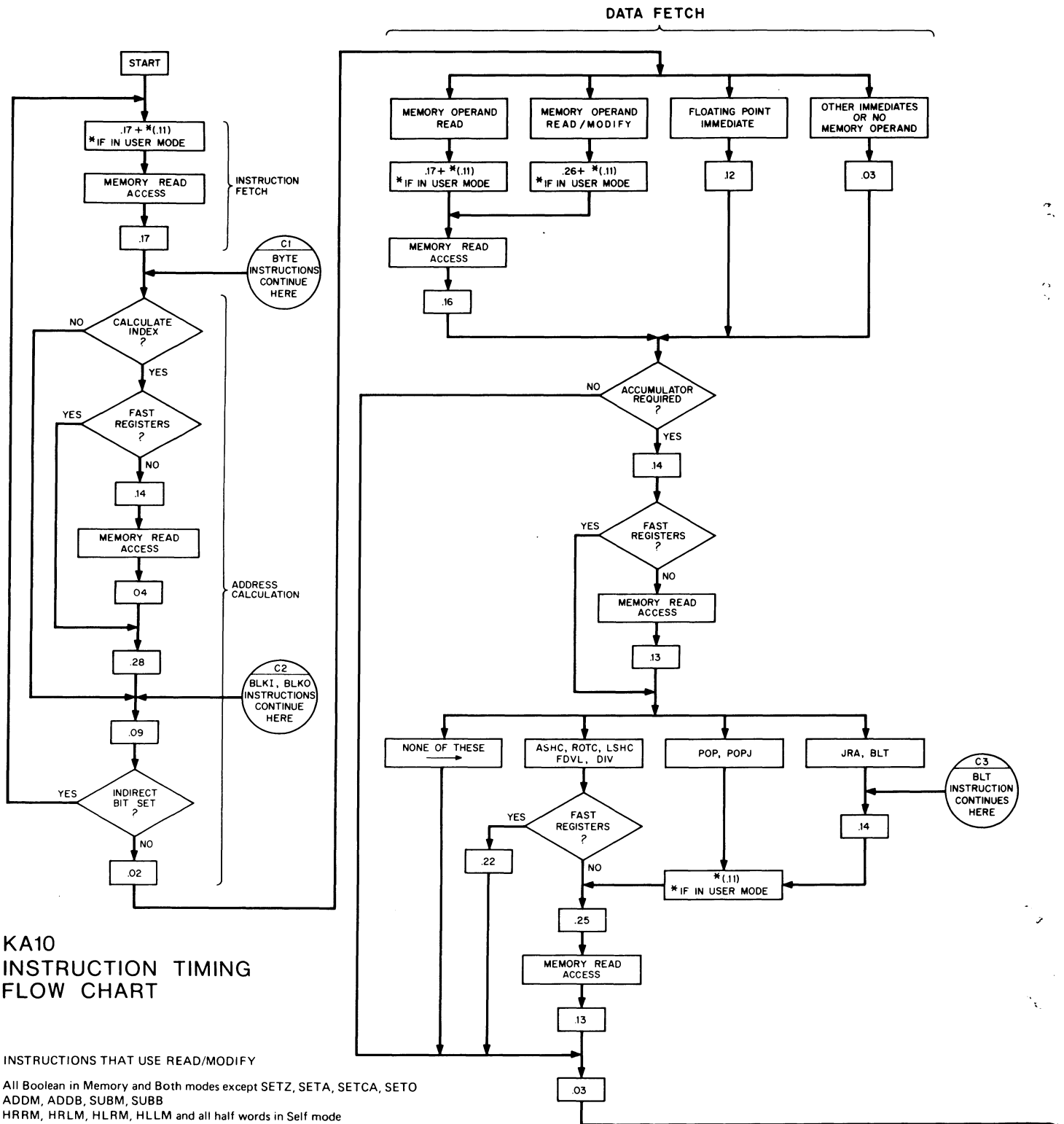
\* +87 if executed by executive XCT, by interrupt, or from console; for refill add 492 + read access

† Instruction fetch

D is exponent difference

**KA10 INSTRUCTION TIMES**

Instruction times for the KA10 can be calculated from the chart on the next two pages (intervals are in microseconds). Times derived from this chart are given with the instruction descriptions in the original *PDP-10 System Reference Manual*, which should be available to you if your system is based on a KA10. For more exact times than those given in that manual, add .06 to the listed time, plus an additional .03 for each memory operand read access, and another .03 if the instruction does not write a result in memory.



**KA10  
INSTRUCTION TIMING  
FLOW CHART**

**INSTRUCTIONS THAT USE READ/MODIFY**

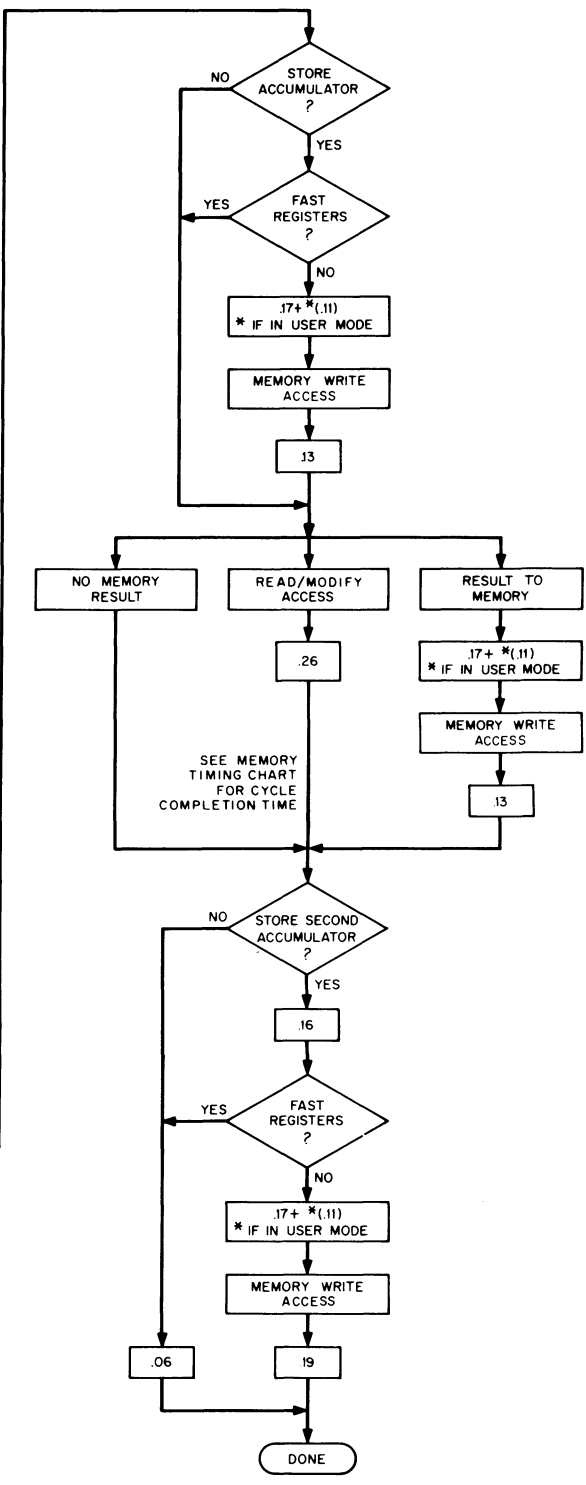
All Boolean in Memory and Both modes except SETZ, SETA, SETCA, SETO  
 ADDM, ADDB, SUBM, SUBB  
 HRRM, HRLM, HLRM, HLLM and all half words in Self mode  
 MOVES, MOVNS, MOVMS, MOVSS  
 ILDB, IDPB (first time only)  
 IBP, BLK1, BLKO, DFN, EXCH  
 AOS, SOS in all modes

INSTRUCTION EXECUTION

DATA STORE

Boolean (except ANDCA, ANDCB, ORCA, ORCB), Half Words (except HLR, HLRI, HRL, HRLI), MOVE, MOVS, EXCH, JFCL, JRST, JSP, XCT, UUU	.27	
ANDCA, ANDCB, ORCA, ORCB, HLR, HLRI, HRL, HRLI, JSR, JSA, JRA, Test class	.62	
MOVN, MOVN, ADD, SUB, AOBJP, AOBJN, CAM, CAI, SKIP, JUMP, AOJ, AOS, SOJ, SOS	.45	
PUSH, PUSHJ, POP, POPJ, DFN	.80	
JFFO	.80	+ .19 times number of leading 0s mod 18
BLT	.69	(+ .11 if User) + memory write access + .52 If not done + .09 and go to C3
IBP	.38	+ .26 if overflow word boundary
LDB, DPB	First time .61	+ .15 per size count Go to C1
ILDB, IDPB	First time .74	{ + .15 per size count } { + .26 if overflow } Go to C1
ILDB, LDB	Second time .45	+ .15 per position count
IDPB, DPB	Second time .95	+ .15 per position count
Shift group	{ .39 Left } { .23 Right }	+ .15 per shift
MUL	6.02	+ .13 per transition
Average except MULI	8.36	(18 transitions for 2.34)
IMUL	6.34	+ .13 per transition
Average except IMULI	7.51	(9 transitions for 1.17)
FMP	6.39	+ .13 per transition
Average except FMPRI	8.21	(14 transitions for 1.82)
Note: Immediate mode multiplication has only half the average number of transitions		
DIV, IDIV	13.78	
FSC	1.52	+ .25 per shift to normalize
FAD, UFA	2.38	{ + .15 per shift to unnormalize } { + .25 per shift to normalize }
Average	4.33	
FSB	Same as FAD + .18	
Rounding (except divide) only when actually done	+.96	
Long mode (except divide)	+.69	
FDVR, FDV (except FDVL)	12.00	
FDVL with fast ACs	13.28	
FDVL without fast ACs	12.32 (+ .11 if User) + memory read access + .89	
COND, CONI, CONSO, CONSZ, DATAO, DATAI	.12	Then wait until 4.50 has passed since last here
CONO, CONI, DATAO, DATAI	+2.69	
CONSO, CONSZ	+2.90	
BLKO, BLKI	.60	Then turn into DATAO, DATAI and go to C2

.03



MEMORY TIMING

MEMORY	MA10	MB10	MB10	FAST	MD10	ME10	MF10
	SINGLE OR MULTI	SINGLE	MULTI	SINGLE (BUILT IN)	SINGLE OR MULTI	SINGLE OR MULTI	SINGLE OR MULTI
PROCESSORS							
CYCLE	1.00	1.65	1.75	—	1.8	1.00	1.00
READ ACCESS	.61	.60	.70	.21	.83	.61	.61
WRITE ACCESS	.20	.20	.30	.21	.33	.20	.20
MODIFY COMPLETION	.57	.97	.97	—	1.23	.65	.63

NOTES:

MEMORY ACCESS TIMES INCLUDE DELAY INTRODUCED BY 10 FEET OF CABLE  
ALL TIMES ARE NOMINAL MAXIMUMS



## APPENDIX E

### PROCESSOR COMPATIBILITY

The table beginning below identifies the programming differences among the various central processors. The reader is forewarned not to assume that he can program a new processor simply by glancing through this table. The simpler differences, principally those associated with individual user instructions, are explained adequately in the table entries. But in more complex cases, the table entries serve only to identify the area of difference and refer the reader to the real substance in Chapter 2. In particular, all programmers, regardless of previous experience with other processors, should read Chapter 1; and all system programmers should read the later material in Chapter 2 on interrupts, processor conditions, and program and memory management.

The table is limited to programming differences, and console switches are mentioned only insofar as they affect programming. Operating differences are so extensive, that upon approaching a new processor an operator must read the complete operating information given for it in Appendix F.

	PDP-6	KA10	KI10
Address Break	No.	Switch and flag — satisfaction of the address condition sets the flag, which is a processor condition and causes an interrupt [refer to <i>CONI APR</i> , §2.14 and <i>Appendix F2 for the address conditions</i> ].	Switch but no flag — satisfaction of an address condition causes an address failure [refer to <i>Page Failure</i> , §2.15 and <i>Appendix F1 for the address conditions</i> ].
Address Failure Inhibit	Not applicable.	Not applicable.	In PC word — see Address Break.
Address stop	Address switches are compared with virtual addresses (unrelocated).	Address switches are compared with physical addresses (relocated).	Address switches are compared with virtual addresses in space selected by paging switches [ <i>Appendix F1</i> ].
Auto restart	No.	No.	Yes [§2.14].
BLKI, BLKO (also see <i>IO instructions</i> )	Pointer is incremented by adding 1000001 to it.	Same as PDP-6.	The two halves of the pointer are incremented independently.
Byte pointer incrementing	Address overflow carries into index field, and effective address calculation from the pointer uses the newly specified index register.	Address overflow carries into index field, but effective address calculation from the pointer in an IDPB or ILDB uses the originally specified index register unless an inter-	Address overflow does not carry into index field.

	PDP-6	KA10	KI10
		rupt occurs between the two parts of the instruction, in which case the new index field is used and the result is as on the PDP-6.	
Carry flags	Subtraction is done in three steps: complement minuend, add, complement sum. The resulting effect on the Carry flags is the opposite of that listed for SUB, SOJ, SOS at the beginning of §2.9. <i>Eg</i> an SOS that decrements $-2^{35}$ sets Carry 1 and has no effect on Carry 0.	Subtraction is done by directly adding the twos complement of the subtrahend, giving the Carry flag effects listed in §2.9.	Same as KA10.
Clock	Program must disable Clock interrupts when operator is using single instruction mode.	The flag is disabled while the single instruction switch is on.	Same as KA10.
Console programming	DATAI APR,.	DATAI APR, and DATAO PI, to load MI [§2.12].	Same as KA10 plus controls some switches with DATAO PTR, [§2.12] and reads switches in left half of CONI PI, [§2.13] and CONI APR, [§2.14].
D-A	Must have IO device.	Same as PDP-6.	Programmed with DATAO APR, [§2.14].
DFAD, DFSB, DFMP, DFDV	No.	No.	Yes.
DFN	No.	Yes.	Yes.
DMOVE, DMOVN	No.	No.	Yes.
FAD, FSB, FMP, FDV	<i>See floating point.</i>		
First Part Done	Set only by interrupt between parts of an IDPB or ILDB, and cleared only when the PC word is saved by an interrupt instruction or an instruction executed by a UUO. (Flag is often referred to as Byte Increment Suppression.)	Set same as PDP-6, but cleared whenever the PC word is saved.	Set by interrupt same as KA10, but also set by page failure in second part of IDPB, ILDB, DMOVEM, DMOVNM, or noninterrupt BLKO or BLKI. Cleared same as KA10.
FIX, FIXR	No.	No.	Yes.
Flags	<i>See JFCL, JRSTF, PC word, processor conditions, interrupt status, and individual flags.</i>		
Floating Overflow	No.	Yes — in PC word and processor conditions.	Yes — in PC word.



	PDP-6	KA10	KI10
Floating point	<i>(Also see overflow.)</i>		
Instructions [§2.6]	FSC plus four single precision standard operations, with and without rounding, in basic, Long, Memory and Both modes.	PDP-6 complement except Immediate replaces Long with rounding, plus UFA and DFN to facilitate software double precision operations.	KA10 complement plus number conversion (FIX, FIXR, FLTR) and hardware double precision (DMOVE, DMOVN, DMOVEM, DMOVNM, DFAD, DFSB, DFMP, DFDV).
Immediate mode	No.	Yes — operand $E,0$ , but only with rounding (FADRI, FSBRI, FMPRI, FDVRI).	Same as KA10.
Long mode			
FADL, FSBL, FMPL	In low order word, fraction begins in bit 1 (no exponent), and sign is not forced to 0.	Low order word has fraction and exponent in standard software format described in §1.1.	Same as KA10.
With rounding	Stores meaningless low order word or remainder.	Replaced by immediate mode.	Same as KA10.
FDVL	Remainder is incorrect and lacks exponent.	Correct remainder is in standard floating point format but is not normalized.	Same as KA10.
Normalization	In add, subtract and multiply, if high order word of double length result is (positive) zero, no normalization takes place. Hence except in long mode, all bits of significance are lost. In FSC, hardware does not normalize result.	Result is always normalized (except of course in UFA and DFN, and remainders and low order words are never normalized separately).	Same as KA10.
Rounding	If low order part is exactly $\frac{1}{2}$ LSB in a negative number, rounding is toward zero (decreasing magnitude).	A low order part of exactly $\frac{1}{2}$ LSB is rounded away from zero (increasing magnitude).	Same as KA10.
Floating Underflow	No.	Yes — in PC word and processor conditions.	Yes — in PC word.
FLTR	No.	No.	Yes.
FMP, FSB	<i>See floating point.</i>		
FSC	Hardware does not normalize result.	Hardware does normalize result.	Same as KA10.
HALT	MA lights display address one greater than that containing instruction that caused halt. Cannot be performed in user mode.	Address display same as PDP-6. Can be performed in user mode only if User In-out set.	AR lights display address instead. Cannot be performed in user or supervisor mode.
IBP	<i>See byte pointer incrementing.</i>		
IDIV	Overflow if dividend $-2^{35}$ .	Dividend $-2^{35}$ OK except No Divide if divisor $\pm 1$ .	Same as KA10.
IDPB, ILDB	<i>See byte pointer incrementing</i>		

	PDP-6	KA10	KI10
In-out Page Failure	Not applicable.	Not applicable.	In processor conditions.
Interrupt [§2.13]	Standard interrupt only.	Standard interrupt only.	Device returns function word that selects one of five interrupt functions described in §2.13.
Interrupt locations	Executive (physical) locations 42-57.	Executive locations 42-57, or 142-157 if offset.	Locations 42-57 in executive process table.
Interrupt instructions	Must use JSR to enter interrupt routine. Only a completed BLKX goes to second location. Only a DATAX or incomplete BLKX dismisses automatically. A condition IO instruction in first location or any IO instruction in second location hangs up processor.	Can use JSR, JSP, JSA, PUSHJ or JRST to enter interrupt routine. Otherwise same as PDP-6.	Can use JSR, JSP, PUSHJ, or MUUO to enter interrupt routine. Go to second location only when skip condition <i>not</i> satisfied in AOSX, SKIPX, SOSX, CONSX or BLKX. All other instructions dismiss automatically except that: In the second location a skip instruction whose condition is not satisfied hangs up the processor; LUUO, BLT, DMOVEM and DMOVNM will not work as interrupt instructions.
Interrupt points (besides between BLT transfers)	Before instruction fetch and each address word fetch.	After instruction fetch and each address word fetch.	After instruction done, after each address word fetch, in first half of DFDV, and when IO waiting for bus.
	The variation from one processor to another in allowable stopping points for an interrupt produces differences in the way the interrupt system responds to error situations (address break, memory protection violation, parity error, nonexistent memory). A common procedure is for the interrupt program, once it has recognized the error, to turn off the flag and get out of the processor channel quickly by switching over to a lower priority channel, dismissing the processor interrupt to the unmodified PC word. This works fine on the PDP-6 because it recognizes the lower priority interrupt before fetching the next instruction. But the greater complexity of the other processors leads to problems explained in the cautions that accompany the discussion of processor conditions in §2.14.		
Program initiated interrupts	Request dropped after interrupt.	Same as PDP-6.	Request stays on until turned off by program.
CONO PI,	22 Not used	22 Not used	22 Drop selected program requests
CONI PI,	Left half not used.	Left half not used.	Left half used for switches and program requests [§2.13].
	18 Power Failure	18 Power Failure	18 Not used
	19 Parity Error	19 Parity Error	19 Not used
	20 Parity Error Interrupt Enabled	20 Parity Error Interrupt Enabled	20 Not used
	21-27 Not used	21-27 Interrupts in progress	21-27 Interrupts in progress

	PDP-6	KA10	KI10
IO instructions	Can be performed in user mode only if User In-out set.	Same as PDP-6.	Cannot be performed in supervisor mode. Can be performed in user mode only with device codes 740 and above or if User In-out set.
JEN	Cannot be performed in user mode.	Can be performed in user mode only if User In-out set.	Cannot be performed in user or supervisor mode.
JFCL bit 12 (JFCL 1, JFOV)	PC change.	Floating Overflow.	Floating Overflow.
JFFO	No.	Yes.	Yes.
JRST 1,	Enter user mode.	Enter user mode.	PORTAL — clears Public when fetched from a nonpublic area, so is valid entry.
JRSTF (JRST 2,)	When used solely with indexing (no indirect), restores flags correctly only if previous instruction leaves left half of AR clear.	No problem.	No problem.
LUUO	<i>See UUO.</i>		
Maintenance programming	No.	No.	Yes [§2.14].
MAP	No.	No.	Yes [§2.15].
Memory management	One each, protection and relocation registers define user area. User illegal memory reference sets Memory Protection flag, a processor condition.	Two each, protection and relocation registers define a two-part user area where the high part can be write-protected [§2.16]. User illegal memory reference sets Memory Protection flag, a processor condition.	Paging hardware, where illegal memory reference causes page failure [§2.15].
Memory Protection interrupt	After an illegal user reference, the interrupt occurs before the next instruction fetch.	After an illegal user reference, the processor executes a zero instruction (UUO), which is trapped in executive location 40. The interrupt occurs after the instruction in executive location 41 is fetched.	Not applicable (page failures are trapped immediately).
Memory areas (= modes)	User (relocated) and executive (unrelocated).	Same as PDP-6.	User and executive areas divided into public and concealed areas distinguished by Public flag. User program execution thus in public or concealed mode; executive similarly in supervisor or kernel mode.

	PDP-6	KA10	KI10
Memory references	Unnecessary memory references are made in SETZ, SETO, SETA and SETCA. Eg SETZ AC,-1 fetches location 777777, which would likely be a non-existent memory reference or a protection violation.  For memory reference instructions in which the mode configuration happens to produce a no-op – such as SETMM AC,M or SKIP 0,M or TDN AC,M – all machines make the reference even though it is unnecessary.	The unnecessary references of the PDP-6 are not made.	Same as KA10.
MUL	AC supplies multiplicand, which if $-2^{35}$ is treated as though it were $+2^{35}$ .	Same as PDP-6.	AC supplies multiplier.
MUOO	<i>See UOO.</i>		
No Divide	No.	Yes – in PC word.	Same as KA10.
Overflow	Overflow (arithmetic) and Pushdown Overflow flags, which cause interrupts. Overflow conditions set flags in all circumstances.	Same as PDP-6 plus Floating Overflow, Floating Underflow and No Divide flags.	Same arithmetic flags as KA10 but no pushdown flag, and overflow handled by trapping instead of interrupts (arithmetic, Trap 1; pushdown, Trap 2). Overflow conditions set no flags in interrupt instructions.
PC Change	Yes.	No.	No.
PC word	0 Overflow  3 PC Change 7 Not used 8 Not used 9 Not used 10 Not used 11 Not used 12 Not used	0 Overflow  3 Floating Overflow 7 Not used 8 Not used 9 Not used 10 Not used 11 Floating Underflow 12 No Divide	0 Overflow in user mode, Disable Bypass in executive mode.  3 Floating Overflow 7 Public 8 Address Failure Inhibit 9 Trap 2 10 Trap 1 11 Floating Underflow 12 No Divide
Processor conditions			
CONO APR,	18 Clear Pushdown Overflow 20 Not used 21 Not used 22 Clear Memory Protection 23 Clear Nonexistent Memory 27 Disable PC Change Interrupt	18 Clear Pushdown Overflow 20 Not used 21 Clear Address Break 22 Clear Memory Protection 23 Clear Nonexistent Memory 27 Disable Floating Overflow Interrupt	18 Reset timer 20 Disable timer 21 Enable timer 22 Disable auto restart 23 Enable auto restart 27 Not used

	PDP-6	KA10	KI10
	28 Enable PC Change Interrupt	28 Enable Floating Overflow Interrupt	28 Clear In-out Page Failure
	29 Clear PC Change	29 Clear Floating Overflow	29 Clear Nonexistent Memory
	30 Disable Overflow Interrupt	30 Disable Overflow Interrupt	30-32 Processor PIA — error
	31 Enable Overflow Interrupt	31 Enable Overflow Interrupt	
	32 Clear Overflow	32 Clear Overflow	
	33-35 Processor PIA	33-35 Processor PIA	33-35 Processor PIA — clock
CONI APR,	Left half not used.	Left half not used.	Left half used for switches, etc [§2.14].
	18 Not used	18 Not used	18 Time Out
	19 Pushdown Overflow	19 Pushdown Overflow	19 Parity Error
	20 User In-out	20 User In-out	20 Parity Error Interrupt Enabled
	21 Not used	21 Address Break	21 Timer Enabled
	22 Memory Protection	22 Memory Protection	22 Power Failure
	23 Nonexistent Memory	23 Nonexistent Memory	23 Auto Restart Disabled
	28 PC Change Interrupt Enabled	28 Floating Overflow Interrupt Enabled	28 In-out Page Failure
	29 PC Change	29 Floating Overflow	29 Nonexistent Memory
	30 Not used	30 Trap Offset	30-32 Processor PIA — error
	31 Overflow Interrupt Enabled	31 Overflow Interrupt Enabled	
	32 Overflow	32 Overflow	
	33-35 Processor PIA	33-35 Processor PIA	33-35 Processor PIA — clock
DATAO APR,	No.	No.	Yes — maintenance and D-A [§2.14].
Parity Error	No.	Read by CONI PI,.	Read by CONI APR,.
POP, POPJ (also see overflow)	Pointer is decremented by subtracting 1000001 from it.	Same as PDP-6.	The two halves of the pointer are decremented independently.
POP AC,AC	AC receives decremented pointer.	AC receives word taken from stack and pointer is lost.	Same as KA10.
Power Failure	No.	Read by CONI PI,.	Read by CONI APR,; also Power Alarm and auto restart feature [§2.14].
Processor serial number	Not available.	Not available.	Can be read by CONI PAG,.
Program management	User can never give HALT or JEN, can use IO only if User In-out set. Illegal instruction executes as UUU.	IO, HALT and JEN illegal unless User In-out set, in which case all are legal. Illegal instruction executes as MUUU.	User always has IO with codes 740-774, can use all IO if User In-out set. Supervisor can never use IO. Neither can ever give HALT or JEN.

	PDP-6	KA10	KI10
			Illegal instruction executes as MUUO.
Programmable margins	No.	No.	Yes [§2.14].
Public	Not applicable.	Not applicable.	In PC word; differentiates between public and concealed modes in memory management [§2.15].
PUSH, PUSHJ (also see overflow)	Pointer is incremented by adding 1000001 to it.	Same as PDP-6.	The two halves of the pointer are incremented independently.
Pushdown Overflow	<i>See overflow.</i>		
Read in	No hardware read in; key allows access to readin area (first 16 core locations) for bootstrap.	Yes. Ends by executing the last word in the block as an instruction [§2.12].	Yes. Selects kernel mode with executive paging disabled. Ends by jumping to the location containing the last word in the block [§2.12].
Shift-rotate	Shifts number of places specified by <i>E</i> (maximum 255).	Same as PDP-6.	Eliminates redundant shifting: Arithmetic or logical shift at most 72 places; Rotate <i>E</i> mod 72 places (except 72 places if <i>E</i> a nonzero multiple of 72).
SOJ, SOS	<i>See Carry flags.</i>		
Status	<i>See PC word, processor conditions, interrupt status.</i>		
SUB	<i>See Carry flags.</i>		
Timer	No.	No.	Yes — processor conditions [§2.14].
Trap flags, trapping	No (except UUO).	No (except UUO).	Yes — for arithmetic and pushdown overflow [§2.9], page failures [§2.15], and UUO; trap flags in PC word.
Trap Offset	No.	Turning on MA TRP OFFSET switch sets flag (a processor condition) and substitutes executive locations 140-161 for MUUO, interrupt and unassigned code locations 40-61 to distinguish between two processors using same memory.	Not applicable.
UFA	No.	Yes.	Yes.
Unassigned codes	100-131, 243, 247, 257. On most machines these execute like UUOs but use executive	100-127 execute like UUOs but use executive locations 60-61 (160-161 if offset).	100-107, 114-117, 123 and 247 execute like MUUOs.

	PDP-6	KA10	KI10
	locations 60-61; on some machines they execute as no-ops (there is no standard).	247 and 257 are not regarded as unassigned and execute as no-ops unless implemented by special hardware.	
Unimplemented operations	If floating point and byte instructions are not implemented in hardware, codes 132-177 act like unassigned codes.	Turning on FP TRP switch causes floating point and byte codes 130-177 to act like unassigned codes.	All assigned codes are implemented in hardware.
User In-out	Allows IO instructions to be performed in user mode. Flag is in PC word and processor conditions.	Allows IO instructions, HALT and JEN to be performed in user mode. Flag is in PC word and processor conditions.	Allows IO instructions with device codes under 740 to be performed in user mode. Also used to control special effects in executive XCT [§2.15]. Flag is in PC word only.
UUO	All UUOs 000-077 use executive locations 40-41.	LUUOs 001-037 use user locations 40-41 in user mode, executive locations 40-41 in executive mode. MUUOs 000 and 040-077 use executive locations 40-41. (Trap offset changes executive locations 40-41 to 140-141.)	LUUOs 001-037 use user locations 40-41 in user mode, locations 40-41 in executive process table in executive mode. MUUOs 000 and 040-077 store <i>CODE A,E</i> in location 424 of the executive process table, save the PC word in 425, and restart with the processor configured according to a new PC word [ <i>for details refer to §2.10</i> ].
XCT	Same in all program modes.	Same as PDP-6.	In executive mode, can be performed as executive XCT [§2.15].

66

67

68

69



## APPENDIX F

### PROCESSOR OPERATION

Sections F1 and F2 of this appendix describe the switches and indicators used in normal operation and program debugging on the KI10 and KA10 processors respectively. Mounted on the front of the right bay of each processor are console operator and maintenance panels. Indicator panels are at the tops of the processor bays. Operating information for the memories and peripheral equipment is given in Appendices G and H. The real time clock is included in the processor discussion.

For information on the running of hardware diagnostics and the use of the switches and indicators for hardware troubleshooting, refer to the appropriate maintenance manual.

F1	KI10 Operation	F1-1
	Indicators	F1-2
	Operating keys	F1-6
	Operating switches	F1-8
	Real time clock DK10	F1-13
F2	KA10 Operation	F2-1
	Indicators	F2-1
	Operating keys	F2-3
	Operating switches	F2-7
	Real time clock DK10	F2-9

#### Cleaning the Equipment

The exterior of all equipment in the DECsystem-10 should be cleaned at least weekly. Vacuum all outside surfaces including cabinet tops and, where possible, underneath the cabinets. Pay special attention to air intake gratings such as on the top of the KI10 processor cabinets and on the bottom front of the KA10 cabinets.

#### CAUTION

When cleaning, be careful not to change the position of any switches as this could easily cause a software crash. Also be very careful not to jar any disk or drum equipment as serious head problems may result.

It is alright to use spray cleaner on exposed vertical surfaces, but do not use it around switches, near intake gratings, or near any other openings, because the "guck" can cause severe problems if it gets inside the equipment.

Interior cleaning is necessary only for certain items of peripheral equipment. Specific instructions for such cleaning are given in Appendix H7.

The "alright" in this caution applies to the sheet metal. Whether the carcinogens that come out of aerosol cans are alright for your lungs is up to you to decide. It has never been shown that the presence or absence of fingermarks or other stains has any effect whatever on the operation of the system. And anyway, it is probably much healthier to get a little exercise using something like Spic and Span.

10

11

12

13

## F1 KI10 OPERATION

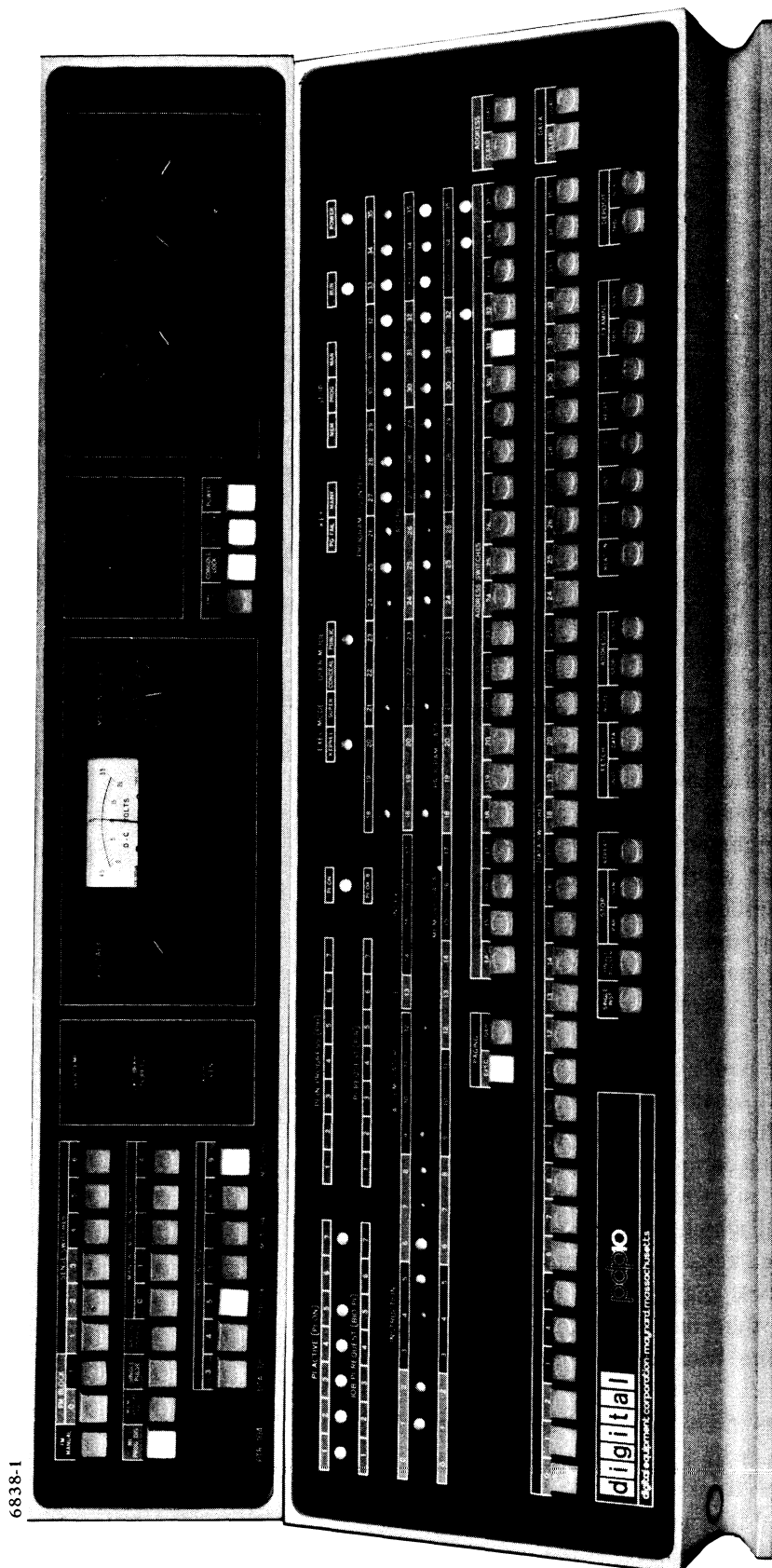
Most of the controls and indicators used for normal operation of the processor and for program debugging are located on the console operator panel, shown on the next page with the maintenance panel above it. In the upper half of the operator panel are four rows of indicators, and below them are three rows of two-position keys and switches. Physically both are push-buttons, but the keys are momentary contact whereas the switches are alternate action. Relative to the internal logic, the switches are actually flip-flops that are controlled by the buttons but which in many cases can also be "operated" by the program. A switch is on or represents a 1 when it is illuminated. Buttons that actually trigger operating sequences in the processor are the operating keys, which are located in the right half of the bottom row. Operating switches are those that supply control levels for governing various processor operations; these include the buttons in the left half of the bottom row (except SINGLE PULSER), the paging switches at the left end of the third row, and the buttons at the left in the top two rows at the left end of the maintenance panel. The remaining buttons are sense switches, groups that constitute switch registers, and various other special keys and switches that supply information to the program or to specific hardware functions, or perform special functions of various sorts separate from the normal processor operating sequence.

The thirty-six numbered switches in the second row from the bottom on the operator panel and the twenty-two numbered switches in the row above them are the data and address switches, through which the operator can supply words and addresses for the program and for use in conjunction with the operating keys and switches. At the right end of each of these switch registers is a pair of keys that clear or load all the switches in the register together. The load button sets up the switches according to the contents of the corresponding bits of the memory indicators (MI) in the fourth row. At the left end of the maintenance panel are switches to select the device for readin mode and a set of sense switches, which can be interrogated by the program.

The center section of the maintenance panel contains a voltmeter and controls for margin checking, and the right section contains speed controls for slowing down the program. Between these is a counter that registers the total time processor power has been on (the counter reads hours if the line frequency is 50 Hz, but at 60 Hz it counts six for every five hours). Below the counter are four special buttons, two of which are locks that are used to prevent inadvertent manipulation of the keys and switches while the processor is running: the console data lock disables the data and sense switches; the console lock disables all other buttons except those that are mechanical, which group comprises the four under the counter and the readin device switches.

Power is supplied to the system by means of the switch at the right end in the group under the counter. This switch is lit while power is on, but the power light in the upper right corner of the operator panel is lit only when the system is actually in operation or is ready for operation; after power turn-on the light does not come on until power is stabilized in the correct range.

A panel indicator is worthless if the bulb is burned out. Before attempting to use the information presented by the panels, press the LAMP TEST button below the counter on the maintenance panel; this turns on all of the lamps so any that are burned out can easily be detected.



At the left of the margin check controls are three red lights that indicate an overtemperature condition somewhere in the processor logic, a tripped circuit breaker, or a cooling assembly door open. Whenever any of these lights goes on the Power Failure flag sets and power automatically shuts down.

### Indicators

When any indicator is lit the associated flipflop is 1 or the associated function is true. Some indicators display useful information while the processor is running, but many change too frequently and can be discussed only in terms of the information they display when the processor is stopped. The program can stop the processor only at the completion of the HALT instruction; the operator can stop it at the end of every instruction, in certain memory references, or following every clock pulse (the last allows extremely slow speed operation with the clock running slowly or each clock pulse triggered individually by the operator).

Of the large groups of lights on the operator panel, the right half of the second row displays the contents of PC, the third row displays the instruction being executed or just completed, and the fourth row is the memory indicators. The left third of the third row displays IR; in an IO instruction the left three instruction lights are on, the remaining instruction lights and the left accumulator light are the device code, and the remaining accumulator lights complete the instruction code. The right half of the row displays the virtual address on the address bus, and the I and index lights reflect the states of the corresponding bits of the memory buffer. Hence the right two thirds of the row changes with every memory reference, and the I and

index lights actually display the indirect bit and the index register address only following an instruction fetch or an indirect reference in an effective address calculation.

Above the memory indicators appear two pairs of words, PROGRAM DATA and MEMORY DATA. If the triangular light beside the former pair is on, the indicators display a word supplied by a DATAO PI,; if any other data is displayed the light beside MEMORY DATA is on instead. While the processor is running, the addresses used for memory reference are compared with the contents of the address switches in a manner determined by the paging switches and the User Address Compare Enable flag. Whenever the two addresses are equal and the comparison is enabled, the contents of the addressed location are displayed in the memory indicators. However, once the program loads the indicators, they can be changed only by the program until the operator turns on the MI program disable switch, executes a key function that references memory, or presses the reset key [see below].

The four sets of seven lights at the left display the state of the priority interrupt channels. The PI ACTIVE lights indicate which channels are on. The IOB PI REQUEST lights indicate which channels are receiving request signals over the in-out bus; the PI REQUEST lights indicate channels on which the processor has accepted requests. Except in the case of a program-initiated interrupt, which is shown in the PI GEN lights at the left end in the bottom row on the indicator panel at the top of the console bay, a REQUEST light can go on only if the corresponding ACTIVE light is on. The PI IN PROGRESS lights indicate channels on which interrupts are currently being held; the channel that is actually being serviced is the lowest-numbered one whose light is on. When an IN PROGRESS light goes on, the corresponding REQUEST goes off and cannot go on again until IN PROGRESS goes off when the interrupt is dismissed. PI ON indicates the priority interrupt system is active, so interrupts can be started (this corresponds to CONI PI, bit 28). PI OK 8 indicates that there is no interrupt being held and no channel waiting for an interrupt; this signal is used by the real time clock to discount interrupt time while timing user programs.

The four lights at the center of the top row indicate the processor mode. One and only one of these lights can be on and they represent the combined states of the User and Public flags. The rest of the top row contains the power light and the following control indicators.

#### RUN

The processor is in normal operation with one instruction following another (although the light remains on at a stop in a memory reference). When the light goes off, the processor stops.

#### STOP MAN

The operator has stopped the processor by pressing STOP or RESET.

*Opposite:* Console Operator and Maintenance Panels

Note: If a REQUEST light stays on indefinitely with the associated IN PROGRESS light off and PC is static, check the PI CYC light on the indicator panel at the top of the console bay. If it is on, a faulty program has hung up the processor. Press RESET.

**STOP PROG**

The processor has been stopped by a HALT instruction. At the completion of the instruction the address lights display the jump address (the location from which the next instruction will be taken if the operator presses the continue key), and the AR lights at the top of bay 2 display an address one greater than that of the location containing the instruction that caused the halt.

**STOP MEM**

The processor has stopped at a memory reference. This can be due to satisfaction of an address condition selected at the console, reference to a nonexistent memory location, or detection of a parity error.

**KEY MAINT**

One of the following switches is on (this light is equivalent to CONI APR, bit 8): FM MANUAL, MEM OVERLAP DIS, SINGLE PULSE, MARGIN ENABLE, SINGLE INST, STOP PAR. Any one of these switches being on implies that the processor is being operated for maintenance purposes, and is not running at maximum speed.

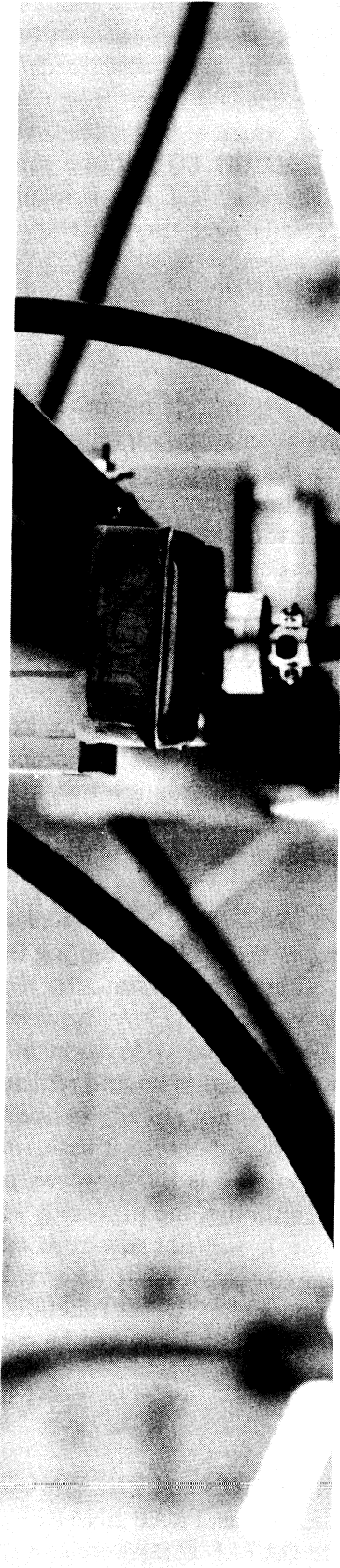
**KEY PG FAIL**

A key function has caused a page failure. No page fail trap is executed in response to a key-induced failure; if the processor is running, it continues the program.

The remaining processor lights are on the indicator panels at the tops of the bays [*illustrated on next page*]. The large groups of lights on the panel at the top of bay 2 display the contents of the adder, the AR, BR and MQ registers, and the selected location in fast memory. The bottom row displays the AR flags – FXU is Floating (exponent) Underflow, DCK is No Divide (divide check). FXU HOLD is a nonprogram flag that plays a role in determining underflow conditions. At the end is the flipflop that inhibits the clock.

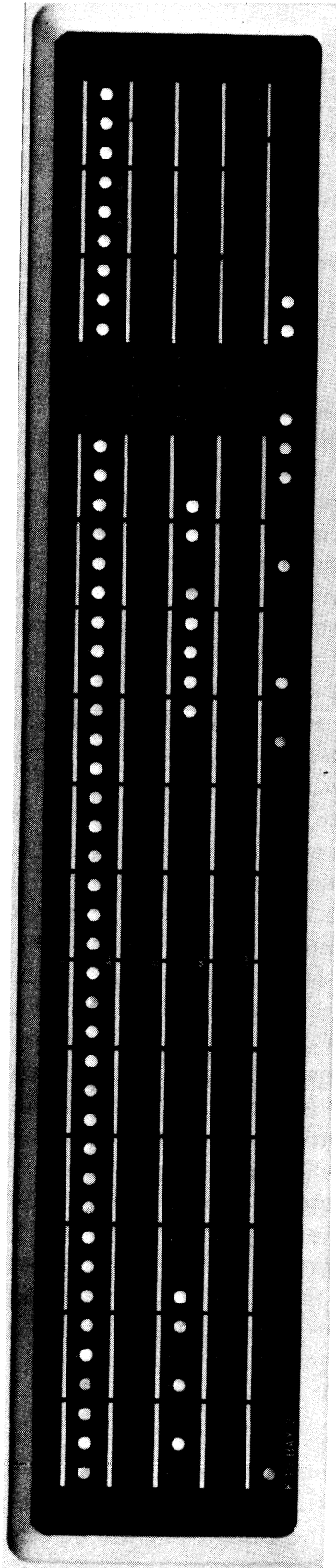
The right halves of the top two rows of the bay 1 panel display the contents of the AD and AR extensions. BYF6 in the top row is the First Part Done flag; the TN lights at the right end of the fourth row are the trap flags (TN 0 is Trap 2). The right half of the bottom row displays the physical address for each memory reference and the type of memory request. At the left are the lights for the associative memory. The AB 14–17 lights at the center are always either off or reflect the states of address switches 14–17.

The lights in the top row of the panel on the console bay (bay 3) display either the contents of the in-out bus, the paper tape reader buffer, MB, or the information supplied by the last DATAO PAG, as selected by the 4-position switch in the right section of the maintenance panel. The large groups of lights in the second row display the user and executive base registers; at the left end are the Small User and User Address Compare



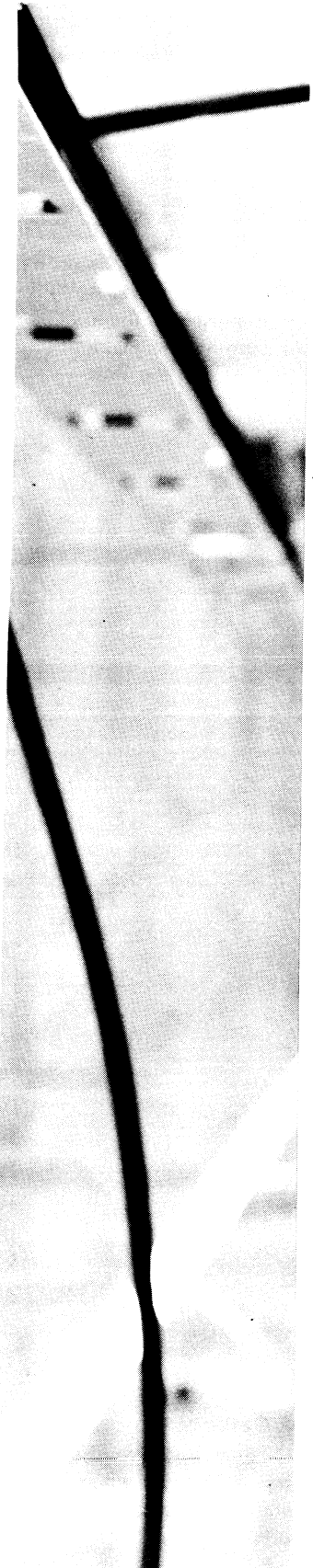
Indicator Panel, KI10 Arithmetic Processor, Bay 1

6369-1



Indicator Panel, KI10 Arithmetic Processor, Bay 2

6369-2



Indicator Panel, KI10 Arithmetic Processor, Console Bay

6369-3

Enable flags, and a pair of lights that indicate which fast memory block is currently selected for the user program. The bottom two rows include the indicators for reader, punch and console terminal, which are described in Appendix H, and the processor flags. Note that the TRAP ENABLE light at the center of the second row is the Page Enable flag, which also enables overflow traps (DATAI PAG, bit 22). PAGE LAST MUOU PUB at the very center of the panel is the Disable Bypass flag. The User IOT flag is in the middle of the third row, and COMP ADR BRK INH near the left end of the bottom row is Address Failure Inhibit.

The remaining lights on the panels are for maintenance. If the operator must use them, he should consult the maintenance manual and the flow charts.

### Operating Keys

The operating keys can be used whether RUN is on or off. If the processor is running when a key is pressed, it simply pauses at an appropriate point in the program to perform a key cycle to execute the function. These key functions are effectively of three types. The first three keys on the left are for the initiating functions, read in, start, and continue: these functions place the processor in operation under conditions determined primarily by the function itself. The next two keys are for the terminating functions, stop and reset: if the processor is running, these functions stop it. The last five keys are for the independent functions, execute, examine, examine next, deposit, and deposit next. These functions have no inherent effect on processor operation: if the processor is not running it simply performs a key cycle and stops; if it is running, it pauses to perform a key cycle and continues the program. (However the data deposited or the instruction executed may have an effect.) Moreover the independent functions are affected by the setting of the paging switches, which determine the address space in which the function is performed.

The logic responds to the keys in two stages. When a key is pressed or several are pressed simultaneously, the logic latches them. From among the buttons latched, the processor then accepts the request for the function that has priority; the priority order is the same as the order of the keys from left to right on the panel except that reset has first priority. As soon as a function request is accepted, the corresponding button lights up and remains lit until the function is completed. If the processor is not already in operation, it performs the accepted function immediately; otherwise it saves the function until it can be performed. While any button is lit, however, no function request can be accepted; in other words, although the processor will interrupt the program to perform a key function, it will not interrupt one key function for another. It will however do one key latch while a key is lit and accept the highest priority latched function once the current function is done. Provision is also made in the logic so that the RESET key can be used to stop the processor no matter what.

#### CAUTION

READ IN does not clear the associative memory, whose contents are unpredictable at power turnon.

#### READ IN

Clear all IO devices and all processor flags. Turn on RUN and EXEC MODE KERNEL (trapping and paging will both be disabled as TRAP ENABLE at the top of the console bay will be off). Execute DATAI *D*,0 where *D* is the



device code specified by the readin device switches at the left end of the maintenance panel. Then execute a series of BLKI *D,0* instructions until the left half of location 0 reaches zero. After storing the last word in the block, fetch that word as an instruction from the location in which it was stored as specified by PC. Since RUN has been set the processor begins normal operation at the location containing the last word. [*For information on the data format refer to §2.12*].

Codes of readin devices are: PTR 104, DTC 320, DTC2 330, TMC 340, TMC2 350.

### START

Turn on RUN and EXEC MODE KERNEL, and begin normal operation by fetching the instruction at the location specified by address switches 18–35. The memory subroutine for the instruction fetch loads the address into PC for the program to continue. This function does not disturb the flags or the IO equipment.

### CONT (Continue)

If STOP MEM is on begin normal operation at the point at which the processor is stopped in a memory subroutine. Otherwise turn on RUN and begin normal operation by fetching an instruction from the location specified by PC.

### STOP

Turn off RUN so the processor stops with STOP MAN on. At the stop PC points to the location of the instruction that will be fetched if CONT is pressed (this is the instruction that would have been done next had the processor not stopped).

### RESET

Clear all IO devices, disable auto restart, high speed operation and margin programming, clear the processor flags (lighting EXEC MODE KERNEL), turn on the triangular light beside MEMORY DATA (turn off the light beside PROGRAM DATA), turn off RUN and stop the processor. Do *not* clear the associative memory.

If this function is not performed within 10 ms (*eg* because READ IN is lit), the key triggers a panic reset that produces all of the standard reset actions and also clears all but the mechanical console keys and switches.

### XCT

Execute the contents of the data switches as an instruction without incrementing PC, even if a skip condition is satisfied in the instruction. If PAGING USER is on and PAGING EXEC is off, execute the instruction in user virtual address space; otherwise use executive address space. If the instruction is an XCT or LUUO, the instruction called by it is also executed.

The rightmost device switch is for bit 9 of the instruction and thus selects the least significant octal digit (which is always 0 or 4) in the device code.

### CAUTION

Note that the key function lasts throughout the processing of the entire block. This means that read in cannot be interrupted for another key function. Hence if it must be stopped (*eg* because of a crumpled tape), press RESET.

The memory restart is not a key function in the sense defined above. In other words, use of CONT to continue at a memory stop is not subject to the restrictions given above for use of the operating keys.

The processor may stop in the middle of a two-part instruction, but pressing CONT restarts the instruction without repeating any first-part actions that would adversely affect the result.

If STOP ever fails to stop the processor, pressing this key will, but not without destroying information. To save the processor state, stop by pressing SINGLE INST and SINGLE PULSE simultaneously.

Note that an instruction executed from the console can alter the processor state like

an instruction in the program: it can halt the processor, can change PC by jumping, alter the flags, or even cause a non-existent memory stop (but not a page fail trap, even if it turns on the KEY PG FAIL light).

#### NOTE

The remaining key functions all reference memory. They can therefore light KEY PG FAIL and set such flags as Nonexistent Memory and Parity Error, and they all turn on the triangular light beside MEMORY DATA, turning off the light beside PROGRAM DATA. *Performing one of these functions with the ADDRESS STOP switch on stops the processor in the memory subroutine (with STOP MEM on).*

These functions use an address supplied by the address switches, and the way that address is interpreted is determined by the paging switches. If both paging switches are off, the function uses a 22-bit absolute physical address supplied by address switches 14–35, and fast memory references are made to the block selected by the FM block switches at the left end of the maintenance panel. If either paging switch is set, the function uses a virtual address supplied by address switches 18–35 and the FM block switches have no effect (in other words the function has access to one of the virtual address spaces defined for a normal program). If PAGING EXEC is on, the function has access to executive address space; if PAGING EXEC is off and PAGING USER is on, the function has access to user address space.

#### EXAMINE THIS

Display the contents of the location specified by the paging and address switches in the memory indicators.

#### EXAMINE NEXT

Add 1 to the address displayed in the address switches, and display the contents of the location then specified by the paging and address switches in the memory indicators.

#### DEPOSIT

Deposit the contents of the data switches in the location specified by the paging and address switches, and display the word deposited in the memory indicators.

#### DEPOSIT NEXT

Add 1 to the address displayed in the address switches, deposit the contents of the data switches in the location then specified by the paging and address switches, and display the word deposited in the memory indicators.

### Operating Switches

Besides defining the address space for the independent key functions, the paging switches also perform this service for address comparison and for the group of five switches just at the left of the operating keys. Whenever the processor references memory or an accumulator, it may compare the virtual address used with that specified by address switches 18–35 and may take some action if the two are identical. There are a number of conditions that

affect the comparison. First, comparison can be made only for memory references and accumulator write references – there is never a comparison for an index register reference or an accumulator read reference. Given the proper type of reference, the comparison must be enabled by the paging switches and the User Address Compare Enable flag, as described below. In a reference of the correct type with the comparison enabled, if the virtual address on the address bus or the fast memory address is identical to the address in switches 18–35, the processor displays the contents of the addressed location or accumulator in the memory indicators (unless the light beside PROGRAM DATA is on).

Except in an AC reference, the same situation that causes the word display can also be made to stop the processor or produce an address failure, depending upon the purpose of the reference as selected by the three address condition switches. The logic that implements the address stop conditions differs from that for the address break conditions in the data fetch case (the break conditions are a subset of the stop conditions). However the differences in the statement of the conditions appear quite large. This is because the conditions are stated in terms of their consequences. And the consequences differ considerably because an address failure occurs in the page check that is done when a memory reference is requested, whereas an address stop occurs after a memory reference is actually made.

The address conditions for a failure are explained in detail in §2.15. Whenever there is a page check for a memory reference that satisfies both the comparison conditions and any selected address condition, ADDRESS BREAK being on causes an address failure except in an instruction performed while COMP ADR BRK INH is on.

Whenever the processor actually makes a memory reference that satisfies both the comparison conditions and any selected address condition, ADDRESS STOP being on halts the processor with STOP MEM on and PC pointing to the instruction that is being performed (running with ADDRESS STOP on slows down the processor). The stop conditions selected by the address condition switches are as follows:

FETCH INST selects access for retrieval of an ordinary instruction, including an instruction executed by an XCT or an LUUO (address 41), and a page refill for same.

FETCH DATA selects access for retrieval of an address word in an effective address calculation, any retrieval of an operand other than in an XCT (read-only and in the read part of a read-modify-write), retrieval of a dispatch interrupt instruction, and a page refill for any of these and for any of the conditions selected by the WRITE switch (*ie* any reference except an instruction fetch). This switch can also cause a stop inadvertently on the retrieval of a trap instruction, a PC word in an MUUO, or a standard interrupt instruction.

WRITE selects access for writing, both write-only and read-modify-write, including writing by an LUUO (address 40), a page refill for any of these, and also for retrieval of the operand in a read-modify-write – in other words the processor stops separately on the read and write parts of a read-modify-write. This switch also causes a stop on the first write in an

When the ADDRESS BREAK and ADDRESS STOP switches are both on, the former has precedence because the page failure cancels the requested access.

MUO if the address switches contain the effective address of the MUO (even though that address is not used for the access), and can cause a failure inadvertently on the second write.

ADDRESS STOP also stops any examine or deposit function in the memory subroutine.

The way the paging switches enable the comparison is as follows. If PAGING EXEC is on and PAGING USER is off, the comparison is enabled for executive address space. If PAGING EXEC is off and PAGING USER is on, the comparison is enabled for user address space provided the program has turned on USER ADR COMP (User Address Compare Enable flag) in the upper left corner of the bay 3 indicator panel. If both paging switches are on, the comparison is enabled for executive address space, provided USER ADR COMP is on (in other words with both switches on, PAGING USER applies the flag condition to PAGING EXEC).

Displaying the contents of a selected location and catching a particular type of reference to a selected location, as described above, are traditional debugging techniques. The paging switches allow these techniques to be used more flexibly in a large system that handles many users. The configuration PAGING EXEC on and PAGING USER off would be used for debugging the Monitor itself or some other executive program, which quite likely would be the only program running. PAGING EXEC off and PAGING USER on limits the procedures to user address space; and control over the comparison by the executive through a flag allows debugging an individual user program without interfering with either the executive or other users. Similarly both switches on allows investigation of that part of the executive associated with a given user, interfering with neither the rest of the executive nor any user. One who uses these switches often works in conjunction with a debugging or diagnostic program, and in the flag-limited cases one would be more apt to use the address break than the address stop, as the latter terminates all operations.

Conditions associated with the comparison are displayed by the COMP lights in the middle of the bay 3 indicator panel. From left to right these indicate an accumulator write reference, a memory read reference, equal addresses in a synchronous reference (an operand reference, but limited to the first in a double operand), and equal addresses in an asynchronous reference (an instruction fetch or the second in a double operand).

The description of each of the remaining switches relates the action it produces while it is on.

#### SINGLE INST

Whenever the processor is placed in operation, clear RUN so that it stops at the end of the first instruction. Hence the operator can step through a program one instruction at a time, pressing START for the first one and CONT for subsequent ones. Each time the processor stops, the lights display the same information as when STOP is pressed.

APR CLK FLAG (Clock flag) on the bay 3 indicator panel is held off to prevent clock interrupts while SINGLE INST is on. Otherwise interrupts would occur at a faster rate than the instructions.

Besides controlling USER ADR COMP with a DATAO PAG, a debugging program can directly manipulate the paging, address, address condition, and address break switches by means of a DATAO PTR,. But for the program to control address stopping (other than by USER ADR COMP), the operator must turn the switch on, and the program can then inhibit its effect by turning off all three address condition switches. Should it be preferred that the address condition be controlled solely by the operator, the program can still disable the stop by setting the address switches to a number that is unlikely to appear on the address bus, such as zero, or better still an address greater than any used in the program. It might seem that an address all 1s is a good candidate for this purpose, but it is in fact a very poor choice and results in inadvertent stops at traps, MUOs and the like. The reason for this is that various types of special access do not use the address bus; and when the bus is not used, it is generally left free to follow the adder, which in turn puts out all 1s when neither of its input mixers is enabled.

Note that read in cannot be done in single instruction mode, as the function extends over many instructions and there is thus no way to continue.

*CAUTION*

It is not generally worthwhile to attempt to use the interrupt system in single instruction mode except with the slowest start-stop devices, such as reader, punch and teletypewriter. In any event an interrupt hangs up the processor, and the operator must dispose of it manually before single instruction operation can continue.

SINGLE INST will not stop the processor if a hangup prevents it from getting to the end of an instruction. Use STOP, RESET, or SINGLE PULSE.

**SINGLE PULSE**

Inhibit the clock so that a single clock pulse is generated each time SINGLE PULSER is pressed. If the processor is not already in operation, an operating key must be pressed before SINGLE PULSER can be used. If the processor is running, it converts to single pulse operation at the beginning of the instruction cycle; hence the clock will not stop if the processor does not reach the instruction cycle, say because it is hung up in a multiply or divide sequence. To force the processor into single pulse operation regardless of its position in the operating sequence, turn on both SINGLE INST and SINGLE PULSE — this stops the processor dead in its tracks.

This type of stop destroys no information, the way pressing RESET would.

**STOP PAR**

Stop with STOP MEM on at the end of any memory reference in which even parity is detected in a word read. A parity stop is indicated by the following: PAR ERR FLAG (Parity Error flag) is on in the bottom row on the bay 3 indicator panel; and among the PAR lights in the third row from the bottom, ERR is on, IGN (ignore parity) is off, and BIT displays the parity bit for the word read. MA points to the location in which the error occurred.

If IGN is on (it displays a signal from the memory), parity errors are not detected and no stop can occur. Running with STOP PAR on slows down the processor.

**STOP NXM**

Stop with STOP MEM on if a memory reference is attempted but the memory does not respond within 100  $\mu$ s. This type of stop is indicated by FLAGS NXM (Nonexistent Memory flag) being on in the bottom row on the bay 3 indicator panel.

**REPEAT**

If SINGLE PULSE is on and the processor is placed in operation, slow down the clock so that the processor runs at a clock rate determined by the speed controls at the right end of the maintenance panel. If the processor is not already running, it can be placed in single-pulse repeat operation by pressing an operating key and then pressing SINGLE PULSER. If the processor is running and the switches are turned on in the order REPEAT/SINGLE PULSE, then it goes into single pulse operation automatically at the beginning of the instruction cycle. If the processor is running with REPEAT off, it stops at the beginning of the instruction cycle when SINGLE PULSE is turned on; to restart it, turn on REPEAT and then press SINGLE PULSER twice. The lamp in the SINGLE PULSER button goes off at each clock pulse and turns back on each time the clock is retriggered; hence the button

glows with an intensity that is relative to the clock duty cycle (eg for a given speed, the light will be dimmer for a program with many memory references). When either REPEAT or SINGLE PULSE is turned off, operation terminates after one more clock.

If SINGLE PULSE is off and any operating key is pressed, then every time the repeat delay can be retriggered, wait a period of time determined by the setting of the speed control and repeat the given key function. The point at which the processor can restart the repeat delay depends upon the type of key function being repeated as follows.

For an initiating function the delay starts when the processor stops with RUN off. This is either when the program gives a HALT instruction (STOP PROG) or following the first instruction if SINGLE INST is on.

For an independent function the delay starts every time the function is done whether RUN is on or off.

A terminating function stops the processor and the delay starts every time the function is repeated. Reset is generally used only to provide a chain of reset pulses on the IO bus, and stop is used to troubleshoot the clock.

In any case continue to repeat the function until REPEAT is turned off.

The speed control includes a six-position switch that selects the delay range and a potentiometer for fine adjustment within the range. Delay ranges are as follows.

<i>Position</i>	<i>Range</i>
1	200 ns to 2 $\mu$ s
2	2 $\mu$ s to 20 $\mu$ s
3	20 $\mu$ s to 500 $\mu$ s
4	500 $\mu$ s to 6 ms
5	6 ms to 160 ms
6	160 ms to 4 seconds

So long as REPEAT remains on, the selected key remains lit and its function continues in effect. In other words the operating keys are disabled.

The function is often repeated once more after the switch is turned off, but this is noticeable only with very long repeat delays.

The remaining switches are located at the left end of the maintenance panel.

#### FM MANUAL

All fast memory references for any purpose (index register, accumulator, memory) and under any conditions are made to the fast memory block selected by the FM BLOCK switches. When FM MANUAL is off, the block switches control fast memory references only in examine and deposit type key functions with both paging switches off (*ie* with the function using physical addressing). Turning on FM MANUAL overrides all other conditions so that all fast memory references are controlled by the block switches.

#### MI PROG DIS

Turn on the triangular light beside MEMORY DATA (turn off the light beside PROGRAM DATA) and inhibit the program from loading any switches or displaying any information in the memory indicators. The indicators will thus continually display the contents of locations selected from the console.

**MEM OVERLAP DIS**

Prevent memory control from overlapping cycles on the memory bus.

**MARGIN ENABLE**

Enable maintenance operation, including writing with even parity in memory and checking speed or voltage margins. Maintenance actions attempted by the program are indicated by the last four lights on the left end of the second row from the bottom on the bay 3 indicator panel. With maintenance operation enabled, writing with even parity and checking speed margins are otherwise entirely under program control. Voltage margins may be checked by the program or the operator.

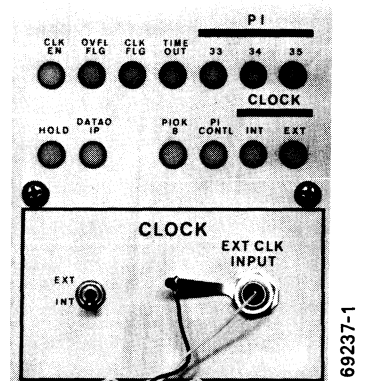
**Real Time Clock DK10**

The real time clock for the KI10 is usually installed under the console operator panel in bay 3 and has a small control panel mounted directly on the logic behind the cabinet door. In the lower part of the panel is a switch for selecting the internal source or an external input from the BNC connector at the right. The external input must be supplied through a 100 ohm coaxial cable and must have a frequency no greater than 400 kHz; its triggering voltage change must be from  $-3$  volts to ground. If the input is a pulse train, the minimum pulse width is 100 ns. If the input is a sequence of level changes, it must have a minimum low level ( $-3$  volts) duration of 400 ns before each positive-going change, a rise time of 60 ns maximum, and a high level duration of 40 ns minimum.

The leftmost light in the upper row at the top of the panel indicates when the clock is on (*ie* when the counter is enabled). The next two lights are the Count Overflow and Count Done flags. TIME OUT indicates when the numbers in the interval register and the clock counter are identical — this light goes out as soon as either changes state. The remaining lights in the upper row are the PI assignment. The two lights at the left in the lower row display signals that synchronize the DATAI and DATAO to the clock so that counting is postponed while the counter is being read and there is no sampling while the interval is being loaded. PI OK 8 is a processor-generated signal which indicates that there is no interrupt being held and no channel waiting for an interrupt; the next light is the User Time flag. The final two lights indicate the origin of the clock source.

This has no effect on pipelining within memory control, such as overlapping the page checking of consecutive memory subroutines.

For information on maintenance operation, including use of the MARGIN SELECT and MANUAL MARGIN ADDRESS switches, refer to Chapter 10 of the maintenance manual.



Clock Control Panel





## F2 KA10 OPERATION

Most of the controls and indicators used for normal operation of the processor and for program debugging are located on the console operator panel shown here. The indicators are on the vertical part of the panel; in front of them are two rows of two-position keys and switches (keys are momentary contact, switches are alternate action). A key or switch is on or represents a 1 when the front part is down.

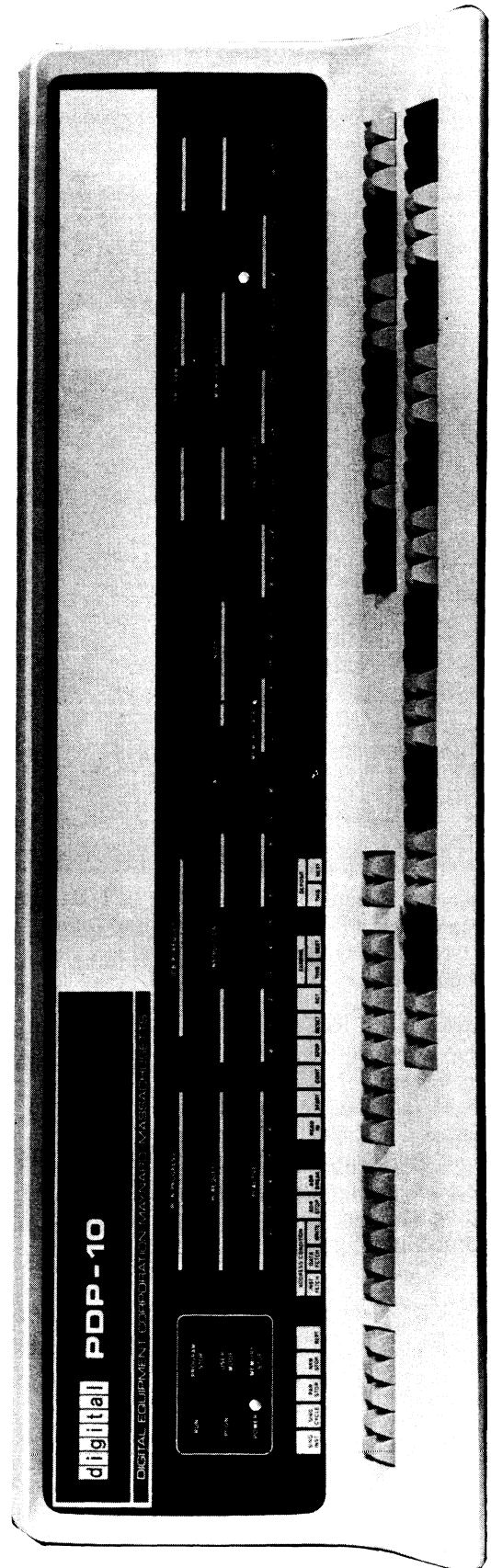
The thirty-six switches in the front row and the eighteen at the right in the back row are respectively the data and address switches through which the operator can supply words and addresses for the program and for use in conjunction with the operating keys and switches. The correspondence of switches to bit positions is indicated by the numbers at the bottom row of lights. At the left end of the back row are ten operating switches, which supply continuous control levels to the processor. At their right are ten operating keys, which initiate or terminate operations in the processor. The names of the operating keys and switches appear on the vertical part of the panel below the lights.

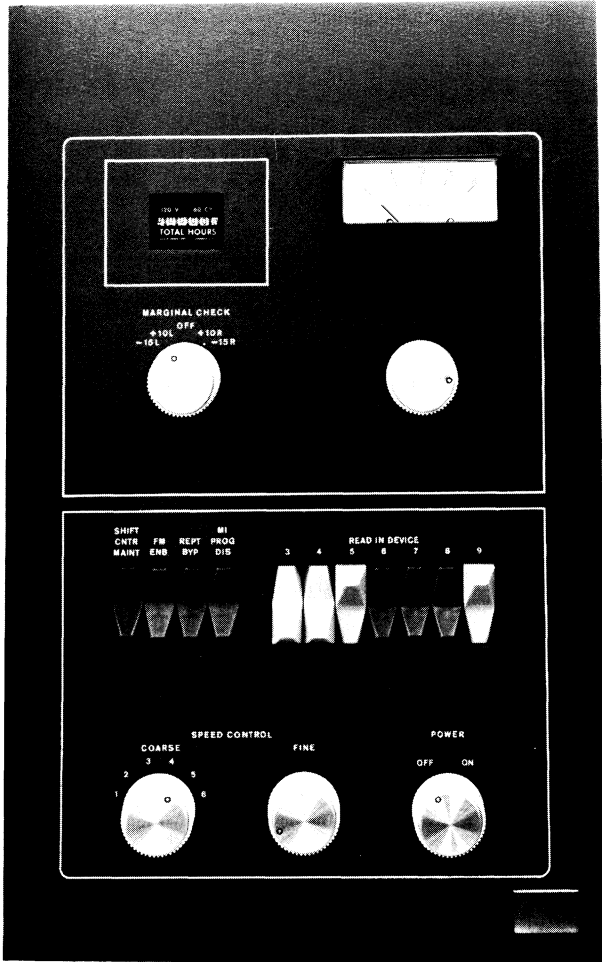
Also of interest to the operator is the small panel shown overleaf, which is located above the main panel at the left of the tape reader. The upper section of this panel contains a total hours meter and the margin-check controls. The lower section contains the power switch, speed controls for slowing down the program, switches to select the device for readin mode (lower part in represents a 1), and four additional operating switches. The normal position for these last four is with the upper part in; in this position FM ENB (fast memory enable) is on, the others are all off.

### Indicators

When any indicator is lit the associated flipflop is 1 or the associated function is true. Some indicators display useful information while the processor is running, but many change too frequently and can be discussed only in terms of the information they display when the processor is stopped. The program can stop the processor only at the completion of the HALT instruction; the operator can stop it at the end of every instruction or memory reference, or for maintenance purposes, after every step in any operation that uses the shift counter (shifting, multiplication, division, byte manipulation).

Of the long rows of lights at the right on the operator panel, the top row displays the contents of PC, the middle row displays the instruction being executed or just completed, and the bottom row are the memory indicators. The right half of the middle row displays MA, the left half displays IR.





*Above:* Margin Check and Maintenance Panel

*Overleaf:* Console Operator Panel

Note: If a REQUEST light stays on indefinitely with the associated IN PROGRESS light off and PC is static, check the PI CYC light on the indicator panel at the top of bay 2. If it is on, a faulty program has hung up the processor. Press STOP.

In an IO instruction the left three instruction lights are on, the remaining instruction lights and the left AC light are the device code, and the remaining AC lights complete the instruction code. The I, index and MA lights change with each indirect reference in an effective address calculation; at the end of an instruction I is always off.

Above the memory indicators appear two pairs of words, PROGRAM DATA and MEMORY DATA. If the triangular light beside the former pair is on, the indicators display a word supplied by a DATAO PI; if any other data is displayed the light beside MEMORY DATA is on instead. While the processor is running the physical addresses used for memory reference (the relocated address whenever relocation is in effect) are compared with the contents of the address switches. Whenever the two are equal the contents of the addressed location are displayed in the memory indicators. However, once the program loads the indicators, they can be changed only by the program until the operator turns on the MI program disable switch, executes a key function that references memory, or presses the reset key (see below).

The four sets of seven lights at the left display the state of the priority interrupt channels [§2.13]. The PI ACTIVE lights indicate which channels are on. The IOB PI REQUEST lights indicate which channels are receiving request signals over the in-out bus; the

PI REQUEST lights indicate channels on which the processor has accepted requests. Except in the case of a program-initiated interrupt, a REQUEST light can go on only if the corresponding ACTIVE light is on. The PI IN PROGRESS lights indicate channels on which interrupts are currently being held; the channel that is actually being serviced is the lowest-numbered one whose light is on. When an IN PROGRESS light goes on, the corresponding REQUEST goes off and cannot go on again until IN PROGRESS goes off when the interrupt is dismissed.

At the left end of the panel are a power light and these control indicators.

### RUN

The processor is in normal operation with one instruction following another. When the light goes off, the processor stops.

### PI ON

The priority interrupt system is active so interrupts can be started (this corresponds to CONI PI, bit 28).

**PROGRAM STOP**

IR now contains a HALT instruction. If RUN is off, MA displays an address one greater than that of the location containing the instruction that caused the halt, and PC displays the jump address (the location from which the next instruction will be taken if the operator presses the continue key).

If RUN and PROGRAM STOP are both on, the processor is probably in an indirect address loop. Press STOP.

**USER MODE**

The processor is in user mode (this corresponds to bit 5 of a PC word).

**MEMORY STOP**

The processor has stopped at a memory reference. This can be due to single cycle operation, satisfaction of an address condition selected at the console, reference to a nonexistent memory location, or detection of a parity error.

The remaining processor lights are on the indicator panels at the tops of the bays [*illustrated on next page*]. Bay 2 displays AR, BR and MQ, the output of the AR adder, and the parity buffer which receives every word transmitted over the memory bus. The RL and PR lights at the lower right display the relocation and protection registers for the low part of the area assigned to a user program and the left eight bits of the relocated address for that part.

The upper four rows on the bay 1 panel include the indicators for reader, punch and terminal, which are described in Appendix H1. The bottom row displays the information on the data lines in the IO bus. The AR lights at the upper right are the flags – FXU is Floating (exponent) Underflow, DCK is No Divide (divide check). OV COND is the condition that the 0 and 1 carries are different, *ie* the condition that indicates overflow. The First Part Done flag is BYF6 in the MISC lights in the top row; User In-out is IOT USER in the EX lights at the center of the panel. The CPA lights in the top row and the five lights under them at the left are the processor conditions – PDL OV is Pushdown (list) Overflow. The AS= lights in the middle row indicate when the (relocated) core memory address or the fast memory address is the same as the address switches.

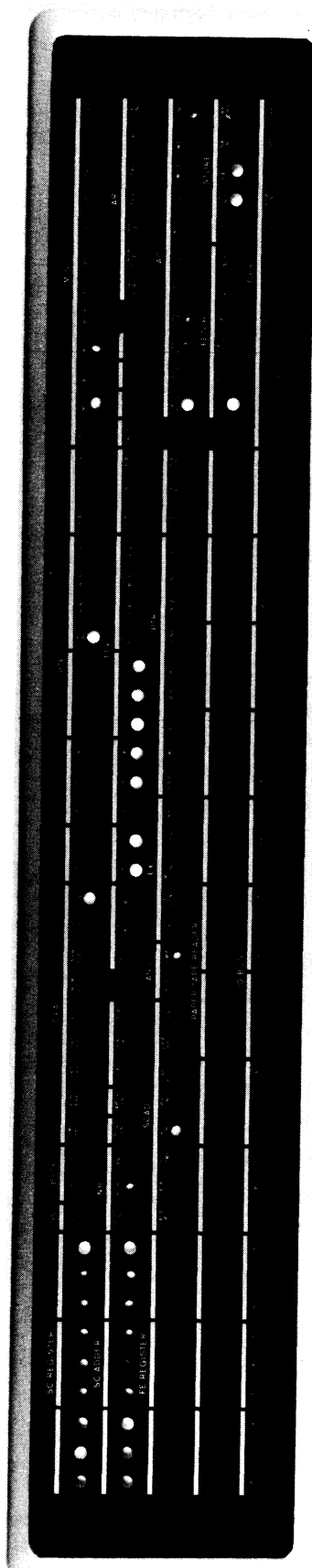
The remaining lights on the panels are for maintenance. If the operator must use them, he should consult the maintenance manual and the flow charts.

**Operating Keys**

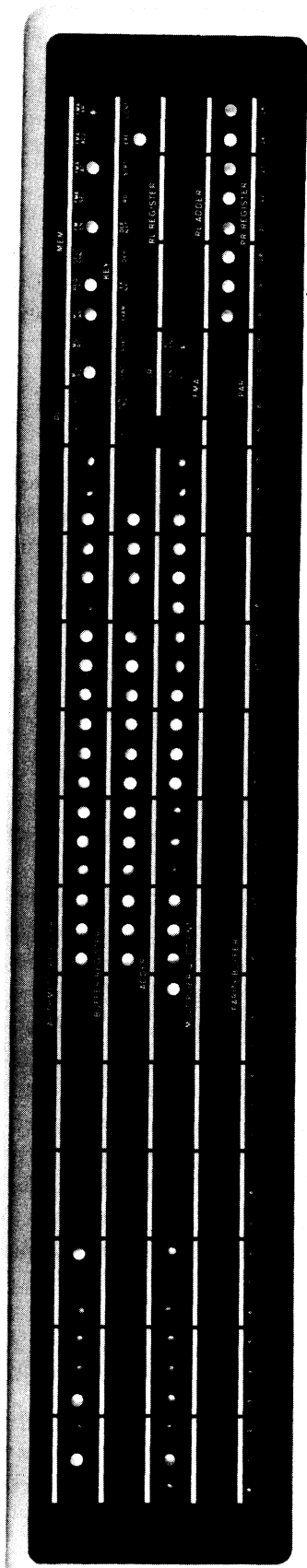
Each key except STOP turns on one of the key indicators at the upper right on the bay 2 panel. These are for flipflops that allow the key functions to be repeated automatically and also allow certain of them to be synchronized to the processor time chain so they can be performed while the processor is running.

**CAUTION**

*Never* press two keys simultaneously as the processor may attempt to perform both functions at once.



Indicator Panel, KA10 Arithmetic Processor, Bay 1



Indicator Panel, KA10 Arithmetic Processor, Bay 2

**READ IN**

Clear all IO devices and all processor flags including User; turn on the RIM light in the upper right on bay 1 and the KEY RDI light in the upper right on bay 2. Execute DATAI  $D,0$  where  $D$  is the device code specified by the readin device switches on the small panel at the left of the reader. Then execute a series of BLKI  $D,0$  instructions until the left half of location 0 reaches zero, at which time turn off RIM and KEY RDI. Stop only if the single instruction switch is on; otherwise turn on RUN and execute the last word read as an instruction. [*For information on the data format refer to §2.12.*]

Codes of readin devices are: PTR 104, DTC 320, DTC2 330, TMC 340, TMC2 350.

**START**

Load the contents of the address switches into PC, turn on RUN, and begin normal operation by executing the instruction at the location specified by PC.

This key function does not disturb the flags or the IO equipment; hence if USER MODE is lit a user program can be started.

**CONT (Continue)**

Turn on RUN (if it is off) and begin normal operation in the state indicated by the lights.

**STOP**

Turn off RUN so the processor stops before beginning the next instruction. Thus the processor usually stops at the end of the current instruction, which is displayed in the lights. However, if a key function that can be performed while RUN is on has been synchronized, the processor performs that function before stopping. In either case PC points to the next instruction.

If the processor does not reach the end of the instruction within 100  $\mu$ s, inhibit further effective address calculation – it is assumed the processor is caught in an indirect addressing loop. Pressing CONT when the processor is stopped in an address loop causes it to start the same instruction over.

**RESET**

Clear all IO devices and clear the processor including all flags. Turn on the triangular light beside MEMORY DATA (turn off the light beside PROGRAM DATA). If RUN is on duplicate the action of the STOP key before clearing.

If RUN is on, pressing this key has no effect.

The rightmost device switch is for bit 9 of the instruction and thus selects the least significant octal digit (which is always 0 or 4) in the device code.

**CAUTION**

Do not initiate any other key function while RIM is on. If read in must be stopped (*eg* because of a crumpled tape), press RESET (see below).

If RUN is on, pressing this key has no effect.

If STOP will not stop the processor, pressing this key will.

Note that an instruction executed from the console can alter the processor state just like any instruction in the program: it can change PC by jumping or skipping, alter the flags, or even cause a non-existent-memory stop.

**XCT**

Execute the contents of the data switches as an instruction without incrementing PC. If RUN is on, insert this instruction between two instructions in the program. Inhibit priority interrupts during its execution to guarantee that it will be finished.

If USER MODE is lit all user restrictions apply to an instruction executed from the console.

**NOTE**

The remaining key functions all reference memory. They use an absolute address and all of memory is available to them; in other words protection and relocation are not in effect even if USER MODE is lit. However they can set such flags as Address Break and Nonexistent Memory.

**EXAMINE THIS**

Display the contents of the address switches in the MA lights and the contents of the location specified by the address switches in the memory indicators. Turn on the triangular light beside MEMORY DATA (turn off the light beside PROGRAM DATA). If RUN is on, insert this function between two instructions in the program.

If RUN is on, pressing this key has no effect.

**EXAMINE NEXT**

Add 1 to the address displayed in the MA lights and display the contents of the location specified by the incremented address in the memory indicators. Turn on the triangular light beside MEMORY DATA (turn off the light beside PROGRAM DATA).

**DEPOSIT**

Deposit the contents of the data switches in the location specified by the address switches. Display the address in the MA lights and the word deposited in the memory indicators. Turn on the triangular light beside MEMORY DATA (turn off the light beside PROGRAM DATA). If RUN is on, insert this function between two instructions in the program.

If RUN is on, pressing this key has no effect.

**DEPOSIT NEXT**

Add 1 to the address displayed in the MA lights and deposit the contents of the data switches in the location specified by the incremented address. Display the word deposited in the memory indicators. Turn on the triangular light beside MEMORY DATA (turn off the light beside PROGRAM DATA).

## Operating Switches

Whenever the processor references memory at the location specified by the address switches (relocated if USER MODE is on), the contents of that location are displayed in the memory indicators (unless the light beside PROGRAM DATA is on). The group of five switches at the left of the keys allows the operator to make the processor halt or request an interrupt when reference is made to the specified location *in core memory* for a particular purpose (no action is produced by fast memory reference). The purpose is selected by the three address condition switches. INST FETCH selects the condition that access is for retrieval of an instruction (including an instruction executed by an XCT or contained in an interrupt location or a trap for an unimplemented operation) or an address word in an effective address calculation. DATA FETCH selects access for retrieval of an operand other than in an XCT (read-only or read-modify-write). WRITE selects access for writing only. Whenever reference to the specified location satisfies any selected address condition, the processor performs the action selected by the other two switches. ADR STOP halts the processor with MEMORY STOP on (PC points to the instruction that was being executed, or if the MC WR light on bay 2 is on, PC may point to the one following it). ADR BREAK turns on the CPA ADR BRK light (Address Break flag, CONI APR, bit 21) on bay 1, requesting an interrupt on the processor channel.

The description of each switch relates the action it produces while it is on.

### SING INST

Whenever the processor is placed in operation, clear RUN so that it stops at the end of the first instruction. Hence the operator can step through a program one instruction at a time, by pressing START for the first one and CONT for subsequent ones. Each time the processor stops, the lights display the same information as when STOP is pressed.

CLK FLAG (Clock flag) on bay 1 is held off to prevent clock interrupts while SING INST is on. Otherwise interrupts would occur at a faster rate than the instructions.

### SING CYCLE

Whenever the processor is placed in operation, stop it with MEMORY STOP on at the end of the first *core memory* reference. Hence the operator can step through a program one memory reference at a time, by pressing START for the first one and CONT for subsequent ones. To determine what information is displayed in the lights, consult the flow charts.

### PAR STOP

Stop with MEMORY STOP on at the end of any memory reference in which even parity is detected in a word read. A parity stop is indicated by the following: CPA PAR ERR (Parity Error flag) on bay 1 is on; and among the

AC and index register references can be included by turning off the FM ENB switch (see below).

If the interrupt for an address break is started before the completion of the instruction that caused it, that instruction will be restarted upon the return from the interrupt routine unless provision is made by the program to do otherwise. In such a case, the address break will recur, producing a loop between the processor interrupt and the interrupted program. The operator can free the processor by momentarily releasing the break switch.

SING INST will not stop the processor if a hangup prevents it from getting to the end of an instruction. Use STOP or RESET.

To stop at AC and index register references, turn off the FM ENB switch (see below).

If IGN is on (it displays a signal from the memory), parity errors are not detected and no stop can occur.

PAR lights in the bottom row on bay 2, IGN (ignore parity) and ODD are off, STOP is on, and BIT displays the parity bit for the word in the parity buffer at the left.

#### NXM STOP

Stop with MEMORY STOP on if a memory reference is attempted but the memory does not respond within 100  $\mu$ s. This type of stop is indicated by CPA NXM FLAG (Nonexistent Memory flag) on bay 1 being on.

#### REPT

The key function is repeated once after REPT is turned off, but this is noticeable only with very long repeat delays.

If any key (except STOP) is pressed, then every time the key function is finished, wait a period of time determined by the setting of the speed control and repeat the given key function. If CONT is pressed and no switch is on that would stop the program (eg SING INST, SING CYCLE), then continue following the repeat delay whenever a HALT instruction is executed. Continue to repeat the key function until RESET is pressed or REPT is turned off.

The end of a key function is equivalent to completion of all processor operations associated with the function only for read in, examine, examine next, deposit, and deposit next. In other cases the processor continues in operation. Eg the execute function is finished once the instruction to be executed is set up internally, but the processor then executes that instruction. Hence when using speed range 6, the operator must be careful not to allow the key function to restart before the processor is really finished.

The speed control includes a six-position switch that selects the delay range and a potentiometer for fine adjustment within the range. Delay ranges are as follows.

<i>Position</i>	<i>Range</i>
1	270 ms to 5.4 seconds
2	38 ms to 780 ms
3	3.9 ms to 78 ms
4	390 $\mu$ s to 7.8 ms
5	27 $\mu$ s to 540 $\mu$ s
6	2.2 $\mu$ s to 44 $\mu$ s

#### MI PROG DIS

Turn on the triangular light beside MEMORY DATA (turn off the light beside PROGRAM DATA) and inhibit the program from displaying any information in the memory indicators. The indicators will thus continually display the contents of locations selected from the console.

#### REPT BYP

If REPT is on, trigger the repeat delay at the *beginning* of the key function. Hence the function is repeated even if it does not run to completion.

#### FM ENB

This switch is left on for normal operation with a fast memory. Turning it *off* (lower part in) substitutes the first sixteen core locations for the fast memory. The switch is left off if there is no fast memory, and it can be used to allow stopping or breaking at fast memory references.



### SHIFT CNTR MAINT

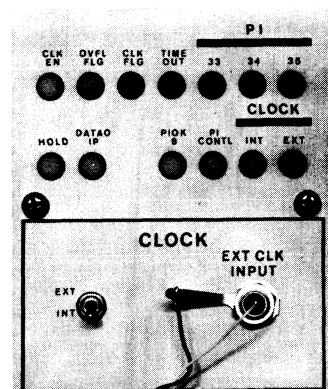
Stop before each step in any shift operation. Pressing CONT resumes the operation. Once a shift has been stopped, the processor will continue to stop at each step throughout the rest of the given shift operation even if the switch is turned off.

At the right end of panel 1J behind the bay doors are two toggle switches. FP TRP causes the floating point and byte manipulation instructions (codes 130-177) to trap to locations 60-61. MA TRP OFFSET moves the trap and interrupt locations to 140-161 for a second processor connected to the same memory.

### Real Time Clock DK10

The real time clock for the KI10 is usually installed under the console operator panel in bay 3 and has a small control panel mounted directly on the logic behind the cabinet door. In the lower part of the panel is a switch for selecting the internal source or an external input from the BNC connector at the right. The external input must be supplied through a 100 ohm coaxial cable and must have a frequency no greater than 400 kHz; its triggering voltage change must be from -3 volts to ground. If the input is a pulse train, the minimum pulse width is 100 ns. If the input is a sequence of level changes, it must have a minimum low level (-3 volts) duration of 400 ns before each positive-going change, a rise time of 60 ns maximum, and a high level duration of 40 ns minimum.

The leftmost light in the upper row at the top of the panel indicates when the clock is on (*ie* when the counter is enabled). The next two lights are the Count Overflow and Count Done flags. TIME OUT indicates when the numbers in the interval register and the clock counter are identical - this light goes out as soon as either changes state. The remaining lights in the upper row are the PI assignment. The two lights at the left in the lower row display signals that synchronize the DATAI and DATAO to the clock so that counting is postponed while the counter is being read and there is no sampling while the interval is being loaded. P1OK8 is a processor-generated signal which indicates that there is no interrupt being held and no channel waiting for an interrupt; the next light is the User Time flag. The final two lights indicate the origin of the clock source.



Clock Control Panel



## APPENDIX G

### MEMORY OPERATION

The DECsystem-10 memory comprises a number of individual memories of various types, whose size and timing are given in §1.3. A memory may be accessed by up to four different processors; in other words any memory may be connected to four memory buses and thus be part of the memory systems of four different processors. The more recent memories are designed for use with either a 22- or 18-bit address and may therefore be used with any PDP-10 processor. Earlier memories were designed specifically for use with the KA10; these memories are limited to an 18-bit address and can be used with other PDP-10 processors only by placing a KI10-M adapter in the bus. Although the programmer usually regards an address simply as the number of a location somewhere in memory, the memory system interprets the address in two parts: the left part is the number of a memory, and the right part is the number of a location in the memory selected by the left part. Every memory has switches for selecting its number, *ie* the number to which that particular memory will respond when it appears in the appropriate bits on the address bus. For a given address length, the number of bits used for the memory number depends upon the size of the individual memory – the larger the size the more bits are needed to specify the location within the memory and the fewer there are needed to select the memory itself. Besides address switches, every memory has a power switch, interleave and deselect switches, a restart or reset switch, and a single step switch.

To deselect a memory relative to a given processor means to remove that memory logically from the bus for that processor; in other words for the given processor that memory no longer exists. If a memory fails, it must be deselected if the system is to continue to run. Moreover, if the processor is a KA10 or the Monitor is a version earlier than 5.06, the deselected memory must be replaced so there is no gap in the physical address space. This may be done by resetting the switches on the highest numbered memory so it fills the gap left by the deselected one. When a system is installed, the system administrator (in consultation with Field Service) should work out a separate procedure to be followed in the event that any given memory fails. In other words there should be a set of procedures, and the set should contain as many procedures as there are memories in the system. A procedure might be as simple as filling a gap left by a deselected memory, but it may also involve the software or entail other complications. Whenever the organization of the system is changed in any way, that fact should be recorded, and the administrator should review, and if necessary change, the procedures to make sure they are appropriate to the new configuration.

Memory setup and operation differs among types, and these are discussed separately in the following pages. However there are general interleaving principles that are the same for all. Some memories can be interleaved only in pairs, whereas others can be interleaved in either pairs or quadruplets. In

The same considerations apply to use with direct access processors; *eg* using an 18-bit memory with a DF10C or a KI10-type bus on a DL10 requires an adapter.

With the MF10 and earlier memories, each unit actually has a separate set of address, interleave and deselect switches for each of the four processors to whose buses it may be connected. Hence a given memory may be number 2 for processor 0 but be number 7 for processor 1; by the same token it may be interleaved with some other memory relative to processor 1 but be deselected altogether from processors 2 and 3. A given memory should, however, have the same number with respect to all processors controlled by a single Monitor; *eg* if the Monitor running in central processor 0 sets up a direct-access processor to move data in or out of the memory connected to processor 0, those memory units used by both processors should have the same numbers.

The system administrator should be aware that even if the hardware and software are capable of dealing with non-contiguous memory, some of the programs being run may require that the memory be contiguous. This could be necessitated only by real time programs, but in general it is best to avoid having memory gaps unless they are known to be of no consequence.

Note that to avoid gaps in a system with different size memories requires arranging them so the smaller memories are at the top or are grouped so as to fill the spaces between the larger memories. Consider a system with two 32K memories and one 16K memory. The 32K memories must be numbered 0 and 1, and using the same numbering scheme, the 16K unit must be numbered 2.0 (really 4 in 16K terms). If there were two 16K units we could number them 2 and 3, with their space straddled by the 32K modules numbered 0 and 2 (*ie* the 16K modules are numbered 1 and 1½ in 32K terms).

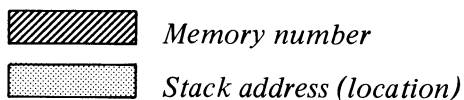
All transfers between bus and core are made through the buffer.

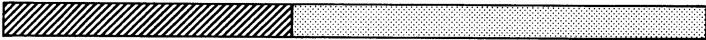
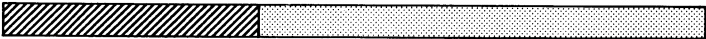
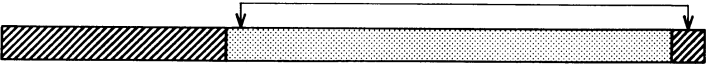
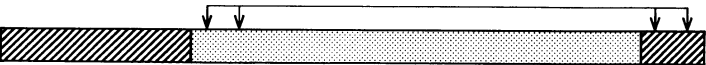

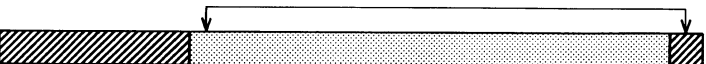
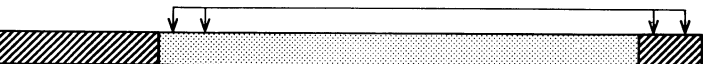

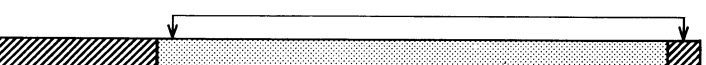
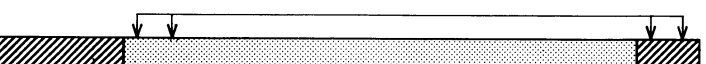


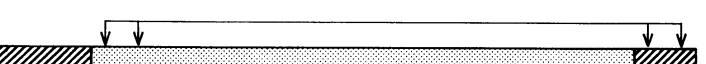

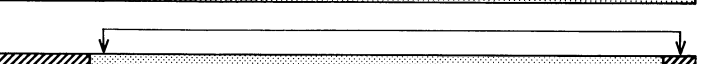
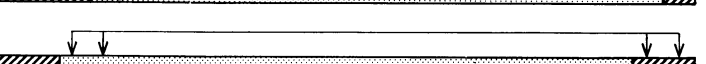
Some memories identify all processors currently making requests. There are also lights that reflect the internal state of the memory (for further information refer to the appropriate maintenance manual).

any event the memories in a group that is to be interleaved must all be the same size and must occupy a contiguous area of the overall address space. For a two-way interleave, the pair of memories must be numbered  $n$  and  $n+1$ , where  $n$  is even. The interleaving is accomplished by setting the interleave switches for the same processor at both memories to the INTL position. This action interchanges the least significant bits of the memory number and the location, so the least significant address switch at the memory is actually selecting a state for memory address bit 35. Hence all even addresses given by the processor in the interleaved set actually address the even-numbered memory, and all odd addresses address the odd-numbered memory. A four-way interleave must be done on a group of memories numbered  $n, n+1, n+2$  and  $n+3$ , where  $n$  is divisible by four. The interleave is accomplished by interchanging both the least significant bits of the memory number and location and the next more significant bits of those two quantities. The illustration on the next page shows the complete address structure for memories of all sizes, with and without interleaving.

Most or all of the switches on a memory are located on a switch panel mounted with the logic wiring inside the front door of the bay. The lights are always on an outside panel at the top of the bay. Every indicator panel has sets of memory address and memory buffer lights. These indicate the last location accessed and display the information read from or written into that location (except the buffer lights are off after the read part of a read-modify-write). The four ACTIVE lights identify the processor that currently has access, and the two LAST lights indicate which of processors 2 and 3 more recently had access. Priority among processors is their numerical order, with 0 first. However adhering strictly to this priority in a memory used by four processors might easily lead to the total exclusion of the lowest priority processor. To make this occurrence less likely, in a conflict between processors 2 and 3, access is granted to the one that has had it less recently. The panel also has a power indicator, lights that identify the type of request, and a parity bit. A memory in operation but idle is indicated by the AW light, meaning the memory is awaiting a request. STOP goes on only following completion of a cycle in single step mode. In a read access a memory completes its cycle without need of further communication from a processor. Following the read part of a write or read-modify-write however, the memory waits with SYNC on for a write restart, receipt of which is indicated by a light often labeled RS. Failure of the processor to supply the restart turns on the INC light (if present). Such an incomplete request may or may not hang up a specific type of memory, but it always results in leaving the addressed location clear. Since the system uses odd parity, a subsequent read at that location will result in a bad parity zero. Completion of a cycle is sometimes indicated by a CYC DONE light.

ADDRESS STRUCTURE



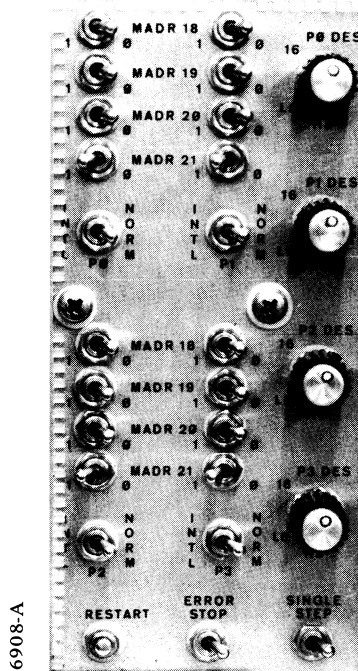
Memory Size	Address Bit Pairs Switched	Minimum Total Memory	Address Bit Configuration																		
			14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
8K	None																				
16K	None																				
16K	1	32K																			
16K	2	64K																			
32K	None																				
32K	1	64K																			
32K	2	128K																			
64K	None																				
64K	1	128K																			
64K	2	256K																			
128K	None																				
128K	1	256K																			
128K	2	512K																			
256K	None																				
256K	1	512K																			
256K	2	1024K																			

## MA10 CORE MEMORY

This unit has a capacity of 16K words, a cycle time of 1.00  $\mu$ s, and operates only with an 18-bit address, of which four bits select the unit. At the left inside the front door are two tall panels of margin check switches, all of which should be set left. The switch panel is in the lower right corner. Note that there is no power switch: power is controlled by the circuit breaker on the power control panel on the rear plenum door. The three switches at the bottom of the panel are for all processors. Ordinarily an incomplete cycle does not affect memory operation; the unit simply drops the unfinished cycle and awaits the next request. But running with ERROR STOP on causes the memory to cease operation with INC on when a processor fails to

supply a write restart. To free the memory so it may await further processor requests, push the RESTART button. Pressing RESTART while SINGLE STEP is on allows the memory to respond to just one processor request. Once this single cycle has been completed, STOP goes on and the memory will acknowledge no further requests, thus giving a nonexistent memory indication in any processor that makes one. Pressing RESTART again allows the memory to respond to one more request. It is possible for a power line transient to hang up the memory in a request; to free it, turn the power off and back on again by means of the circuit breaker at the back.

The remaining switches are in four sets for the individual processors. In each set the MADR switches allow selection of the memory number for address bits 18–21. Setting the fifth toggle to INTL interchanges address bits 21 and 35 for a two-way interleave. Setting the deselect switch to the 16 position deselects the whole unit from the corresponding processor.

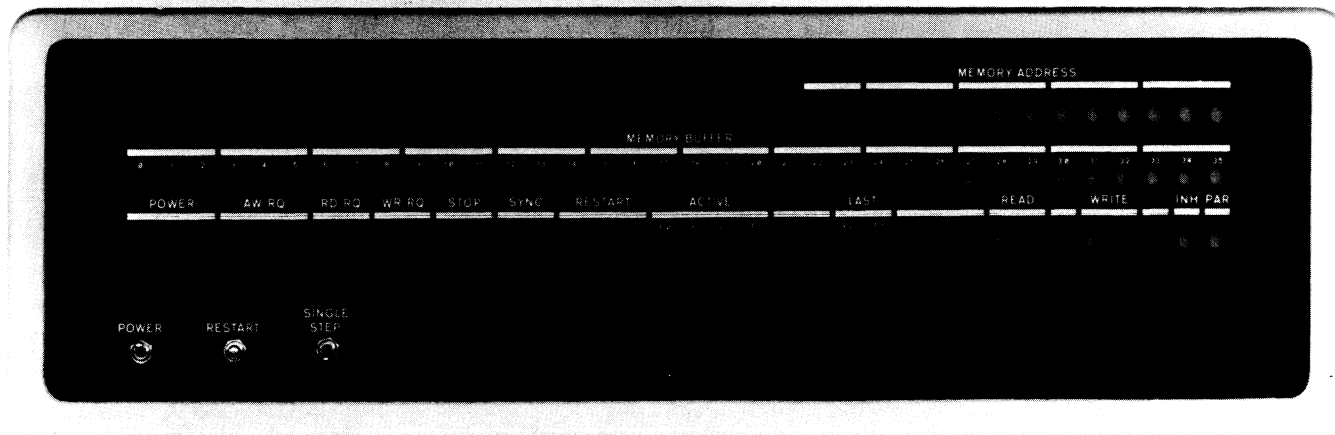


Above: Switch Panel  
Left: Indicator Panel

However the switch also has positions for deselecting the lower or higher half of the memory, resulting in an 8K memory with a 5-bit number. Eg setting the switch to L8 deselects the lower half and selects a 1 for address bit 22; the unit is then an 8K memory whose locations are addressable in the upper half of the original 16K address space. For 8K operation the interleave switch should always be set to NORM.

**MB10 CORE MEMORY**

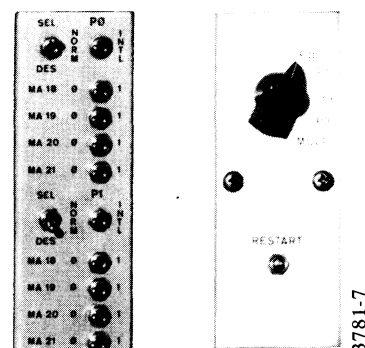
This unit has a capacity of 16K words, a cycle time of 1.65  $\mu$ s, and operates only with an 18-bit address, of which four bits select the unit.



Several switches for all processors are located on the indicator panel. Inside the front door are three switch panels. The one in the upper right corner has a rotary switch for selecting operation with any single processor or selecting the more usual multiprocessor operation. Also on this small panel is a button labeled RESTART, even though there is another button with the same label on the indicator panel. Should a processor fail to supply a write restart, the memory ceases operation with SYNC remaining on; to free the memory so it may await further processor requests, push both restart buttons at the same time. Pressing the indicator panel RESTART while SINGLE STEP is on allows the memory to respond to just one processor request. Once this single cycle has been completed, STOP goes on and the memory will acknowledge no further requests, thus giving a nonexistent memory indication in any processor that makes one. Pressing the top RESTART again allows the memory to respond to one more request. It is possible for a power line transient to hang up the memory in a request; to free it, turn the power off and back on again.

On the two small panels in the lower right corner are four sets of switches for the individual processors. Each set contains a deselect switch, an interleave switch, and four MA switches that allow selection of the memory number for address bits 18–21. Setting the interleave toggle to INTL interchanges address bits 21 and 35 for a two-way interleave.

Indicator Panel



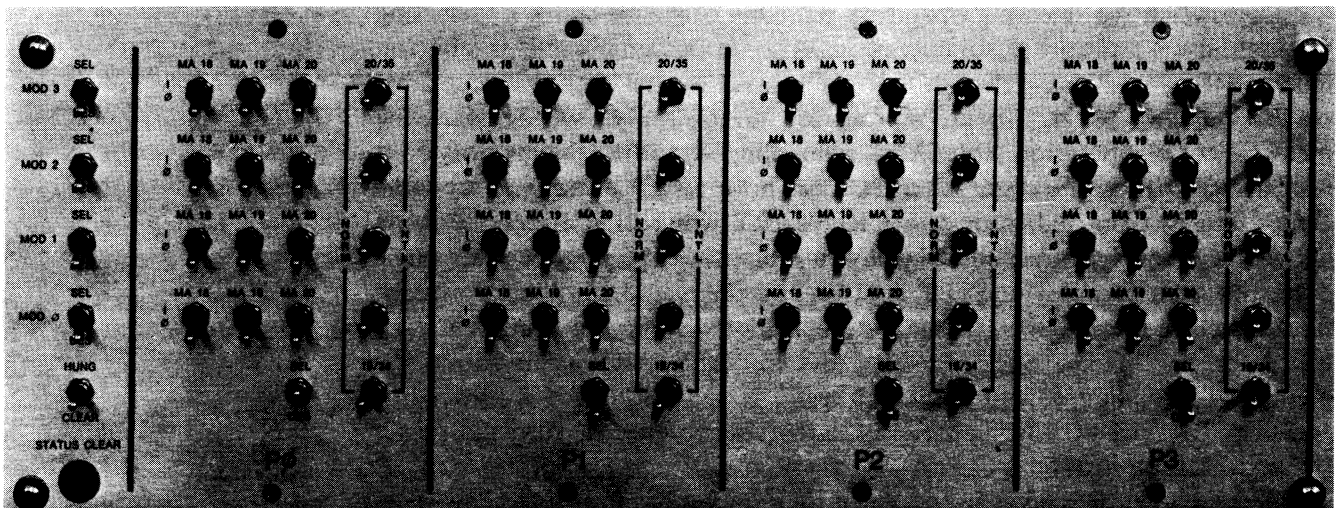
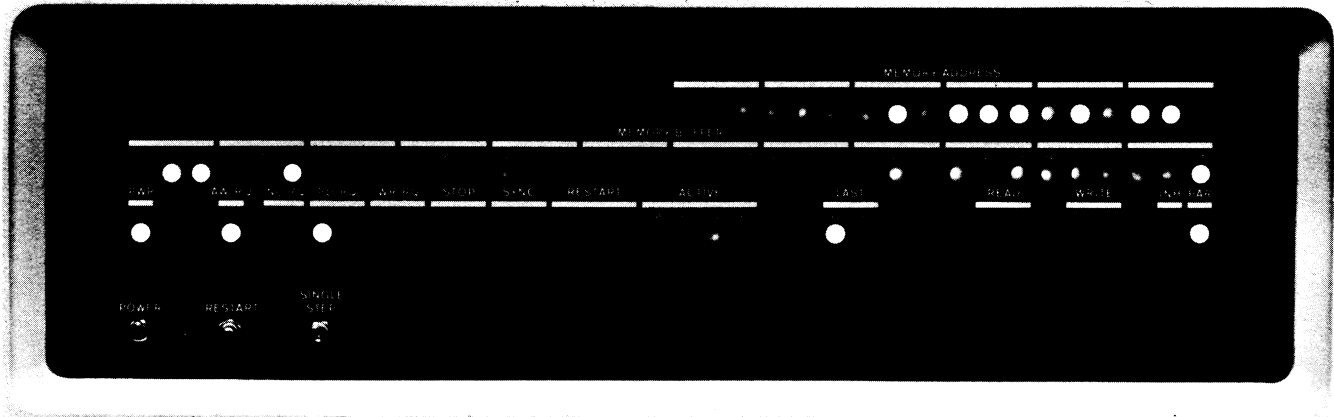
Switch Panels

The toggles at the left end of the logic rows are margin check switches, all of which should be set down. The toggle on the little panel between rows U and V should be set left (+10 FXD).

3781-7

## MD10 CORE MEMORY

This unit may contain up to four 32K memory modules, which are numbered and interleaved independently, but which share a common interface with the bus and therefore otherwise act as a single memory of 32, 64, 96 or 128K words. The cycle time is 1.8  $\mu$ s or less, and the unit operates only with an



Upper: Indicator Panel  
Lower: Switch Panel

18-bit address, of which four bits select the individual module. Interleaving among modules within a single MD10 is useless, because the whole unit is tied up whenever any module is performing a cycle even if it has already disconnected from the bus, as while a word is being written. Hence interleaving must be done between a module and one or three other 32K memories, which may themselves be modules in other MD10s. Note that there is no requirement of continuity of the address space within a given MD10 — if there were, interleaving would be impossible. Suppose a memory system consisted of two full-size MD10s with complete two-way interleaving. The modules in one unit could just as well be numbered 0, 3, 5 and 6, with the remaining numbers applied to the modules in the other unit.

Several switches for all processors are located on the indicator panel. Pressing RESTART while SINGLE STEP is on allows the memory to



respond to just one processor request. Once this single cycle has been completed, STOP goes on and the memory will acknowledge no further request, thus giving a nonexistent memory indication in any processor that makes one. Pressing RESTART again allows the memory to respond to one more request.

The remaining switches are on the big panel inside the front door. This panel has four large sections for the individual processors and a column at the left end for all processors. The upper four toggles in the column allow deselecting a single module from all processors. The button at the bottom is not used, and the remaining switch actually has three positions, including an unmarked center one. With this switch in the unmarked position, an incomplete cycle does not affect memory operation; the unit simply drops the unfinished cycle and awaits the next request. But running with the switch up in the HUNG position causes the memory to cease operation with INC RQ on when a processor fails to supply a write restart. To free the memory so it may await further processor requests, press the switch down to the momentary-contact CLEAR position.

Each of the large sections of the panel contains four sets of MA switches for independently selecting the numbers of the individual modules for address bits 18-21. (Note that each of the upper four rows of switches extending across the entire panel affects the single module specified by the label at the left end.) Each deselect switch allows the operator to disconnect the entire unit from the specified processor. The right column of each section contains five interleave switches, of which the upper four are for two-way interleaving of individual modules and the bottom one is for four-way interleaving of all modules together. Setting one of the upper four switches right interchanges address bits 20 and 35 only for the given module with respect to the specified processor. Setting the bottom toggle to the right interchanges address bits 19 and 34 only for the specified processor but for all modules in the unit.

Hence for a given processor some modules can be used normally while others are interleaved on a two-way basis with other memories outside the unit. But if one module enters into a four-way interleave with respect to a given processor, all modules must. This means that when a unit is used in a four-way interleave for a given processor, among the switches for that processor the 19/34 switch and all four 19/35 switches must be set to INTL. If four-way interleaving is used among MD10s, there must be four of them and each must contain the same number of modules.

Above the switch panel is a narrow panel with three margin check switches, all of which should be in the center NOM position. Above that is a power supply panel whose switch is not operative. The margin toggles at the bottom left should all be set down.

Note that one can deselect all modules from one processor, or one module from all processors, but not a single module from a single processor.

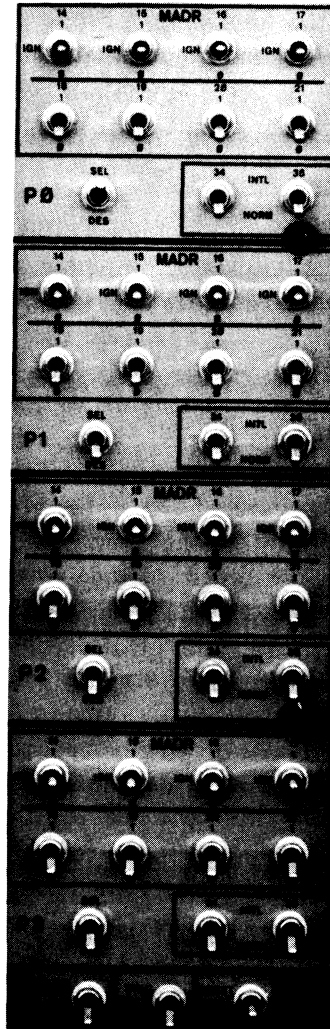
The restriction on number of units can be bypassed through worthless interleaving among modules within a single unit. This produces somewhat lopsided interleaving — *eg* one could have a three-way interleave of three MD10s with 32K, 32K and 64K.

## ME10 CORE MEMORY

This unit has a capacity of 16K words, a cycle time of 1.00  $\mu$ s, and operates with either a 22- or 18-bit address, of which eight or four bits select the unit. At the left inside the front door is a tall panel of margin check switches, all of which should be set left. The switch panel is at the lower right. Note that there is no power switch: power is controlled by the circuit breaker on the power control panel on the rear plenum door. The three switches at the bottom of the panel are for all processors. Ordinarily an incomplete cycle

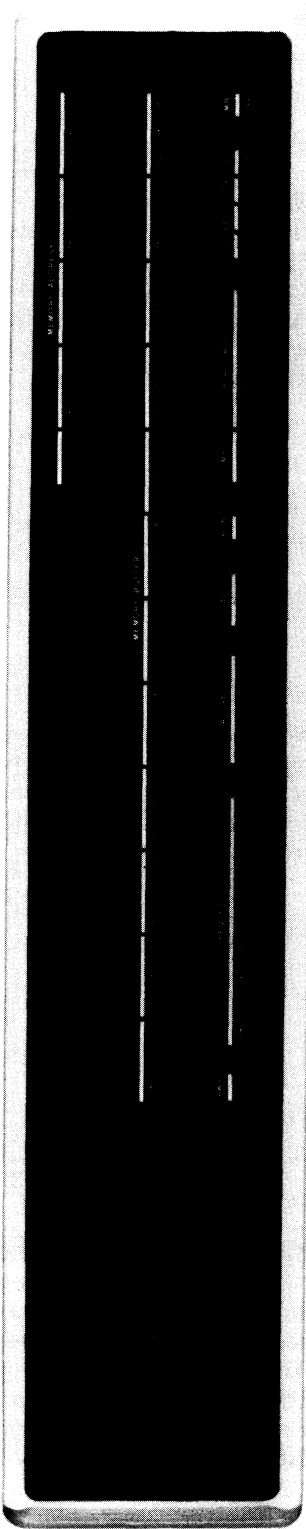
does not affect memory operation; the unit simply drops the unfinished cycle and awaits the next request. But running with ERROR STOP on causes the memory to cease operation with INC on when a processor fails to supply a write restart. To free the memory so it may await further processor requests, push up the RESET switch. Pressing RESET up while SING STEP is on allows the memory to respond to just one processor request. Once this single cycle has been completed, STOP goes on and the memory will acknowledge no further requests, thus giving a nonexistent memory indication in any processor that makes one. Pressing RESET again allows the memory to respond to one more request.

The rest of the panel is in four sections for the individual processors. Each section has two rows of MADR switches for selecting the memory number. If the bus supplies a 22-bit address, all eight MADR switches are used to select the memory number for address bits 14-21. For an 18-bit address the upper row must all be set to the center IGN position, and the lower row is used to select the memory number for address bits 18-21. Completing each panel section are a deselect switch and a pair of interleave switches. Setting the 35 toggle to INTL interchanges address bits 21 and 35 for a two-way interleave. Setting the 34 toggle up interchanges bits 20 and 34 for a four-way interleave.



5041-9

Above: Switch Panel  
Left: Indicator Panel

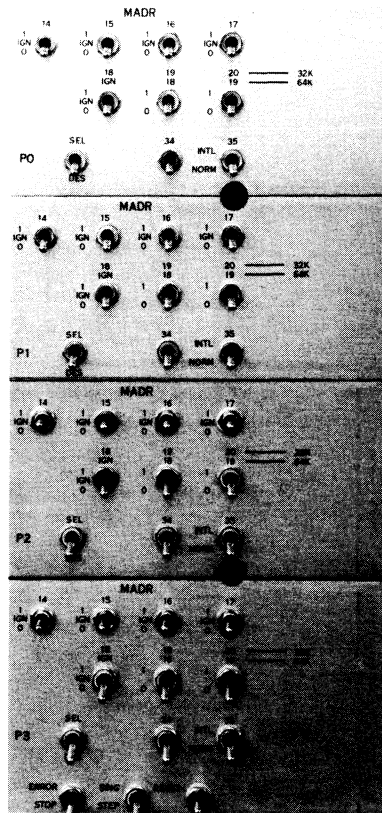


## MF10 CORE MEMORY

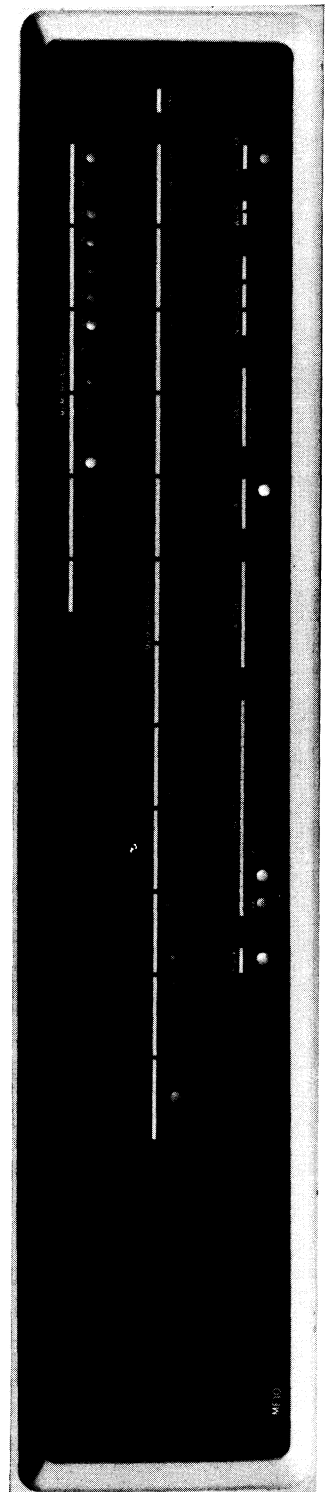
This unit has a capacity of either 32K or 64K words, a cycle time of 1.00  $\mu$ s, and operates with either a 22- or 18-bit address. The number of bits that select the unit depends on both the address length and the unit capacity. All switches are on panels in the lower half of the unit inside the front door. The tall panel at the left has margin check switches, all of which should be set left. At the upper right is a small maintenance panel, of interest in that it contains a size switch, which is set by Field Service to make the unit operate in a manner appropriate to the installed capacity, and whose position therefore indicates that capacity. Note that there is no power switch: power is controlled by the circuit breaker on the power control panel on the rear plenum door.

The four-part switch panel is at the lower right. The three switches at the bottom are for all processors. Ordinarily an incomplete cycle does not affect memory operation; the unit simply drops the unfinished cycle and awaits the next request. But running with ERROR STOP on causes the memory to cease operation with INC RQ on when a processor fails to supply a write restart. To free the memory so it may await further processor requests, push up the RESET switch. Pressing RESET up while SING STEP is on allows the memory to complete only one cycle in response to a single processor request. In the absence of a second request in the interim, pressing RESET again allows another request-cycle combination. However, if following completion of a cycle, a processor makes a request before RESET is pressed, the memory will return an acknowledgement, thus avoiding a nonexistent memory indication. Of course the memory will hang in the second cycle, either waiting to write or with the processor awaiting a read restart. Pressing RESET in this circumstance causes the memory to complete its cycle leaving it free to acknowledge yet another request.

The remaining switches are in four sets for the individual processors. Each set has two rows of MADR switches for selecting the memory number. If the bus supplies a 22-bit address, the upper row is used to select the left four bits (address bits 14-17) in the 6- or 7-bit memory number. For an 18-bit address the upper switches must all be set to the IGN position. The configuration of the switches in the lower row depends only on memory size. For a 32K unit these switches correspond to address bits 18-20 and are used to select a 3-bit memory number or the right three bits in a 7-bit number.



Above: Switch Panel  
Right: Indicator Panel



For a 64K unit the left switch must be set to the center IGN position; the other two correspond to address bits 18 and 19 and are used to select a 2-bit memory number or the right two bits in a 6-bit number. Completing the switch complement for each processor are a deselect switch and a pair of interleave switches. Setting the 35 toggle to INTL interchanges either address bit 19 or 20 (depending on unit size) with bit 35 for a two-way interleave. Setting the 34 toggle up interchanges either address bit 18 or 19 with bit 34 for a four-way interleave.

## APPENDIX H

### OPERATION OF PERIPHERAL EQUIPMENT

This appendix describes the various switches and indicators associated with the peripheral equipment and outlines procedures for loading tapes, changing paper, and so forth. Although such procedures are given here, the best and easiest way to learn how to do any of these things is to have someone who knows show you how. The final section of the appendix describes various cleaning procedures that are generally performed by the operator and which must be performed periodically.



## H1 CONSOLE IN-OUT EQUIPMENT

The console terminal is generally on a stand by the console. The reader and punch are located in a drawer above the operator panel, but the face of the reader is available on the front of the drawer, and at its right are a slot for removing tape from the punch and a pair of switches for feeding tape through reader and punch. Indicators for all three devices are on the panel at the top of the console bay on the KI10 and at the top of bay 1 on the KA10.

Photographs of the indicator panels of the KI10 and KA10 appear on pages F1-5 and F2-4 respectively.

### Paper Tape Reader

On the KI10 the contents of the reader buffer can be displayed in the top row of lights on the console indicator panel by setting the IND SELECT switch on the maintenance panel to PTR. Busy, Done and the PI assignment are displayed in the PTR lights in the middle of the bottom row; the remaining flags and maintenance lights are near the left end of the second row.

On the KA10 the paper tape reader lights in the second row from the bottom on the bay 1 indicator panel display the contents of the buffer. The flags, PI assignment and maintenance signals are displayed in the PTR lights in the middle of the third row (EOT is the Tape flag).

Tapes for the reader must be uncoiled and opaque. To load the reader, place the fanfold tape stack vertically in the bin at the right, oriented so that the front end of the tape is nearer the read head and the feed holes are away from you. Lift the gate, take three or four folds of tape leader from the bin, and slip the tape into the reader from the front. Carefully line up the feed holes with the sprocket teeth to avoid damaging the tape, and close the gate. Make sure that the part of the tape in the left bin is placed to correspond to the folds, otherwise it will not stack properly. Briefly press the feed switch at the right, making sure not to go beyond the leader into the data area; this sets the Tape flag to signal the program that the tape is loaded. After the program has finished reading the tape, run out the remaining trailer by pressing the feed switch; do not simply remove the tape from the read head by hand – run it out in order to clear the Tape flag.

### Paper Tape Punch

Indicators for the punch are the PTP lights near the middle of the bottom two rows on the console indicator panel on the KI10, in the top row on the bay 1 indicator panel on the KA10. The numbered lights display the last line punched.

The punch is behind the reader in the console drawer. Fanfold tape is fed from a box at the rear of the drawer. After it is punched, the tape moves into a storage bin from which the operator may remove it through the slot on the front. Pushing the feed switch beside the slot clears the punch buffer and punches blank tape as long as it is held in. Busy being set prevents the switch from clearing the buffer, so pressing it cannot interfere with program punching.

To load tape, first empty the chad box behind the punch. Then tear off the top of a box of fanfold tape (the top has a single flap; the bottom of the

box has a small flap in the center as well as the flap that extends the full length of the box). Set the box in the frame at the back and thread the tape through the punch mechanism. The arrows on the tape should be underneath and should point in the direction of tape motion. If they are on top, turn the box around. If they point in the opposite direction, the box was opened at the wrong end; remove the box, seal up the bottom, open the top, and thread the tape correctly.

To facilitate loading, tear or cut the end of the tape diagonally. Thread the tape under the out-of-tape plate, over the rear guide plate, and through the punch die block; open the front guide plate (over the sprocket wheel), push the tape beyond the sprocket wheel, and close the front guide plate. Press the feed switch long enough to punch about a foot and a half of leader. Make sure the tape is feeding and folding properly in the storage bin.

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

### Console Terminal

Operating information for terminals that may be used at the console is given in Chapter 9. Appendix B does however include a complete table of the ASCII code from which the character sets of the various terminals are drawn. Besides the standard definitions of the control characters, the table gives additional information about their generation and effect in relation to the more commonly used console terminals.

KI10 indicators for the terminal interface are the TTI, TTO and TTY lights in the right half of the bottom two rows on the bay 3 indicator panel. The flags and PI assignment are in the bottom row. Among the TTO lights, ACTIVE indicates when the transmitter is in operation and the numbered lights display the contents of the buffer. The numbered TTI lights display the last character typed in from the keyboard (bit 8 is parity on most DECsystem-10 terminals). The KA10 indicators are the TTY lights in the second row of the bay 1 indicator panel. The ACT lights indicate activity in the transmitter and receiver. The numbered lights display the last character typed in from the keyboard. At the right are the flags and PI assignment (the Input and Output Done flags are labeled TTI FLAG and TTO FLAG).

Connections for the console terminal are at a vertical panel behind the door below the console table. In the upper half of the KI10 panel are two rotary switches that select the input and output speed, a toggle switch that selects the signal type, and a pair of sockets for the signal cable. If the terminal were a typical Model 35 Teletype, both rotary switches would be set to 110, the toggle would be set to CURRENT, and the signal cable would be plugged into the upper socket. For a terminal that uses EIA standard levels, the toggle would be set to EIA and the cable would be plugged into



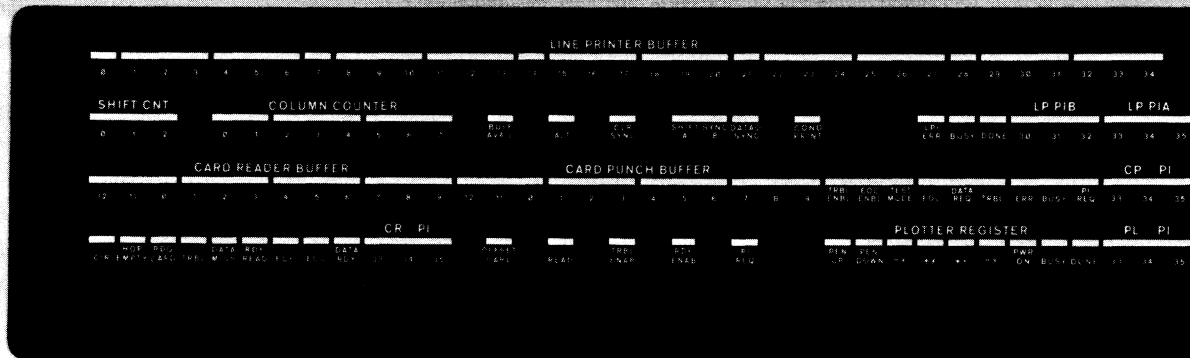
the lower socket. In the lower half of the panel are switched and unswitched convenience outlets for the terminal, scopes, and other equipment used at the console.

In the KA10 the connector panel is at the right and has a convenience outlet for the power cable. Mode and speed changes must be handled by Field Service.



## H2 HARDCOPY EQUIPMENT

DEC supplies the interfaces for line printer, plotter, card reader and card punch as a single unit, the BA10 hardcopy control. This control handles all four devices simultaneously, but is limited to one of each type. Indicators for the interfaces are on a panel at the top of the control cabinet. Inclusion



of a single printer, reader or punch in the system requires a BA10. But interface logic for a plotter is also available separately for mounting in the same cabinet with a DECTape control, in which case the indicators are in the bottom row on the DECTape control panel.

Indicator Panel,  
BA10 Hardcopy Control

### H2.1 LINE PRINTER LP10

Lights for the printer interface are in the top two rows on the hardcopy control indicator panel. The top row displays the contents of the character buffer; the 7-bit characters are shifted left for processing. The shift and column counters at the left end of the second row indicate the last character processed (0-4) and the last buffer position loaded. The group of lights at the right display the status conditions. Of the group in the center, BUFF AVAIL indicates the printer line buffer is ready for the next character; the remaining lights are for maintenance.

#### Models LP10F, H

These two models are essentially the same machine – they differ only in drum size, and hence in printing speed. Most controls and indicators are on a pair of panels, the operator panel at the left on top of the printer [illustrated on page H2-3], and the test panel mounted above the card cage inside the door on the left side. Controls for normal operation are those visible on the front part of the operator panel; controls on the back half of the panel are covered by a plate, which can be lifted from the front.

When the POWER and READY lights are on, the operator can place the unit on line by pressing the ON/OFF LINE switch. While the ON LINE

indicator is lit, pressing this same switch takes the unit off line. The unit also goes off line and unready when any of the error lights at the back of the panel goes on. The left light indicates simply that the drum gate is unlatched. Paper faults indicated by the center light are runaway paper motion, that the paper has run out or is torn, that the format tape loop is not installed properly, or the ribbon has reversed a specified number of times. The third light indicates an electrical malfunction or that the operator has inhibited printing by means of a switch located in the card cage. When the operator takes the unit off line or a ribbon change is indicated, the unit does not actually go off line until the buffer is empty (in other words until the printer finishes printing a line currently being loaded or printed). Paper running out is recognized only upon completion of the final line of the final form. READY goes out immediately if any other alarm condition occurs or power fails. When ON LINE is out or the cable to the interface is not connected, the Error flag is set and the printer cannot respond to the program. While the unit is off line, the operator can use the TOP OF FORM switch to position the paper or the format tape.

With extremely short forms (two inches or less), the printer may think the paper has run out upon completing the second to last form.

TOP OF FORM actually spaces the paper as indicated by column 1 on the format tape. If there is no format tape, the switch steps the paper one line.

To load paper follow this procedure.

1. Open the doors in the lower part of the printer front and set in a new box of fanfold paper. A box of paper with forms eleven inches or less in length will fit inside the printer with the doors closed. For longer forms, the printer must be run with the doors open.

2. Open the window at the top of the printer. Inside at the right end of the drum gate is a tall lever with a knob on the end. Push the lever to the left to unlatch the gate and pull it forward to swing the gate open to the left.

*CAUTION*

Do not proceed until the printer drum has stopped rotating.

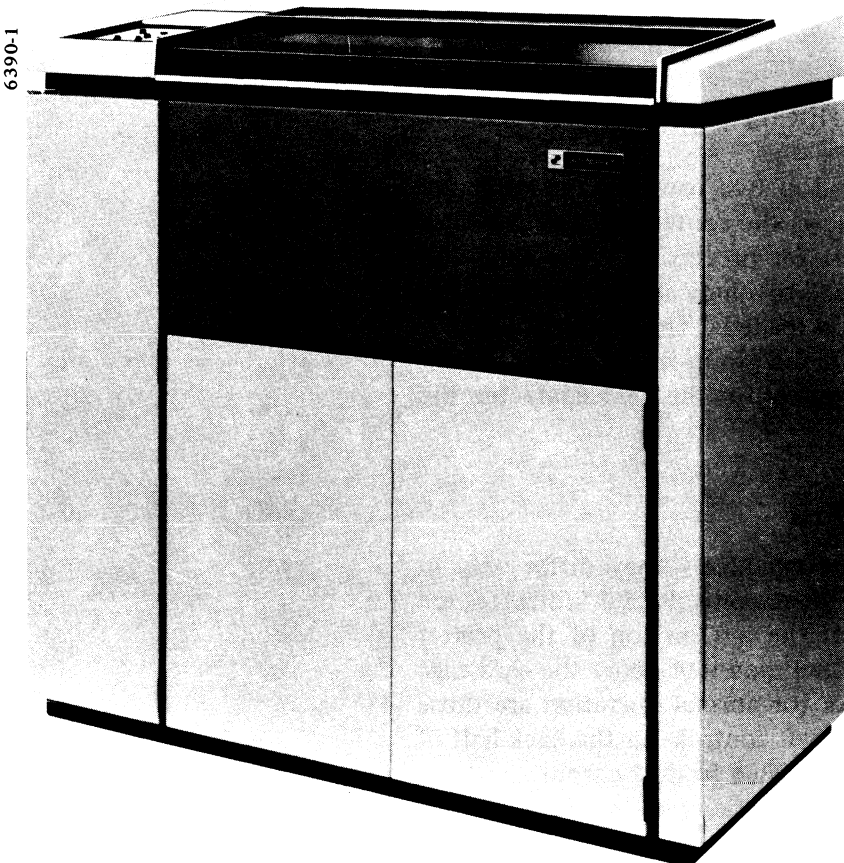
3. Press TOP OF FORM to position the format tape and run out the rest of the old paper.

4. At the top on the right end of the paper carriage is a lever for selecting the number of copies. Set this for the form being used. The total thickness of multicopy forms must be no more than .020 inch.

5. Line up the upper and lower left tractors vertically.

6. On the outsides of the right tractors are fine adjustment thumbwheels and tractor locks. On both upper and

Line Printer LP10F, H



6390-1

lower right tractors turn the thumbwheels clockwise as far as they will go, unlock the tractors, and move them to the extreme right.

7. Open the pressure plates on all four tractors. Set the left edge of the form in the upper left tractor, with the tractor pins through the feed holes, and close the pressure plate to hold the paper in place.

8. Slide the upper right tractor to the correct position, aligning the feed pins with the right feed holes. Lock the tractor to the guide shaft and close the pressure plate.

9. On the top at the right end of the carriage is a knob for controlling paper tension, and around the shaft of the knob is a locking disk. Unlock the knob and then turn it counterclockwise as far as it will go.

10. Install the paper in the lower left tractor and close the pressure plate. Slide the lower right tractor to position, install the paper, lock the tractor to the guide shaft, and close the pressure plate.

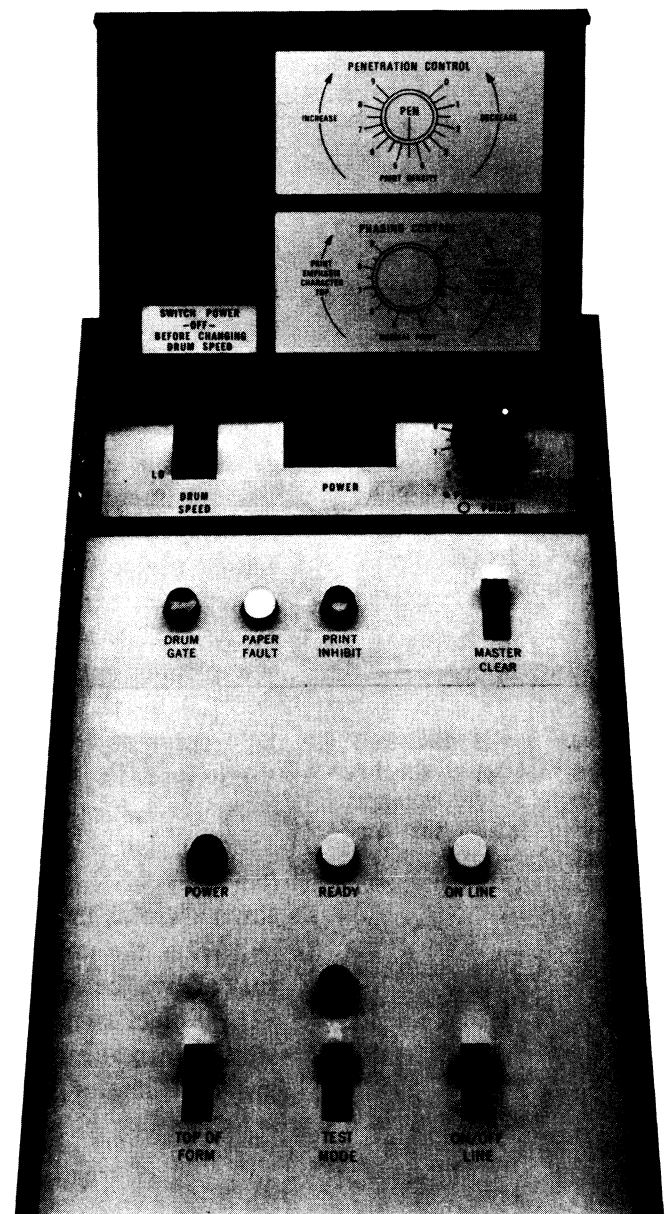
11. Turn the paper tension knob clockwise until there is a slight deformation in the tops of the feed holes. Lock the knob.

12. On both upper and lower right tractors, turn the fine adjustment thumbwheels counterclockwise until there is a slight deformation of the right edges of the feed holes.

13. At the extreme right end of the carriage is a RUN/ADJUST lever, and at the very top at the left end is a thumbwheel for controlling the paper drive mechanism. Move the lever to the ADJUST position, and turn the thumbwheel to move the paper until the top of form (the desired top line of the form) is aligned with the print hammers. For precise alignment of a special form, an alignment scale is contained in a sheath on the right side of the paper storage area. To align a special form, mount the scale across the hammer bank using the dowel pins located on castings at both sides of the bank. Horizontal paper position is controlled by the thumbwheel at the right end of the upper tractor guide shaft, and fine adjustment of the paper vertically is made by means of the knob just to the left of the tension knob. Using the thumbwheel, move the paper horizontally until the first print column is lined up with the first column on the scale; then using the knob, line up the top of form with the print line on the scale. Remove the alignment scale and put it back.

14. Move the RUN/ADJUST lever to RUN, close and latch the drum gate, close the printer window, and close the doors to the paper storage area.

The printer can accommodate paper 4–19 inches wide.



LP10F, H Operator Panel

For precise alignment of pre-printed forms, the vertical position knob can be manipulated while the printer is running.

15. Check the paper tension by pressing TOP OF FORM several times, making sure the paper does not pull loose from the tractors.

At the back of the operator panel (under the plate) is a POWER switch that operates in conjunction with a LOCAL/REMOTE switch on the test panel. In LOCAL, power is controlled entirely from the operator panel; in remote, the POWER switch is left on, and power is controlled from the processor. Pressing MASTER CLEAR initializes the printer logic. At the back left of the panel is a rocker switch for selecting the printing speed.

#### *CAUTION*

Change the drum speed only when the drum motor is not running. This means that either the power must be off or the drum gate unlatched.

The pair of concentric controls at the back right are for regulating penetration and phase. The penetration control allows varying the hammer force to compensate for the difference between new and old ribbons, to match a preprinted form, and to improve the quality of multiple copies. The phase control is used to equalize the top-to-bottom density of the characters; this adjustment is necessary only when the operator has changed the penetration or the drum speed.

Adjustments discussed above are generally made during a test run. Pressing the TEST MODE switch lights the unmarked indicator behind it and causes the unit to print lines of the character selected by the numbered switches on the test panel. If the toggle at the lower left on the test panel is set to NOT-FORM, the unit prints every line. With the toggle set to FORM, the printer spaces between lines according to the format tape column selected by the right three numbered toggle switches (numbers 0-7 select tape columns 1-8). Adjustments are a little easier if the NORMAL/SLOW toggle is set to SLOW: this reduces the printing speed by about 55%.

At the left end of the drum gate is a counter with switches by which the operator can set the number of allowable ribbon reversals. Once the ribbon has reversed the specified number of times, an alarm light on the counter turns on; this in turn lights the paper fault indicator on the operator panel and takes the unit off line. Also on the counter is an alarm override switch that allows the operator to put the printer back on line and continue to run.

Complete instructions for changing ribbons, preparing and installing a special format tape, changing drums, and switching to eight lines per inch are given in Chapter 3 of the Data Products manual for Line Printer Model 2470.

#### **Models LP10B, C, D, E**

At the left on the front of the printer are two round indicators and two columns of square buttons and indicators, some of which are not used. The round lights indicate whether the printer has power: green light for power on, red for off.

The buttons at the top of the columns operate the printer. Pushing START places the printer on line so it can respond to the program (the

In other words the same switches that select the low order digit of the character code also select the tape column.

button is lit while the unit is on line). Pushing STOP takes the unit off line; the operator can then use the TOP OF FORM button to position the paper (or the format tape). If the program has left anything in the buffer, it can be printed by pressing MANUAL PRINT. The maintenance button TEST can also be used while STOP is lit. START, STOP and TOP OF FORM are duplicated at the rear of the printer.

At the bottom of the columns are four alarm lights that indicate when the paper supply is low, the printer is out of paper or the paper is broken, the yoke is open, or there is a circuit malfunction (ALARM STATUS). When the operator presses STOP or there is a paper low alert, START does not go out until the buffer is empty (in other words until the printer finishes printing a line currently being loaded or printed). START goes out immediately if any other alarm condition occurs or power fails. When START is out or the cable to the interface is not connected, the Error flag is set and the printer cannot respond to the program.

The printer usually shuts down somewhere within the final form, but pressing START while PAPER LOW ALERT is lit allows the program to print a single line (after a noticeable delay). Hence the operator can allow the program to complete the form by pressing START for each line. However this should be done only if the program is set up to stop itself upon sending out the data for the last line of the present form. It is difficult for the operator to see exactly what line is being printed, and it is therefore up to the program to stop after the last line even if the operator should again press START. Otherwise the printer may be damaged by printing after the paper has run out.

To load the paper follow this procedure.

1. Press STOP. If START does not go out, the program probably left the last line in the buffer: press MANUAL PRINT. When printing is complete, the light will go out.
2. Press TOP OF FORM to position the format tape.
3. Open the printer cover. At the left and right front are two toggle switches; switch both of them to OPEN. The printer yoke will slide forward. Lift the guide plates over the two pairs of tractors and pull out any remaining paper.
4. Bring the new paper up through the open yoke throat and thread it on the upper tractors so the paper feed holes fit over the pins. Just outside the



Line Printer LP10B

lower tractors are knobs for moving the right and left pairs of tractors horizontally; use these to adjust the tractor positions to the paper width. Close the upper guide plates.

5. Grasp the paper below the yoke and gently pull it taut. Place the feed holes over the pins of the lower tractors and close the guide plates.

6. Establish the appropriate horizontal tension by adjusting the right tractors until there is a slight deformation of the right edges of the feed holes.

7. Beside the left yoke toggle switch is a knob for adjusting the vertical tension of the paper. The correct vertical tension is a slight deformation in the tops of the feed holes at the upper tractors.

8. Behind the yoke is a plastic scale – it should be leaning against the yoke. Lay this scale back against the paper. Close the lock lever on the lower right tractor so all four tractors move together horizontally. Using the vertical lines on the plastic scale for reference, move the tractors to set the left margin of the paper.

9. Open all four guide plates. Without moving the tractor pins, move the paper vertically until the top of form (the desired top line of the form) is approximately between the two black lines across the bottom of the scale. Thread the paper back on the tractor pins and close the guide plates. At the top left of the tractor carriage is a knob for fine vertical adjustment of the paper. Use this knob for closer alignment of the top of form between the lines on the scale.

10. Release the tractor lock, switch the toggles to CLOSE, close the cover, and press START. After the paper has been advanced enough to reach the rear of the printer, position it between the paper puller roller rings and over the static eliminator bar.

At the left end on top of the yoke is a penetration control, which allows varying the hammer force to compensate for the difference between new and old ribbons, to match a preprinted form, and to improve the quality of multiple copies.

### Model LP10A

This printer is smaller than the LP10B, but otherwise very similar. The lights are at the right, the single PAPER ALARM indicates the paper is either low or torn, and there are no buttons on the back. With the cover open the yoke is controlled by two unmarked plastic switches on either side at the top. Pressing them in at the end nearer the front opens the yoke. This printer has only one pair of tractors, but it has a pair of bars below the yoke. The paper must go over the stationary bar and under the movable one.

## H2.2 PLOTTER XY10

Lights for the plotter are the group at the right end in the bottom row on the hardcopy control indicator panel [*page H2-1*]. These display the status

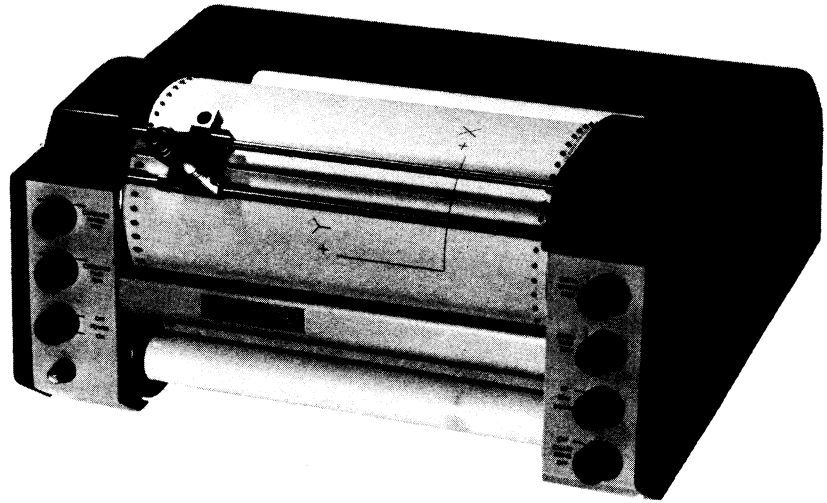
For precise alignment of pre-printed forms, the vertical position knob can be manipulated while the printer is running.



conditions and the plotting data supplied by the last DATAO.

On the drum plotter the supply roll is behind the drum. Bring the paper over the drum, down in front, and above and behind the pickup roll underneath the drum (use a piece of masking tape to attach the paper, or roll some onto the tube).

The controls are on the front. To put the plotter on line simply turn on the power and the chart drive. The remaining controls are for manual operation: raising and lowering the pen, moving the carriage and drum in either direction, rapidly or single step. The switch that selects the step size on a 600-series plotter is on the back. The bed plotter has similar controls.



Plotter XY10A

**H2.3 CARD READER CR10**

Lights for the reader interface are in the bottom two rows on the hardcopy control indicator panel [page H2-1]. The left section of the upper row displays the contents of the card column buffer; the lights are marked by card row. The left section of the bottom row displays bits 24-35 of the status conditions. (The second light from the left is labeled HOP EMPTY, but it goes on when the hopper is empty or the stacker is full.) Of the five lights in the center, the left one is the momentary offset signal, which is applicable only to the CR10A/B. READ is on when a read command has been given but the reader is not yet ready. The next two lights display bits 18 and 19 of the status conditions, and the last light is on while an interrupt is being requested whatever the cause.

**Models CR10D, E, F**

The CR10D reader is illustrated below. The other models are very similar: the F is smaller and slower, whereas the E is a floor model that is much larger and faster, and has an end-of-file button that the others lack. On all models the input hopper is on the right and the stacker is on the left. To load a deck, first fan the cards and jog them on the top of the reader. Pull the hopper follower back and put the deck into the hopper with the front of the cards toward the front of the reader and the 9 edge down so column 1 is read first. Add more cards as desired, but only until the hopper is loosely filled.

Approximate	hopper	and
stacker	capacities	are:
	CR10D	950
	CR10E	2200
	CR10F	550

*CAUTION*

Do not pack the hopper so full as to inhibit the riffle action.



Card Reader CR10D

Cards can be added to the hopper while the reader is running provided the hopper is a third to a half full so there is enough pressure on the front of the deck to maintain the riffle action. Cards can be removed from the stacker at any time simply by pulling the stacker barrier forward and lifting them out.

The reader is operated by the row of illuminated buttons and indicators at the left front. Power is controlled by the switch at the left end except on the CR10F, which has only an indicator on the front and a power switch on the panel at the rear. Pushing RESET places the reader on line so the

program can read cards. Pushing STOP turns off the reader, taking it off line.

The four alarm indicators in the middle of the row are as follows.

- |              |  |
|--------------|--|
| READ CHECK   | Either there is a failure in the read electronics, or more likely, the card just read has a torn leading or trailing edge or has punches outside of columns 1-80.  |
| PICK CHECK   | The reader has received a pick command but the card has failed to reach the read station. Inspect the cards in the hopper for mutilated edges, torn webs, staples or extreme warpage. If nothing appears wrong with the cards, check the picker face for buildup of ink glaze, and clean it with solvent if necessary [see §H7.3]. |
| STACK CHECK  | The last card read is not properly seated in the stacker. Make sure the card track is clear and check the stacker for a badly mutilated card.  |
| HOPPER CHECK | The hopper is empty or the stacker is full.  |

When one of these lights goes on, the RESET light goes out (the reader always finishes the current card before stopping). Do not attempt to reread a worn or damaged card that has caused a read, pick or stack failure — duplicate if first. After the trouble has been corrected, press RESET to allow the program to continue reading the deck. If the trouble persists, enter it into the system log and notify maintenance personnel.

At the right end of the row on the CR10E only is an END OF FILE button. Pressing this button when the reader is off line, as when the hopper is empty, sets the End of File flag.

On the back of the reader is a panel with a lamp test button and two toggle switches. For normal operation the MODE toggle is in REMOTE and the SHUTDOWN toggle is in AUTO. Switching MODE to LOCAL takes the reader off line so the operator can pick individual cards by pressing RESET. Switching SHUTDOWN to MANUAL keeps the riffle blower running continuously even when the hopper is empty.

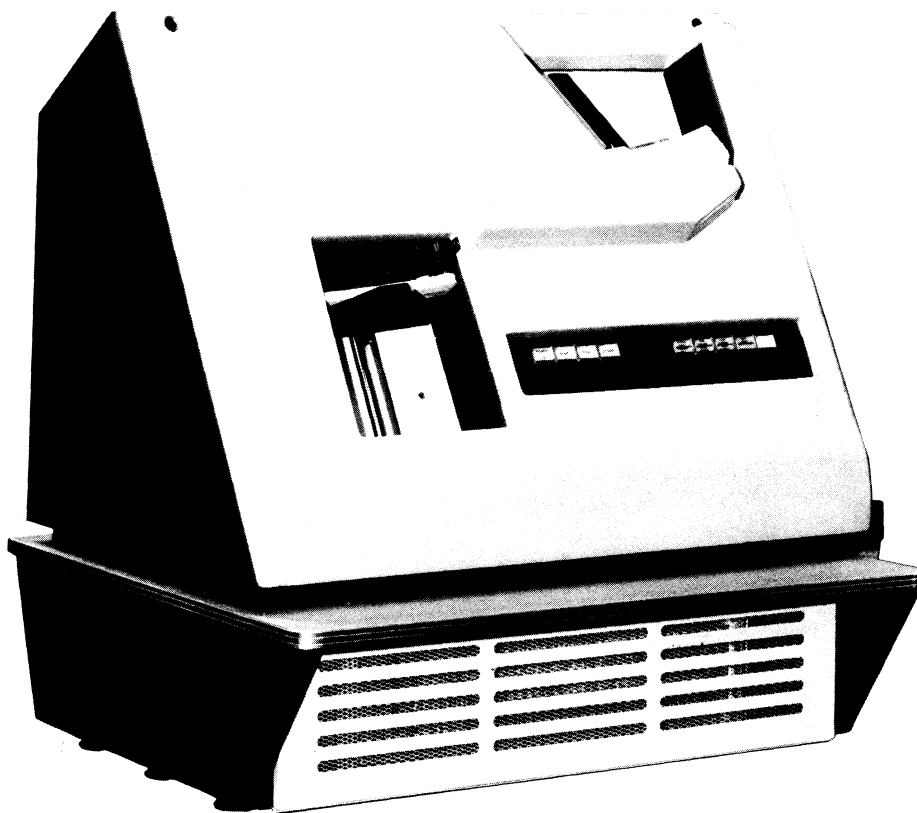
### Model CR10A/B

The CR10A/B reader has a hopper and stacker capacity of 1000 cards. To load a deck, first fan the cards and jog them on the reader shelf. Turn the deck over and put the first hundred cards (about an inch of the deck) into the hopper (upper right) with the 9 edge against the back so column 1 is read first. Place the rest of the deck on top of the first part. Cards can be added to the hopper while the reader is running, but always stop the reader before removing cards from the stacker.

The reader is operated by the buttons at the left. The alternate-action POWER switch lights green when power is on. Pushing START places the reader on line so the program can read cards. Pushing STOP turns off the reader, taking it off line.

The lights at the right indicate an empty hopper, a full stacker, a pick failure, a card motion error, and a photocell output that is too weak or too strong. When one of these lights goes on the STOP light also goes on (the reader always finishes the current card before stopping). Do not attempt to reread a worn or damaged card that has caused a pick failure or motion error — duplicate it first. If any trouble light remains on after the problem is corrected press the CLEAR button; this turns off both the lights and the corresponding status signals read by a CONI. Press START to allow the program to continue reading the deck. If the trouble persists, enter it in the system log and notify maintenance personnel.

Pressing the END OF FILE button (at the right) when the reader is off line, as when the hopper is empty, sets the End of File flag. When the TEST MODE light is on, the reader processes cards off line (the test switch is behind the panel under the shelf).



Card Reader CR10A/B

## H2.4 CARD PUNCH CP10

Lights for the punch interface are in the second row from the bottom on the hardcopy control indicator panel [page H2-1]. The middle section of the row displays the contents of the card column buffer; the lights are marked by card row. Among the lights in the right section, PI REQ is on while an interrupt is being requested whatever the cause. The remaining lights display some of the status conditions read by a CONI.

A good way to reduce card problems in the punch is to first run the blank cards through the reader in local. Then any cards that fail in the reader can be removed from the deck before it is loaded into the punch.

The CP10 card punch has a hopper and stacker capacity of 1000 cards. To load the hopper, first fan the cards and jog them on the punch shelf. Turn the deck over and put the first hundred cards (about an inch of the deck) into the hopper (upper right) with the 9 edge against the back so column 1 is punched first. Hold the right end higher so the leading edge of the bottom card rests against the picker throat, and drop the cards in place. Put the rest of the deck on top of the first part. Cards can be added to the hopper while the punch is running, but always stop the punch before removing cards from the stacker. To remove cards, push down the elevator and lift the stack out.

The punch is operated by the buttons in the upper part of the panel at the right. The alternate-action POWER switch lights green when power is on. Pushing START places the punch on line so the program can punch cards.

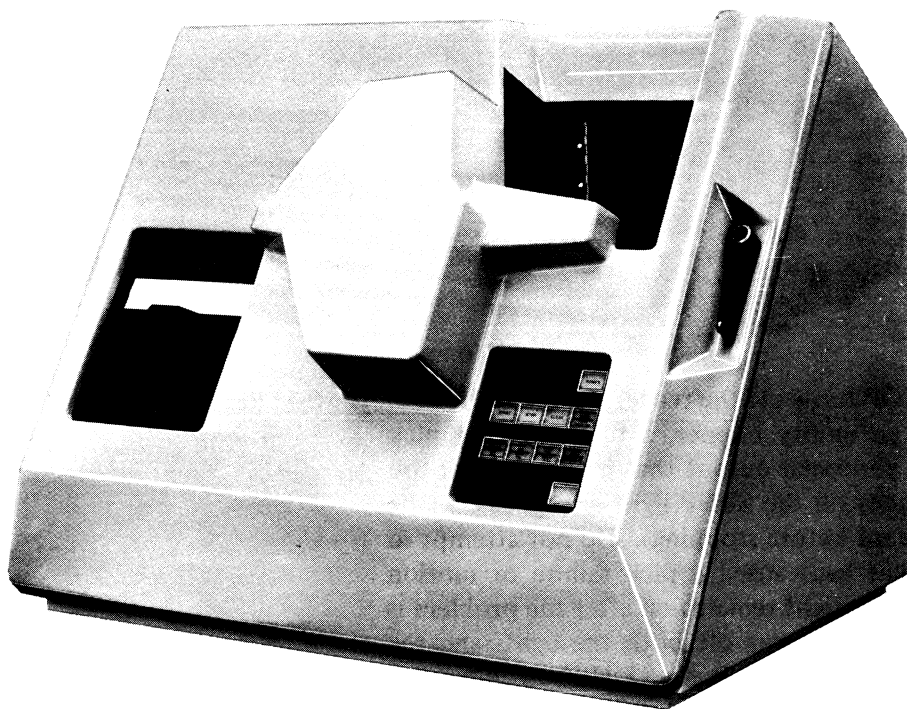
The OPERATE indicator at the lower right lights green when the punch motor is up to speed. Pushing STOP takes the reader off line but does not stop the motor; the motor is turned off only by pressing CLEAR.

The lights in the bottom row indicate an empty hopper or full stacker, a full chip box, a pick failure, an eject failure, and a stack failure. When one of these lights other than CHIP BOX goes on, the STOP light also goes on (the punch always finishes the current card before stopping). If any trouble light remains on after the problem is corrected, press the CLEAR button; this turns off both the lights and the corresponding

status signals read by a CONI. Pressing CLEAR also ejects a card if one is in the head assembly, and the button glows red when clear action is required (eg when a card has gotten stuck). For a pick failure, empty the hopper, throw out the bottom card, and reload. Press START to allow the program to continue punching. If the trouble persists enter it in the system log and notify maintenance personnel.

A full chip box does not stop the punch, but once it has been stopped by some other condition (such as an empty hopper), pressing START will not place the unit on line until the box has been emptied.

At the right is a light for the Card in Punch flag. The ERROR light displays the signal that sets the Error flag; it goes off when CLEAR is pressed. The CHECKOFF light is not used. When the TEST light is on, the device punches cards off line in a test pattern (the test switch is behind the panel under the shelf).



Card Punch CP10  
(without pedestal)

### H3 DATA INTERFACES

*To be added – in the meantime refer to Chapter 5.*



## H4 MAGNETIC TAPE

The maxim for tape operations is cleanliness is next to godliness. Nothing can ruin a tape run more easily than dust, ash or a piece of dirt. The transport should be kept clean and well adjusted, and in particular the tape path should be cleaned frequently. Maintenance personnel are generally responsible for most tape servicing procedures, but often the operator should expect to do daily cleaning. Instructions are given in §H7.1.

◆ *Never smoke* in a computer room or tape storage area. Tobacco smoke and ash are especially damaging to magnetic tape.

◆ Paper dust is very damaging to tape. Never put a tape transport near a line printer or other device that uses paper. Never place any notes or labels inside tape containers.

◆ Always store tapes inside their containers in an area with temperature between 40 and 90°F and relative humidity between 20 and 80%. Keep empty containers shut so that no dust or dirt can get inside. Always hold a standard tape reel with the hand through the hub hole – squeezing the reel flanges damages the tape edges. Never touch the tape with your fingers inside the markers; never touch any part of the tape to clothes, or allow the end of the tape to drag on the floor, or get cleaning fluids or solvents on the tape.

◆ Do not use cellophane mending tape. To mark reels, use only labels with adhesive that leaves no residue and does not shed.

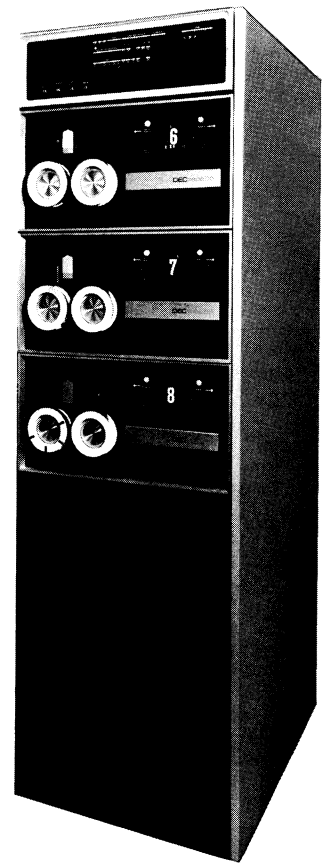
◆ Inspect all tapes, reels and containers frequently. Cut off the end of a tape that has been damaged. Replace takeup reels that are old or damaged.

◆ Do not eat in a computer or tape storage area, and keep such areas clean at all times.

### H4.1 DECTAPE TD10

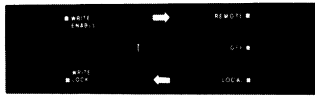
Several DEctape transports are often mounted in the same cabinet with the control as illustrated here. There are two transport models, but operationally both are essentially equivalent. The TU56 contains two drives that utilize solid state switching circuits; the TU55 has a single drive that utilizes relay switching. At the control panel of any drive, the operator selects the number of the drive by dialing the thumbwheel in the center of the panel; the selector also has an OFF LINE position in which the drive cannot be selected from the control. At the right end of the panel is the mode switch (just right of the thumbwheel on the TU55). Turning this switch OFF idles the drive, turning off power to the drive motors and releasing the motor hubs and brakes so the operator can handle the tape reels (the switch does not turn off logic power in the drive; this is supplied by the control). With the mode switch in OFF, the drive can be selected by giving the number dialed into the thumbwheel, but only for purposes of checking status – the drive cannot be placed in operation. Turning the switch to REMOTE places the drive in the hands of the DEctape control; the light at the right of the switch (above on the TU55) goes on when the transport is selected. Turning the mode switch to LOCAL allows the operator to run the drive from the panel. The TU56

For further information refer to “Cutting expenses by taking care of your tape”, by Robert G. Devitt, *Computer Decisions*, Vol. 2 No. 10, October 1970, pp. 42–45.



DEctape System

6114-1



TU56 Control Panel

4689-8



TU55 Control Panel

Never use adhesive to attach the tape to the reel.

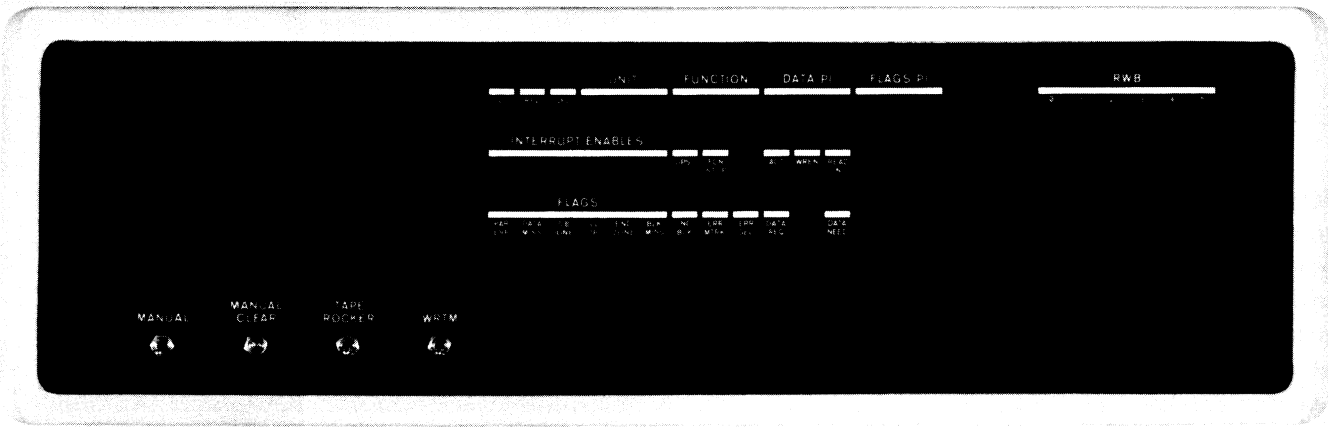
has a single rocker switch with large arrows at top and bottom and an unmarked, spring-loaded center off position; pushing in the top or bottom of this switch causes the tape to run in the direction indicated by the arrow so long as the switch is held in. The TU55 has two switches under the large arrows at either end of the panel; pushing in the upper part of either of these switches causes the tape to run in the indicated direction. The WRITE ENABLE(D)/WRITE LOCK switch controls writing on the tape by the control. Pressing in the upper part of the switch lights the indicator at its left (above on the TU55) and allows the control to write information on the tape when the program gives a write function. Pushing in the lower part of the switch prevents writing and sets Write Lock, CONI DTS, bit 24.

To load tape, turn the mode switch to OFF and simply press a full reel of tape onto the left hub and an empty reel on the right hub. Unreel about a foot of tape from the supply reel, place the tape over the guides and the tape head, and wind one or two revolutions of tape onto the takeup reel. The forward direction is defined as tape motion from the left reel to the right reel.

To rewind a tape place the transport in local and make sure the thumb-wheel is set to a number that is not selected by the control. Then, while holding in the reverse switch, put the transport in remote. Release the reverse switch and the entire tape will rewind onto the left reel.

The switches and indicators for the control are on a panel at the top of the control cabinet. The switches in the lower left are turned on by pushing them up. Turning on WRTM allows the program to execute a Write Mark Track function in which data supplied by the program is written in the mark track and the timing track is also recorded. Turning on MANUAL takes the

P3699



TD10 Control Panel

control off line and enables the remaining two switches. MANUAL CLEAR duplicates the IO reset. TAPE ROCKER simulates readin mode within the control but without any data connection to the processor; the control selects unit 8 and executes a Read Data function back and forth the entire length of the tape, turning it around every time the end zone is encountered.

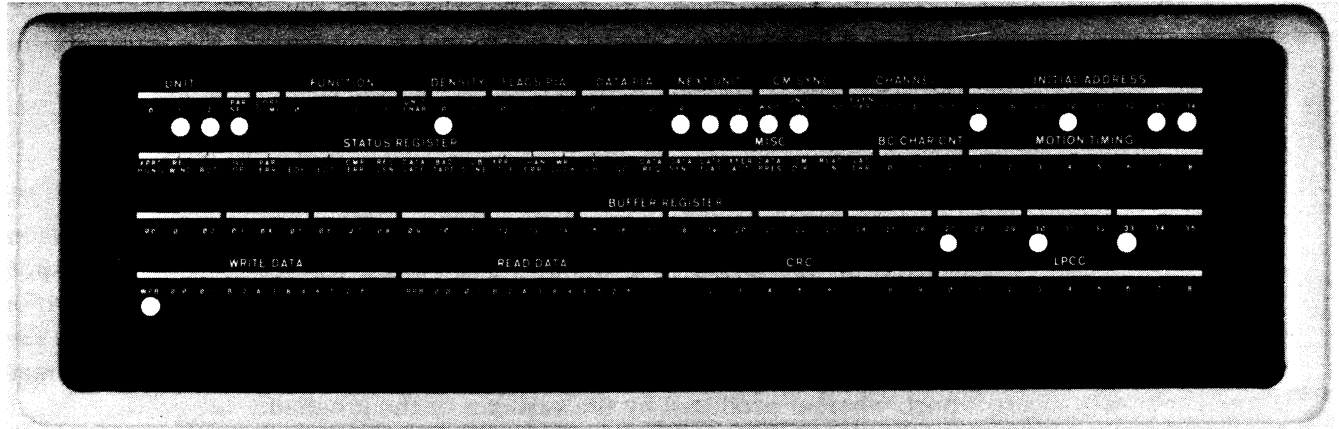
The RWB lights, in the upper right of the DECTape control panel, display the contents of the read-write buffer through which all data passes between control and tape. The remaining lights in the top row display the conditions read by a CONI DTC, except that the left three lights differ somewhat in meaning. The GO light indicates that the selected transport is in motion and



is thus the opposite of bit 18 of the input conditions. The on and off states of the second light indicate reverse and forward motion, and are thus equivalent to 1s in bits 20 and 19 respectively. Similarly the third light indicates whether the addressed unit is selected or not, and thus the on or off state is equivalent to a 1 in bit 23 or 22. The other two rows of lights are mostly for status conditions. The interrupt enabling lights in the second row are not labeled, but they correspond to the flags as labeled in the third row; UPS1 is Up to Speed (CONI DTS, bit 8). At the right end of the two rows are several control flipflops. WREN enables writing, and it is on whenever Active (ACT) is set, provided there is no conflict with a switch setting in a write function. READ IN is on at the beginning of readin mode until the first data is read, but it is on throughout a tape rocker operation. DATA NEED indicates that the buffer is ready to receive a word from the program in a write function; Data Missed sets if DATA NEED is still on when the control is ready to take another half word from the buffer.

#### H4.2 STANDARD MAGNETIC TAPE TM10

In a tape system, activity by an operator is confined to the transport, but information about the internal operation of the control is displayed on a panel at the top of the control cabinet. The lights in the left half of the top row display bits 18–35 (*ie* the right half word) of the input conditions read by a



7140-2

CONI TMC,; the NEXT UNIT lights located at the right display bits 15–17. The next three lights control the synchronization of tape commands: WAIT is on from the time a function is given until the addressed transport is ready; UNIT OK is on from the time a new unit number is loaded until the control starts moving tape (or Job Done sets in a function that does not involve tape motion); CONT is on between functions when the tape remains in motion. The rest of the lights in the top row are for the data channel and include the initial control word address and Write Even Parity. The three remaining lights indicate the following conditions: STRT indicates the channel is connected to this device or any device of lower priority, ACT is on from the time the control needs the channel until it disconnects, and SYNC is on

TM10 Indicator Panel

while the control is waiting to connect to the channel.

The left half of the second row displays the status conditions read by CONI TMS, except that GO is the inverse of Load Next Unit, *ie* it is on while the control is moving tape. (BOT, EOF and EOT respectively are beginning of tape, end of file and end of tape.) The next set of lights is as follows.

DATA SYNC	Synchronizes the data IO instructions to the TM10A and the data channel to the TM10B.
DATA FLAG	This is an internal data request flag: it synchronizes transfers to and from BR.
XFER ACT	Transfers and spacing occur while this light is on; it is turned off by a CONO TMS,1 or an internal stop condition.
DATA PRES	The tape is within a data record.
CM DIR	This light goes on when the selected tape is up to speed in reverse, and then stays on until a tape is up to speed forward or the control stops tape and executes a function that does not move tape.
READ IN	The control is initiating Rewind in readin mode.
LAT ERR	A lateral parity error has been detected. This light is not equivalent to Tape Parity Error, which can go on spuriously during a record.

The rest of the row displays the character counter and a counter that controls tape motion outside of the record areas.

The third row of lights displays BR. The fourth row displays the character buffers through which data is sent to and received from the transport and the registers in which CRC and LPCC are computed.

### Tape Transport TU10

The control panel is at the lower left of the transport door. At the bottom center of the panel is a thumbwheel switch for selecting the transport number. The remaining controls are all rocker switches, of which the two on either side of the thumbwheel have three positions and the three in the row above have two. The lights above the switches indicate the state of the transport, whether produced by the switches or the program.

The upper left switch controls power to the transport, whose state is indicated by the PWR light at the top. When the lower left switch is in the unmarked center position, the vacuum motor is off and the brakes are on; pressing in the lower part of the switch releases the brakes, whereas pressing in the upper part turns on the vacuum motor to draw tape into the buffer columns. Pushing the upper part of the switch above the thumbwheel places the unit on line provided it is ready for use by the program, *ie* the tape is properly loaded, etc. Pressing the lower part takes the unit off line and enables the operator to control tape motion by means of the two switches at the right: the upper switch starts and stops the tape, which moves in the direction selected by the lower switch.

Of the lights at the top, LD PT and END PT indicate tape position; FWD,

REV and REW indicate tape motion called for by the program or the operator. The remaining lights indicate the following.

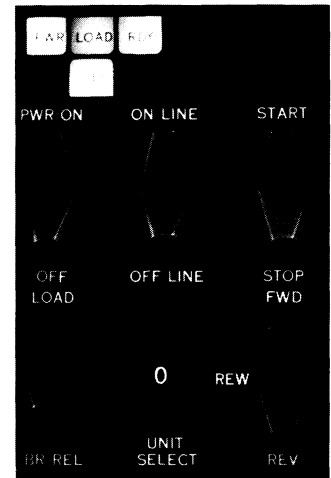
LOAD	The vacuum is on and tape is in the buffer columns.
RDY	The transport is ready for use by the control.
FILE PROT	The right enable ring is absent from the supply reel.
OFF LINE	The transport is unavailable to the control, and the operator can use the switches to manipulate the tape.
SEL	The transport is currently selected by the control.
WRT	The control is now writing or erasing tape.

To load a tape, first clean the tape path [§H7.1], then follow this procedure.

1. Place a write enable ring in the groove on the supply reel if writing is to be allowed; if the data on the tape must not be written over or erased, make sure there is no ring in the groove.
2. With the brakes on and the vacuum motor off (the LOAD/BR REL switch in the unmarked center position), rotate the reel holddown knob of the lower hub counterclockwise as far as it will go, and place the supply reel over the holddown knob with the groove toward the back. Hold the reel firmly against the hub flange and turn the holddown knob clockwise until it is tight. If there is no takeup reel, install one on the top hub in the same manner.
3. Press BR REL and pull the tape from the supply reel. The tape should unwind from the left with the oxide (dull) side toward the hub. If the tape unwinds from the top, check that the reel is mounted with the groove toward the back.
4. The large plate at the left of the reels covers the vacuum columns. At the upper right of the large plate is a smaller plate that covers the tape heads, and at the top is an even smaller one that covers the upper tape guide. The two small plates protrude beyond the surface of the large plate. Grasp the tape in both hands with the left hand at the end and the right hand back about a foot. Place the tape against the bottom and left side of the head cover over the threading slot. Pull the tape taut and slide it upwards and into the slot. Lead the tape around the upper right corner of the vacuum column cover, slide it into the slot between the cover and the upper tape guide, and lead it around the guide. Place the end of the tape over the top of the takeup reel hub, and wind about six turns of the tape clockwise on the reel.
5. Press LOAD to draw tape into the vacuum columns. Select FWD and push START to advance the tape to loadpoint. When the loadpoint marker is sensed, the tape stops, the FWD light goes out, and LD PT comes on.

#### NOTE

If tape motion continues for more than ten seconds, the tape must be beyond loadpoint. Press STOP, select REV and press START. The tape should move back and stop at loadpoint.



TU10 Control Panel

#### CAUTION

If the reel is mounted correctly but the tape unwinds from the right or the shiny side is toward the hub, the tape has been wound improperly and cannot be used.

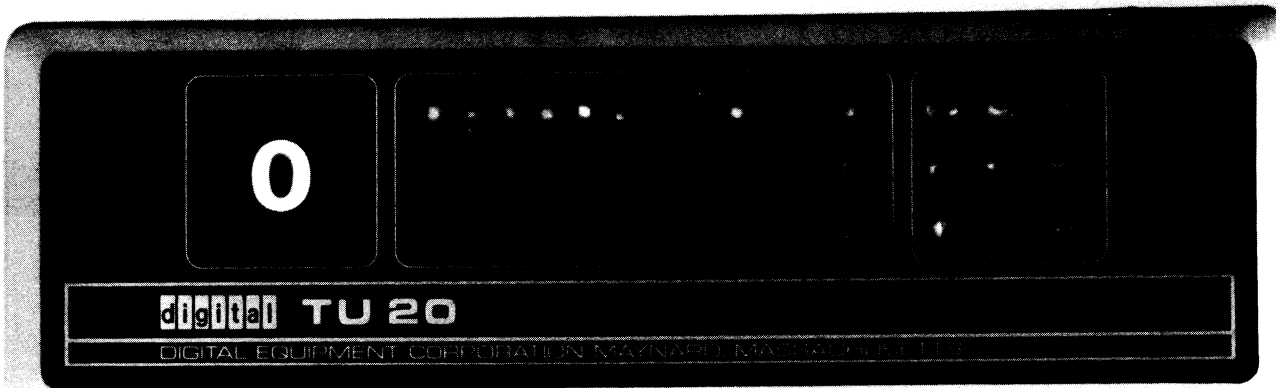
To remove a reel, first rewind to loadpoint. Press BR REL and wind the remaining tape onto the supply reel. Turn the holddown knob counterclockwise and remove the reel.

*CAUTION*

When hand winding tape, do not jerk the reel. This could cause irreparable damage by stretching or compressing the tape.

### Tape Transport TU20

At the left on the control panel at the top of the transport is a thumbwheel switch for selecting the transport number. In the center of the control panel are the operating switches, which are turned on by pushing in the upper part. The lights above the switches indicate the state of the transport whether produced by the switches or the program. The switch at the right applies



5208-1

TU20 Control Panel

power to the transport. The switch at its left places the unit on line provided it is ready for use by the program, *ie* the tape is properly loaded, the door closed, etc. The rightmost switch in the group at the left takes the transport off line, enabling the remaining switches for local control. The LOAD light indicates the tape is in the vacuum columns; pressing the LOAD switch moves the tape *forward* to loadpoint. REWIND, REV and FWD move the tape in the specified manner. RESET stops the tape, but the tape also stops automatically if loadpoint or endpoint is encountered in forward or loadpoint is encountered in rewind. Of the lights at the right, LOADPOINT and ENDPOINT indicate tape position and the numbered lights indicate the transport type (7 or 9 track). The remaining lights indicate the following.

SELECT	The transport is currently selected by the control.
READY	The transport is ready for use by the control.
WRITE LOCK	The write enable ring is absent from the supply reel.
WRITE STATUS	The control is now writing or erasing tape.
REWIND	The tape is now rewinding.

To load a tape, first clean the tape path [§H7.1], then follow this procedure.

1. Place a write enable ring in the groove on the supply reel if writing is to be allowed; if the data on the tape must not be written over or erased, make sure there is no ring in the groove.

2. Rotate the reel holddown knob of the upper reel counterclockwise as far as it will go, and place the supply reel over the holddown knob with the groove toward the back. Hold the reel firmly against the turntable and turn the holddown knob clockwise until it is tight.
3. The section at the left between the two reels is the head cover; just at the right of it is the threading slot, and still further right is the transport switch. Hold the switch in the BRAKES position while pulling about three feet of tape from the supply reel. The tape should unwind from the bottom of the reel with the oxide (dull) side toward the hub. If the tape unwinds from the top, check that the reel is mounted with the groove toward the back.
4. Grasp the tape in both hands with the right hand at the end and the left hand back about fifteen inches. Place the tape against the right side of the raised portion of the head cover over the threading slot. Pull the tape taut and move it downward and into the threading slot. Pull the remaining slack tape through the slot, wrapping it around the vacuum chamber guides.
5. Place the end of the tape over the top of the takeup reel hub. Push the transport switch to BRAKES and wind about six turns of tape clockwise on the reel. Make sure there is no slack between the two reels and release the transport switch.
6. Make sure the tape is properly lined up with the openings of the vacuum chambers. Push the transport switch to START; this causes the reel motors to pull the tape into the vacuum chambers. When the motors have stopped, release the switch.
7. Close the door, press LOAD to position the tape at loadpoint, and when the tape stops, place the unit on line.

To remove a reel, first rewind the tape to loadpoint. Open the door, push the transport switch to BRAKES and wind the remaining tape onto the supply reel. Turn the holddown knob counterclockwise and remove the reel.

### Tape Transport TU30

At the right of center on the control panel at the top of the transport is a thumbwheel switch for selecting the transport number (positions 8 and 9 are not used). The round lights at the left of the thumbwheel indicate the density when the unit is selected by the control. The square lights at the right indicate when a write enable ring is in the supply reel and when the tape is beyond the endpoint.

At the left are six illuminated pushbuttons, five of which light when on.

TU30 Control Panel



AUTO places the unit on line provided it is ready for use by the program, *ie* the tape is properly loaded, the door closed, etc. LOCAL-RESET takes the unit off line, enabling the remaining switches for local control. LOAD-POINT initiates a search that positions the tape at loadpoint: the tape first moves forward, and if it does not reach the loadpoint within twenty-five feet, the tape then goes in reverse until it does (the light is on only when the tape is so positioned). FORWARD and REVERSE move tape in the direction indicated; pressing LOCAL-RESET or AUTO stops the tape, but the tape also stops automatically if it reaches endpoint going forward or loadpoint going reverse. The last button rewinds the tape to loadpoint.

To load a tape, first clean the tape path [§H7.1], then follow this procedure.

1. Place a write enable ring in the groove on the supply reel if writing is to be allowed; if the data on the tape must not be written over or erased, make sure there is no ring in the groove.
2. Rotate the reel holddown knob of the right reel counterclockwise as far as it will go, and place the supply reel over the holddown knob with the groove toward the back. Hold the reel firmly against the turntable and turn the holddown knob clockwise until it is tight.
3. Located just below and between the two reels is the head cover; just below it is the threading slot, and still further below is the transport switch. Hold the switch in the BRAKES position while pulling about three feet of tape from the supply reel. The tape should unwind from the right side of the reel with the oxide (dull) side toward the hub. If the tape unwinds from the left, check that the reel is mounted with the groove toward the back.
4. Grasp the tape in both hands with the left hand at the end and the right hand back about fifteen inches. Bring the tape up against the protruding lip at the bottom of the head cover over the threading slot. Pull the tape taut and move it left and into the threading slot. Pull the remaining slack tape through the slot, with it running outside the supply reel guide.
5. Run the tape outside the takeup guide and over the top of the takeup reel hub. Push the transport switch to BRAKES and wind about six turns of tape clockwise on the reel. Make sure there is no slack between the two reels and release the transport switch.
6. Make sure the tape is properly lined up with the openings of the vacuum chambers. Push the transport switch to START; this causes the reel motors to pull the tape into the vacuum chambers. When the motors have stopped, release the switch.
7. Close the door, press LOADPOINT to position the tape, and when the tape stops, place the unit on line.

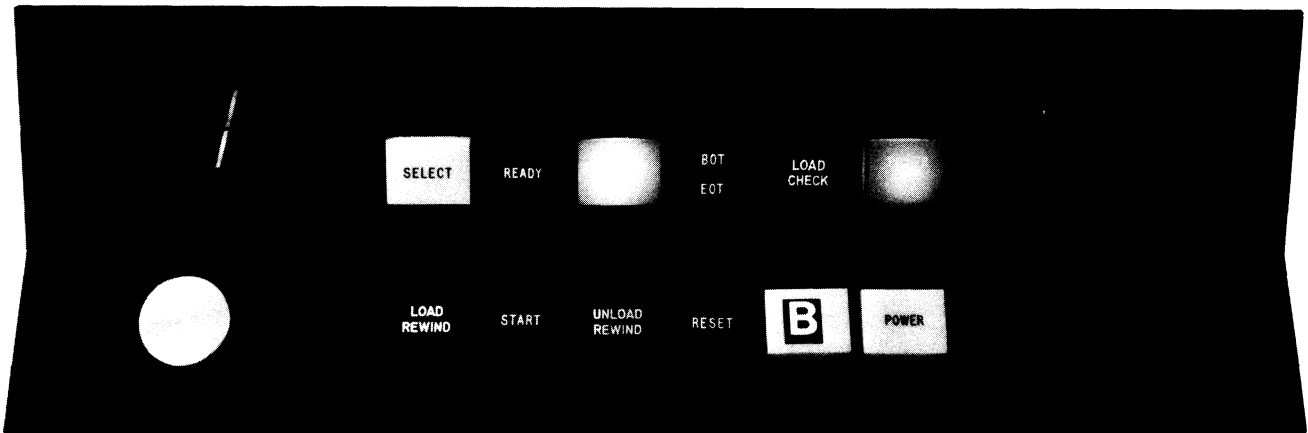
To remove a reel first rewind the tape to loadpoint. Open the door, push the transport switch to BRAKES and wind the remaining tape onto the supply reel. Turn the holddown knob counterclockwise and remove the reel.

#### Tape Transport TU40

At the left on the control panel at the top of the transport is a rotary switch for selecting the transport number; the selected number appears in the

#### CAUTION

If the reel is mounted correctly but the tape unwinds from the left or the shiny side is toward the hub, the tape has been wound improperly and cannot be used.



TU40 Control Panel

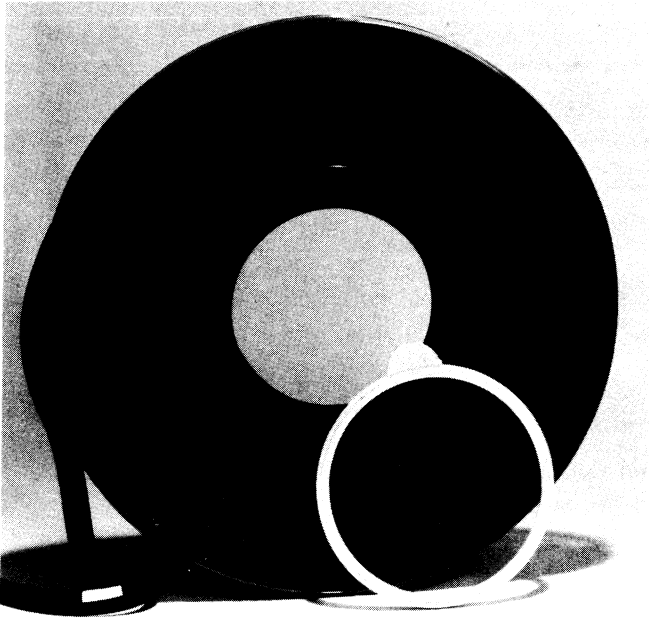
window above the switch. The lower part of the panel contains a row of pushbuttons, all of which are momentary-contact except the one at the right end, which controls power to the transport. The lights above the switches indicate the state of the transport, whether produced by the switches or the program. Pressing **START** places the unit on line provided it is ready for use by the program, *ie* the tape is properly loaded, the window closed, etc. The transport being on line is indicated by the **READY** light, and **SELECT** goes on whenever the transport is selected by the control. **FILE PROTECT** means the write enable ring is absent from the supply reel.

Pressing **RESET** stops the tape and takes the unit off line, enabling the remaining switches for local control. Note however that if the transport is rewinding, pressing **RESET** once switches the unit from rewind to normal reverse speed, and the switch must be pressed again to stop tape. **LOAD REWIND** raises the transport window and loads tape from supply reel to takeup reel, unless the tape is already loaded, in which case it rewinds the tape to loadpoint. Detection of a failure of any sort during loading operations lights **LOAD CHECK** and the unit shuts down. Pressing **UNLOAD REWIND** rewinds the tape to loadpoint, and then unloads it entirely onto the supply reel and lowers the transport window. A tape positioned at loadpoint or endpoint is indicated by the **BOT** and **EOT** lights (beginning of tape, end of tape).

To load a tape, first clean the tape path [§H7.1], then follow this procedure.

1. Place a write enable ring in the groove on the supply reel if writing is to be allowed; if the data on the tape must not be written over or erased, make sure there is no ring in the groove.
2. If the transport is closed, press **UNLOAD REWIND** to lower the window. Pull one end of the reel lock release on the right hub and place the supply reel over the hub. If the reel does not have a cartridge, slip the end of the tape into the shoe at the lower right of the reel. If the reel has a cartridge, place it on the hub so that the projections from the cartridge fit into the two reel positioning guides. The pin on the motor face plate should go into the opening on the tape cartridge.
3. Press **LOAD REWIND** to load the tape automatically. After the window rises (in about seven seconds), press **START** to place the unit on line.

The tape should unwind from the right with the oxide (dull) side toward the hub. If the tape unwinds from the left, check that the reel is mounted with the groove toward the back.



Tape Marker and  
Write Enable Ring

4906

To remove a reel, press RESET and then UNLOAD REWIND. Once the window has opened, remove the tape by opening the reel lock release and pulling the reel from the hub.



**H5 DISKS AND DRUMS**

*To be added – in the meantime refer to Chapter 7.*



## H6 DATA COMMUNICATIONS

Some data communication systems actually include small computers, and for information about such computers the operator must refer to the appropriate computer handbook. In particular the DC68A is based on a PDP-8/I computer and has no controls or indicators of any kind except those on the minicomputer and on the DA10 interface [Appendix H3] through which it is connected to the PDP-10.

### Data Line Scanner DC10

The lights and switches on the DC10 indicator panel are primarily for maintenance, although the programmer should keep Ready Hold on continuously by pushing up the DTR DIS toggle switch; the state of the signal is indicated by the DTR OS light (the program triggers a one-shot to hold it on).



5919-3

The scanner actually checks groups of eight lines, and only on encountering a group with some flag on (indicated by the SCAN light) does it then scan through the individual lines in the group. It stops when it finds an individual flag on, and either TRAN FGST or RCVR FGST goes on depending on whether a transmitter or receiver flag is strobed (both lights may go on at once). PI REQ goes on indicating the scanner is requesting an interrupt on the channel indicated by the lights in the center of the top row, for the line whose number is displayed by the lights in the left half of the bottom row. Then CNT RSTR and CNT HOLD go on, the latter stopping the count clock. If RCVR FGST is on, RCVR FLAG also goes on; it is this flipflop in the scanner logic that actually controls the direction of transfer and the restarting conditions. When the program responds, RCVR FLAG being on causes the DATAI to read data and turn off CNT RSTR; otherwise, the program must give a DATAO (indicated by the light so labeled), which loads the transmitter and turns off CNT RSTR. This last flipflop synchronizes the restarting of the scanner: when it goes off, the next pulse from the clock source turns off CNT HOLD so that subsequent pulses advance the scan counter.

Setting the MAINT switch up completely divorces the scanner from the IO bus (except for the IO reset) and disables Ready Hold, allowing the operator to check the scanner and terminal circuits without a program and without

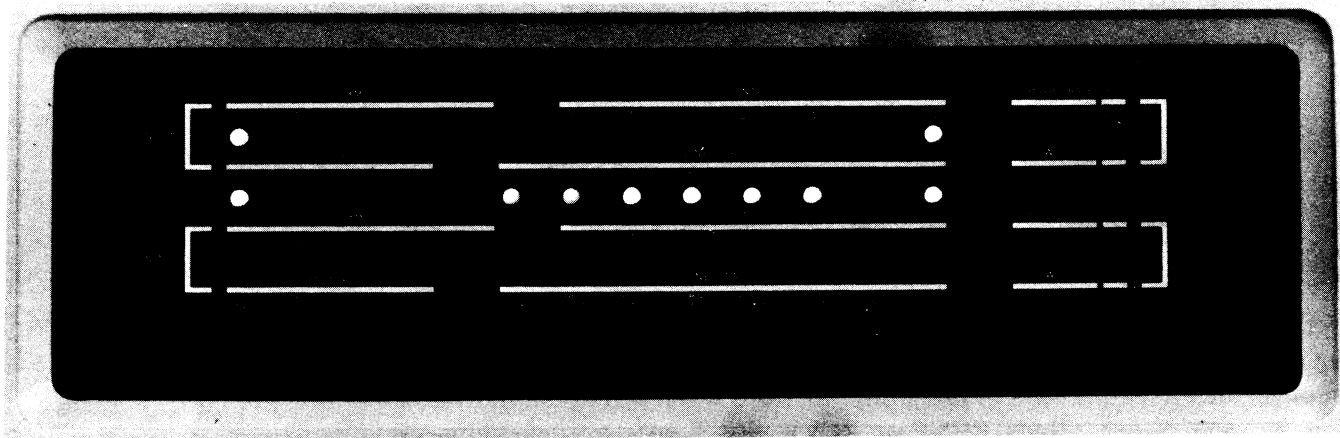
DC10 Control Panel

TRANS FGST and RCVR FGST are input condition bits 31 and 32; RCVR FLAG is DATAI bit 27.

interference from remote stations through the data sets. With the switch on, any character typed in from a terminal is transmitted back over the same line. The transmitter and receiver flags strobed by the scanner for the line selected by the group and unit rotary switches are indicated by LINE TRAN and LINE RCVR, which stay on for about a tenth of a second. In response to a receiver flag, the maintenance logic loads and enables the transmitter; in response to a transmitter flag, the maintenance logic simply turns off the transmitter.

### Single Synchronous Line Unit DS10

The indicator panel contains two identical sets of lights for two DS10s mounted in the same cabinet. The high- and low-priority lights at the right end in each set indicate respectively the PI channel assignments for data and flags; the INTER lights indicate when an interrupt is being requested on the corresponding channel. The line status lights in the second row display the actual modem control signals, several of which are available to the program as status. The clock lights are the receiver and transmitter clocks from the data set.



DS10 Indicator Panel

Although some lights represent flags that may remain on for a significant interval, most are relatively meaningless while the unit is running unless they represent repetitive functions, wherein a soft glow indicates that the function is operative (as with the clock lights). The remaining lights are as follows.

#### XMIT

ACTIVE	Transmitter Active.
INC CNT	The counter that controls character transmission is now being incremented.
IDLE	Idle.
SYNC CHAR	The first character in the transmitter word buffer is the sync character.
LAST BIT	The last bit of a character is now being transmitted.

#### RECEIVE

ACTIVE	Receiver Active.
--------	------------------

INC CNT	The counter that controls the assembly of a character is now being incremented.
SYNC CHAR	The character just received is a sync character.
EOT CHAR	The character just received is an EOT character.
CONTROL	
UNIT SELECT	This DS10 is now being selected by an IO instruction (with either device code).
XMIT FLAG	Transmitter Done.
REC FLAG	Receiver Done.
6-BIT MODE	Inverse of the 8-bit Length flag.
END FLAG	End error.
ERROR FLAG	Data Error.
EOT FLAG	EOT Received.



## H7 CLEANING PROCEDURES

As indicated at the beginning of Appendix F, the exterior of all equipment should be vacuumed at least weekly. Here we are concerned with special procedures for cleaning the interiors of specific peripherals. Every installation should have the following equipment and materials for cleaning.

A vacuum cleaner with rubber or plastic attachments and an air blower outlet.

Ordinary commercial cleaners and solvents for use on sheet metal and glass panels.

Lint free wipers (cloth or paper), such as Kimwipes type 900-S, stock no. 3415 (about 8" × 5").

Isopropyl alcohol, at least 90%, such as Merck or NF (99% by weight) or Lilly (91% by volume).

Various size swabs (cotton-tipped sticks), such as Q-tips.

A small stiff brush.

Cleaning procedures are most critical for tapes and disk packs, but other equipment must receive regular attention as well. It is especially important to clean air intake filters on all equipment regularly.

Store alcohol in its original container or in glass.

### H7.1 TAPE EQUIPMENT

It is imperative that the tape path be kept clean. Allowing the oxide that wears off the tape to build up in the path can cause data errors and even tape damage.

#### DECtape

DECtape is somewhat less sensitive to dirt than standard magnetic tape, but should still be cleaned at least weekly, and much more frequently with heavy usage or in an excessively dirty environment. Use only the head cleaning solvent supplied in the DEC head cleaning kit (Potter Cleaning Kit A425484B).

Unload all tapes, and with a wiper remove all lint, dust and loose oxide from the front mounting panel. With a swab moistened in solvent clean the oxide from the edges of the tape guides and abutting surfaces. If very old, hard oxide deposits have formed on the edges of the tape guides, remove them with a pointed wooden dowel that has been soaked in solvent. Clean the tape guide path and the top of the read/write head with a wiper moistened in solvent, and then remove excess solvent with a dry wiper. Wait at least a minute for any remaining solvent to evaporate before reloading the tapes.

At least monthly clean the exterior and the interior of the cabinet with a vacuum cleaner and other ordinary cleaning materials. Clean the most frequently used tapes by placing a clean, dry wiper over the read/write head

#### CAUTION

Do not allow the solvent to come in contact with the tape or any painted surface, and make sure that all cleaned surfaces are completely dry before loading tape.

Be careful not to contaminate the solvent in the container with dirt from a used swab or wiper.

and manually running the tape over the cloth. Periodically clean all tapes in this manner. Clean any takeup reel on which oxide appears around the hub.

### Standard Magnetic Tape

Always unload the tape before cleaning any part of a transport. Every week vacuum the interior surfaces of the transport using a brush attachment, dust and clean the transport door, and clean the reel hubs with a wiper moistened in alcohol. The tape path should be cleaned at least once *every shift* using the following procedures.

*TU10.* Open the door over the tape buffer columns (the door is held by the two chrome screws on the front). With a swab moistened in alcohol, clean the tape-bearing surfaces of the head, head guides, and the upper and lower guides. Gently clean the capstan and the buffer columns with a wiper moistened in alcohol. Do not scrub the capstan and do not touch it with your hands. Wipe the inside glass surface of the door and replace it.

*TU20.* Carefully remove the two glass buffer covers and the path cover. Move the shield away from the head by setting the transport switch to the BRAKES position. With a swab moistened in alcohol, clean the surfaces of the head and shield that normally make contact with the tape and clean the rubber pinchrollers. Gently clean the buffers and the glass buffer covers with a wiper moistened in alcohol. Replace the buffer covers and the tape path cover.

*TU30.* Remove the head cover and lower the capstan cover. With a wiper moistened sparingly in alcohol, scrub the interior metal surfaces and glass cover of one vacuum chamber and its fixed and rotary guides. Clean the other chamber and guides with a fresh wiper. Moisten another wiper and clean the surface of the head and the guides adjacent to the head. Hold the cleaning switch up and with another moistened wiper or swab, clean the forward (left) capstan and pinchroller, gently scrubbing across their surfaces from the left. Hold the cleaning switch down and with a fresh wiper or swab, clean the reverse (right) capstan and pinchroller, scrubbing gently from the right. Clean the vacuum chamber covers using a wiper moistened with a commercial glass cleaner. Raise the capstan cover and replace the head cover.

Every forty hours clean the tape cleaners. Remove the head cover and lower the capstan cover. Carefully remove the caps and ceramic washers of the two tape cleaners. With a swab moistened in alcohol, clean the interiors of the tape cleaners, the cleaning slots, and the ceramic washers. Reassemble the tape cleaners, making sure the round-edged sides of the ceramic washers face the transport. If the washers are installed incorrectly, tape will guide against their sharp edges, dropping mylar and oxide particles in the head area.

*TU40.* Open the tape path cover and lower the ferrite head shield by pressing the black release button at the left. With a swab moistened in alcohol, clean the surfaces of the head and shield that make contact with the tape, the tape cleaner blade, the tape guides, and threading channels. With a moistened wiper clean the air bearings, the vacuum columns and the glass vacuum column surfaces. Oxide deposits may be removed with a small brush. Cover your finger with a wiper and rotate the capstan while lightly wiping the rubber surface with another wiper moistened in alcohol. Wipe

The slide panel in the transport door is made of plexiglass, so clean it only with soap and water or some commercial cleaner. Do not wipe it with a dry cloth as this will create a static charge that attracts dust.



the capstan surface dry, raise the ferrite shield and close the tape path cover.

**Tapes.** The tapes themselves should also be cleaned occasionally. The best way to do this is by using a commercial tape-cleaning machine or service. If a tape contains wanted data, make sure it is *only* cleaned and *not* error checked or certified, which involves erasing and rerecording. Even tapes that are not used (archival tapes) should be rewound about once a year to redistribute stresses and minimize print-through noise.

## H7.2 DISK PACKS

Fixed-head disks and drums are sealed units requiring no interior cleaning. Since disk packs are removable, the heads and the packs themselves must be kept clean. Cleaning should be done by *competent maintenance personnel* using the following equipment in addition to the standard materials listed at the beginning of this appendix.

Pipe cleaners.

Wooden tongue depressors, 6" × 3/4".

A high intensity light or other strong light source.

A piece of white cardboard or stiff paper (8½" × 11").

Inspect the heads for dirt accumulation at least twice monthly (weekly for around-the-clock operation). When necessary, clean them carefully with a swab soaked in alcohol, and clean out the two holes in each head with a pipe cleaner, also soaked in alcohol. Keep all cabinet and pack filters clean and fresh, as dirty heads can be caused by poor head flight due to poor air flow through the pack or cabinet. Always replace a single head that collects dirt while others in the same drive remain clean.

Do not clean all of the packs at an installation frequently just for the sake of cleaning them. If a pack is subject to random errors, or vital information that is not duplicated elsewhere cannot be retrieved, and the problem is not alleviated by cleaning the heads, then clean the pack. In any event, depending upon environmental conditions, test all packs every few months by cleaning a couple of surfaces at random in each; clean all surfaces in any pack in which dirt is discovered.

To clean a pack, mount it on a drive from which the upper case panels have been removed or on a free-standing spindle mechanism that allows the pack to be turned by hand with the cover removed. Place the light with the white cardboard as a background so that plenty of light shines into the pack. Follow this procedure for cleaning each surface.

1. Position the light so the surface is clearly visible.
2. Wrap a fresh wiper around a depressor with the wiper extending 3/8 inch beyond the wood at one end.
3. Soak one side of the wrapped depressor with the alcohol (be sure to wet the full width of the depressor).
4. Spin the pack by hand at 40–60 rpm (use the flat top surface of the plastic bezel on the top of the pack).

5. With the pack spinning, insert the prepared depressor with the protected end toward the center, and press the wet side against the surface. Maintain the pack spin while applying about 5–10 pounds pressure against the surface, making sure to wet the surface across the full width of the tracks. The pressure may be lightened as the surface dries, but be sure to keep the wiper on the surface with the pack spinning until the surface is completely dry and has a high gloss. Keep the pack spinning while removing the depressor, and check the wiper for dirt.
6. Inspect the surface carefully for scratches. If a scratch corresponds in position to a bad sector as determined by the program, then further cleaning is unlikely to make the sector usable.

*WARNING*

Do *not* attempt to use the drive motor when cleaning a pack. Always turn drive power off and spin the pack manually.

Do not clean a pack if either the pack or the alcohol is below 40°F (otherwise water vapor might condense on the surface).

### H7.3 OTHER EQUIPMENT

#### Paper Tape Reader and Punch

Simply wipe the reader clean, being especially careful around the light bulb and the area under it. The punch should be cleaned at least every shift, or better yet every time a new box of tape is loaded. Empty the chad box and then blow out and vacuum the entire unit.

#### Line Printer

Check the line printer every time paper is loaded, and clean it at least daily. With the power off, blow out the paper path through the printer and vacuum the entire unit. Inspect the drum for excessive ink or paper residue. Generally Field Service will clean the drum and ribbon areas, but the operator may clean the drum by wiping it with alcohol and vacuuming the drum housing. If the printer has a changeable drum, remove it for cleaning.

#### Card Reader and Punch

In this equipment, the critical area is the card track, where the accumulation of card dust and chad may cause data errors or even a card jam. The reader track should be cleaned at least weekly, more often if the equipment is used heavily. The punch track should be cleaned every 2000 cards but at least daily.

*CR10D, E, F.* With a swab moistened in alcohol clean the Neoprene surface of the picker sector. If any of the vacuum holes are plugged with card

debris, gently push it through the holes with a paper clip (the vacuum system will remove it). Using a  $\frac{5}{64}$  allen wrench, remove the four button-head screws holding the top track cover. Remove the cover and vacuum the card track, particularly around the picker and stacker castings. Use a small brush to clean around the picker and stacker rollers and the picker sector.

*CR10A/B and CP10.* To expose the card track, press the two black release buttons on the top of the front cover, tilt the cover away and lift it off. With the power off, use the vacuum cleaner to first blow out and then vacuum the track from the hopper to the stacker.



# Index

- A 2-1
- A+1 2-1
- AC 2-1
- access time 1-13
- accumulators 1-4
- ADD 2-27
- Address Break 2-102
- address failure 2-109
- Address Failure Inhibit 2-60, 2-110
- addressing 1-4, 1-13
- address space 1-7, 2-105
- address structure G-3
- AND 2-20
- ANDCA 2-20
- ANDCB 2-21
- ANDCM 2-20
- AOBJN 2-45
- AOBJP 2-45
- AOJ 2-48
- AOS 2-49,
- APR 2-86, 2-98, 2-101, 2-119
- AR 1-3
- arithmetic shifting 2-30, A-19
- arithmetic testing 2-45, A-20
- AS 1-2
- ASCII code B-2
- ASH 2-31
- ASHC 2-31
- associative memory 2-108
- automatic calling 8-4
- auto restart 2-98
  
- BA10 H2-1
- base address 2-112
- base page number 2-105
- base register 2-105
- bit assignments, in-out C-1
- BLIST 2-14
- BLKI 2-83
- BLKO 2-83
- block IO 2-83
- block transfer 2-10
- BLT 2-10
- Boolean functions 2-17, A-15
  
- BR 1-3
- Busy 2-84
- byte manipulation 2-15, A-16
- byte pointer 2-15
  
- CAI 2-47
- CAM 2-47
- card codes B-8
- card punch CP10 4-15
  - cleaning H7-3
  - operation H2-9
  - timing 4-18
- card reader CR10 4-11
  - cleaning H7-3
  - operation H2-7
    - CR10A/B H2-9
    - CR10D, E, F H2-7
  - timing 4-15
- carries 2-26
- Carry 0, Carry 1 2-59
- CCI 5-7
- CDP 4-16
- central processor 1-1
- cleaning F-1, H7-1
- CLEAR 2-18
- CLK 2-121
- clock
  - line frequency 2-99, 2-102
  - real time DK10 1-120
  - operation F1-13, F2-9
- Clock flag 2-99, 2-102
- communication signals 8-3
- compatibility E-1
- complement 1-7
- computer-computer interface 5-7
  - see DA10
- concealed mode 1-5, 2-64
- concealed page 2-101
- conditions in 2-84
  - see status
- conditions out 2-83
  - card punch 4-16
  - card reader 4-11
  - clock 2-121
  - console 2-87, C-2
  - console terminal 3-7, C-9
  - DA10 5-7
  - data channel DF10 5-4
  - DC10 8-29
  - DECtape 6-5, 6-7
  - disk/drum RC10 7-5
  - disk pack RP10 7-20, 7-21
  - DS10 8-37
  - interrupt
    - KA10 2-96, C-7
    - KI10 2-91, C-2
  - line printer 4-4
  - magnetic tape TM10 6-20, 6-22
  - paging 2-112
  - plotter 4-9
  - processor
    - KA10 2-101, C-6
    - KI10 2-98, C-3
  - punch 3-5, C-8
  - reader 3-1, C-8
- CONI 2-81
- CONO 2-81
- CONSO 2-82
- console 2-86
- console in-out 3-1, C-8, H1-1
- console operator panel
  - KA10 F2-1
  - KI10 F1-2
- console terminal 3-6
  - operation H1-2
- CONSZ 2-82
- counting ones 2-75
- CR 4-11
  
- DA10 5-7
  - instructions 5-7
  - programming 5-11
  - status 5-8
  - timing 5-12
- data channel 5-1
  - see DF10
- data communication 8-1
  - signals and procedures 8-3
- data communication system
  - DC68A 8-7
    - call control
      - DC08H 8-19
      - 689AG 8-24
    - data multiplexing 8-11
    - modem control
      - DC08F 8-17
      - 689AG 8-21
- DATAI 2-82
- data line scanner DC10 8-26
  - data line programming 8-33
  - instructions 8-33
  - modem control 8-35
  - operation H6-1
  - status 8-30
  - timing 8-34
- DATAO 2-82
- data set 8-2, 8-6
- decimal print routine 2-77
- DECtape TD10 6-1, H4-1
  - compatibility 6-4
  - format 6-2
  - formatting 6-14
  - handling 6-4
  - instructions 6-5
  - operation H4-1
  - readin mode 6-14

- programming 6-11
- status 6-7, 6-8
- timing 6-12
- DF10 5-1
  - conditions 5-4
  - operation 5-5
- DFAD 2-42
- DFDV 2-43
- DFMP 2-43
- DFN 2-38
- DFSB 2-43
- direct-access processor 1-1
- direct addressing 1-11
- Disable Bypass 2-59, 2-115
- disk 7-1
  - see disk/drum RC10
  - disk pack RP10
- disk/drum RC10 7-2
  - format 7-3
  - instructions 7-4
  - operation 7-14
    - control 7-15
    - disk 7-14
    - drum 7-14
  - programming 7-9
  - status 7-7, 7-9
  - timing 7-11
- disk pack RP10 7-18
  - cleaning H7-3
  - format 7-18
  - functions 7-27
  - instructions 7-20
  - operation 7-30
  - programming 7-28
  - status 7-23, 7-24
  - timing 7-29
- dismissing an interrupt
  - KA10 2-95
  - KI10 2-90
- dispatch interrupt 2-89
- DIV 2-28
- DK10 F1-13, F2-9
- DLS 8-29
- DMOVE 2-44
- DMOVEM 2-44
- DMOVN 2-44
- DMOVNM 2-44
- Done 2-84
- double precision floating
  - point 1-10, 2-42, 2-79
- DPB 2-16
- DPC 7-20
- drum 7-1
  - see disk/drum RC10
- DS 1-2
- DSI 8-37
- DSK 7-5
- DSS 8-37
- DTC 6-5
- DTS 6-5
- E 1-11, 2-1
- E+1 2-42
- effective address calculation 1-11
- 18-bit computer interface DA10 5-7
  - see DA10
- entry point 1-5, 2-64, 2-104
- EQV 2-23
- excess 128 code 1-8
- EXCH 2-9
- execute 2-2
- executive mode 1-5, 2-104, 2-117
- executive process table 2-107
- executive stack pointer 2-113, 2-116
- executive XCT 2-114
- FAD 2-39
- FADR 2-36
- fast memory 1-4
- FDV 2-41
- FDVR 2-37
- 50 Hertz 2-99
- First Part Done 2-60
- FIX 2-34
- fixed point arithmetic 2-26, A-16
- fixed point numbers 1-7
  - double length 1-8
- FIXR 2-35
- flags 2-58
- flag restoration 2-63
- Floating Overflow 2-60, 2-102
- floating point arithmetic 2-31, A-16
- floating point numbers 1-8
  - double precision 1-9, 2-79
- Floating Underflow 2-61
- FLTR 2-35
- FMP 2-40
- FMPR 2-37
- formats A-2
- FSB 2-40
- FSBR 2-37
- FSC 2-34
- full word data transmission 2-9, A-17
- half word data transmission 2-2, A-18
- HALT 2-64
- hardcopy control H2-1
- hardcopy equipment 4-1, H2-1
- hardware addressing 1-13
- hardware read in = read in
- HLL 2-3
- HLLC 2-4
- HLLD 2-4
- HLLZ 2-4
- HLR 2-7
- HLRE 2-8
- HLRO 2-8
- HLRZ 2-8
- HRL 2-5
- HRLE 2-6
- HRLO 2-5
- HRLZ 2-5
- HRR 2-6
- HRRE 2-7
- HRRO 2-7
- HRRZ 2-6
- I 1-11
- IBP 2-16
- IDIV 2-29
- IDPB 2-17
- ILDB 2-16
- IMUL 2-28
- indicator panels F1-5, F2-4
- indicators
  - KA10 F2-1
  - KI10 F1-2
- index registers 1-4
- indirect addressing 1-11
- initial conditions 2-83
- in-out 2-80, A-19
- in-out bit assignments C-1
- in-out devices A-12, App. C
- In-out Page Failure 2-99, 2-109
- input-output 2-80, A-19
- input-output codes B-1
- instruction format 1-10
- instruction modes 2-1
- instructions A-13
  - arithmetic testing 2-45, A-20
  - Boolean functions 2-18, A-15
  - byte 2-16, A-16
  - double moves 2-44, A-17
  - fixed point 2-27, A-16
  - floating point 2-34, A-16
    - double precision 2-42
    - single precision 2-36
    - without rounding 2-38
    - with rounding 2-36
  - full word 2-9, A-17
  - half word 2-3, A-17
  - in-out 2-80, A-19
  - jump 2-61, A-19
  - logic 2-18, A-15
  - logical testing 2-52, A-21
  - move 2-11, A-17
  - number conversion 2-34, A-17
  - pushdown 2-13, 2-67, A-19
  - scaling 2-34
  - shift 2-25, 2-31, A-19
  - rotate 2-25, A-19
- instruction times 2-2, D-1
  - KA10 D-9
  - KI10 D-3

- interleaving 1-13, G-1
- interrupt 2-87
- interrupt conditions
  - KA10 2-96, C-7
  - KI10 2-91, C-2
- interrupt functions 2-88
- interrupt instructions
  - KA10 2-94
  - KI10 2-89
- interrupt request 2-87
- IO 2-80
- IO bit assignments C-1
- IOR 2-21
- IR 1-3
  
- JCRY 2-62
- JCRY0 2-62
- JCRY1 2-62
- JEN 2-64
- JFCL 2-61
- JFFO 2-61
- JFOV 2-62
- JOV 2-62
- JRA 2-66
- JRST 2-63
- JRSTF 2-64
- JSA 2-66
- JSP 2-62
- JSR 2-62
- JUMP 2-47
  
- kernel mode 1-5
- KEY RDI = read in
  - KA10 F2-3
  - KI10 F1-6
  
- LDB 2-16
- line printer LP10 4-1, 4-8
  - cleaning H7-3
  - codes B-4
  - instructions 4-3
  - operation
    - LP10A H2-6
    - LP10B, C, D, E H2-4
    - LP10F, H H2-1
  - output format 4-2
  - speed 4-3
  - timing 4-6
- logic 2-17, A-15
- logical shifting 2-24, A-19
- logical testing and
  - modification 2-51, A-21
- LPT 4-4
- LSH 2-25
- LSHC 2-25
- LUUO 2-70
  
- MA 1-2
  
- machine modes 1-4
- magnetic tape 6-1
  - care for H4-1
  - cleaning H7-1, H7-3
  - see DECTape
    - standard magnetic tape
- maintenance panel F1-2, F2-2
- MAP 2-113
- margin check panel F1-2, F2-2
- margins 2-101
- memories G-1
  - MA10 G-4
  - MB10 G-5
  - MD10 G-6
  - ME10 G-8
  - MF10 G-9
- memory 1-12
- memory access time 1-13
- memory allocation 1-14
- memory management
  - KA10 2-117, C-7
  - KI10 2-104, C-4
- memory protection 2-117
- Memory Protection 2-102
- memory stop F1-4, F2-3
- MI 1-2
- mnemonics 1-15, A-1
  - alphabetic A-8
  - derivation A-4
  - device A-12
  - numeric A-5
- modem 8-2
- modes, instruction 2-1
  - arithmetic testing 2-46
  - fixed point 2-27
  - floating point 2-36, 2-39
  - half word 2-3
  - logic 2-17
  - logical testing 2-52
  - move 2-11
- modes, machine 1-4
- Monitor programming
  - KA10 2-119
  - KI10 2-111
- MOVE 2-11
- MOVM 2-12
- MOVN 2-11
- MOVS 2-11
- MQ 1-2
- MUL 2-28
- MUUO 2-71
  
- nested subroutines 2-67
- No Divide 2-61
- Nonexistent Memory 2-99, 2-102
- no-ops 2-58
- normalization 2-33
- normalized operands 1-9
  
- number conversion 2-34, 2-77
- number formats 1-10, A-3
- number system 1-7
  - fixed point 1-7
  - floating point 1-8
  
- octal-to-decimal conversion 2-77
- ones complement 1-7
- operating keys
  - KA10 F2-3
  - KI10 F1-6
- operating switches
  - KA10 F2-7
  - KI10 2-99, F1-8
- operation App. F, G, H
  - BA10 H2-1
  - card punch H2-9
  - card reader H2-7
  - clock DK10 F1-13, F2-9
  - console terminal H1-2
  - DA10 5-12
  - data channel 5-5
  - DC10 H6-1
  - DECTape H4-1
  - disk/drum RC10 7-14
  - disk pack RP10 7-30
  - DS10 H6-2
  - line printer H2-1
  - magnetic tape TM10 H4-3
  - memories App. G
  - processor F-1
    - KA10 F2-1
    - KI10 F1-1
  - plotter H2-6
  - punch H1-1
  - reader H1-1
  - tape transport TU10 H4-4
  - tape transport TU20 H4-6
  - tape transport TU30 H4-7
  - tape transport TU40 H4-8
- OR 2-21
- ORCA 2-21
- ORCB 2-22
- ORCM 2-22
- overflow 2-26, 2-33
- Overflow 2-59, 2-102
- overflow trapping 2-69
  
- PAG 2-112
- page 2-105
- Page Enable 2-112
- page failure 2-109
- page fail word 2-109
- page map 1-5, 2-105
- page number 2-105
- page refill 2-109
- page refill failure 2-110
- page table 2-108

- paging 1-5, 2-105
- paper tape punch 3-4
  - cleaning H7-4
  - operation H1-1
- paper tape reader 3-1
  - cleaning H7-4
  - operation H1-1
- parity 2-72
- Parity Error
  - KA10 2-97
  - KI10 2-99
- PC 1-2
- PC word 2-58
- PDP-10 1-1
- peripheral equipment App. H
- per process area 2-105
- PI 2-86, 2-96
- plotter XY10 4-8
  - operation H2-6
  - timing 4-10
- PLT 4-9
- pointer
  - byte 2-15
  - IO block 2-83
- POP 2-13
- POPJ 2-68
- PORTAL 2-64
- Power Failure
  - KA10 2-97
  - KI10 2-99
- powers of two A-23
- printer 4-1 *see* line printer
- priority interrupt 2-87
  - KA10 2-94
  - KI10 2-88
- processor compatibility E-1
- processor conditions 2-98
  - KA10 2-101
  - KI10 2-98
- processor identification 2-72
- processor serial number 2-113
- process table 2-105, 2-107
- program control 2-58, A-19
- program management
  - KA10 2-119
  - KI10 2-104
- programming conventions 1-15
- programming examples 2-72
- program stop F1-4, F2-3
- proprietary violation 2-110
- protection 2-117
- PTP 3-5
- PTR 2-87, 3-1
- Public 2-60, 2-99
- public mode 1-5
- public page 2-101
- PUSH 2-13
- pushdown list 2-12, A-19
  - defined 2-13
  - subroutines 2-68
- Pushdown Overflow 2-102
- PUSHJ 2-67
- RDI = read in
- read in F1-6, F2-5
- readin mode 2-85, 3-3, 6-14, 6-32
- real time clock DK10 *see* clock
- recursive MUUOs 2-116
- relocation 2-117
- reload counter 2-109, 2-113
- restore 2-63
- reverse BLT 2-13
- reverse digits 2-29
- RIM = read in
- ROT 2-25
- rotate 2-24, A-19
- ROTC 2-26
- rounding 2-36
- RSW 2-86
- RUN F1-3, F2-2
- scaling 2-33
- sense switches 1-2, 2-99
- serial number 2-113
- SETA 2-18
- SETCA 2-19
- SETCM 2-19
- SETM 2-19
- SETO 2-18
- SETZ 2-18
- shift A-19
  - arithmetic 2-30
  - logical 2-24
- single precision floating point 2-34
- single synchronous line
  - unit DS10 8-36
    - instructions 8-37
    - operation H6-2
    - programming 8-40
    - status 8-39
    - timing 8-42
- SKIP 2-48
- small user 1-14, 2-104
- Small User 2-112
- small user violation 2-110
- software double precision 2-79
- SOJ 2-49
- SOS 2-50
- stack pointer 2-113, 2-116
- standard magnetic tape TM10 6-16
  - cleaning H7-2
  - format 6-16
    - core dump 6-18
    - 7-track 6-17
    - 9-track 6-17
  - functions 6-27
  - read 6-28
  - read-compare 6-29
  - rewind 6-31
  - space 6-30
  - write 6-27
- instructions 6-19
- operation H4-1, H4-3
  - control H4-3
  - TU10 H4-4
  - TU20 H4-6
  - TU30 H4-7
  - TU40 H4-8
- programming 6-31
- readin mode 6-32
- status 6-21, 6-23
- timing 6-33
  - TU10 6-33
  - TU20 6-34
  - TU30 6-35
  - TU40 6-35
- transport *see* operation
- status 2-84
  - card punch 4-16
  - card reader 4-12
  - console terminal 3-7, C-9
  - clock 2-121
  - DA10 5-8
  - data channel 5-4
  - DC10 8-30
  - DEctape 6-7, 6-8
  - disk/drum RC10 7-7, 7-9
  - disk pack RP10 7-23, 7-24
  - DS10 8-39
  - interrupt
    - KA10 2-97, C-7
    - KI10 2-92, C-2
  - line printer 4-4
  - magnetic tape TM10 6-21, 6-23
  - paging 2-113
  - plotter 4-12
  - processor
    - KA10 2-102, C-6
    - KI10 2-99, C-3
  - punch 3-5, C-8
  - reader 3-2, C-8
- SUB 2-27
- supervisor mode 1-5
- subroutines 2-65
- switches
  - KA10 F2-7
  - KI10 F1-8
- table searching 2-78
- tape transport *see* TU
- TD10 6-1, H4-1
  - see* DEctape
- TDC 2-55
- TDN 2-55



- TDO 2-56
- TDZ 2-55
- test instructions
  - arithmetic 2-45, A-19
  - logical 2-52, A-21
- timer 2-99
- time sharing 1-4
- timing 2-1, D-1
  - card punch 4-18
  - card reader 4-15
  - charts
    - KA10 D-9
    - KI10 D-6
  - clock 2-122
  - console terminal 3-8
  - DA10 5-12
  - DC10 8-34
  - DECtape 6-12
  - disk/drum RC10 7-11
  - disk pack RP10 7-29
  - DS10 8-42
  - interrupt
    - KA10 2-97
    - KI10 2-93
  - line printer 4-6
  - plotter 4-12
  - processor D-1
    - KA10 D-3
    - KI10 D-9
  - punch 3-6
  - reader 3-2
  - tape transport TU10 6-33
  - tape transport TU20 6-34
  - tape transport TU30 6-35
  - tape transport TU40 6-35
- Time Out 2-99
- TLC 2-54
- TLN 2-53
- TLO 2-54
- TLZ 2-54
- TM10 *see* standard magnetic tape TM10
- TMC 6-20
- TMS 6-20
- Trap 1, Trap 2 2-60
- Trap Offset 2-102
- trapping, traps
  - overflow 2-69
  - page failure 2-109
  - Uuo 2-70
- TRC 2-53
- TRN 2-52
- TRO 2-53
- TRZ 2-52
- TSC 2-57
- TSN 2-56
- TSO 2-57
- TSZ 2-56
- TTY 3-7
- TU10
  - cleaning H7-2
  - operation H4-4
  - timing 6-33
- TU20
  - cleaning H7-2
  - operation H4-6
  - timing 6-34
- TU30
  - cleaning H7-2
  - operation H4-7
  - timing 6-35
- TU40
  - cleaning H7-2
  - operation H4-8
  - timing 6-35
- TU55, TU56
  - cleaning H7-1
  - operation H4-1
- 12-bit computer interface DA10 5-7
- twos complement 1-7
- UFA 2-38
- unassigned codes 2-71
- unimplemented operations 2-70
- User 2-60
- User Address Compare Enable 2-112
- user fast memory block 2-112, 2-115
- User In-out 2-60, 2-102, 2-115, 2-119
- user mode 1-4, 2-104, 2-119
- user process table 2-107
- user programming
  - KA10 2-119
  - KI10 2-104
- Uuo 2-70
- virtual address space 1-1, 1-6, 1-14, 2-105
- Word Empty 2-113
- word format A-2
- words
- X 1-11
- XCT 2-61, 2-114
- XOR 2-22
- Y 1-11
- @ 1-15
- . 1-16
- : 1-16
- [] 1-16



READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

Did you find errors in this manual? If so, specify by page.

---

---

---

---

---

---

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

---

---

---

---

---

---

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

---

---

---

---

---

---

Please indicate the type of user/reader that you most nearly represent.

- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Non-programmer interested in computer concepts and capabilities

Name \_\_\_\_\_ Date \_\_\_\_\_

Organization \_\_\_\_\_

Street \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip Code \_\_\_\_\_  
or  
Country

If you do not require a written reply, please check here.

Please cut along this line.





