PRIME 200 SYSTEM REFERENCE MANUAL

PREFACE

This manual describes the operation and machine-level programming for the PRIME 200 computer. Chapter 1 provides an introduction to the computer system, including information about the organization of the system, the formats of the various instruction types, the conventions used in machine-level programming, and the software available with the system. Chapter 2 describes the operation and programming of the central processor and gives the basic information needed for handling the input/output system. The remaining chapters describe the individual peripheral devices. Several appendices at the back of the manual provide summary information on such topics as a review of twos complement conventions, instruction mnemonics, instruction execution times, I/O device codes and the like.

CONTENTS

CHAPTER 1
INTRODUCTION

The PRIME 200 is a general purpose, stored program computing system with a 16-bit word length and fully parallel operation. Its MOS memory has a cycle time of 750 nanoseconds and is expandable in 8K word increments to a maximum of 32K words. Standard processor features include double-precision arithmetic, multiply-divide, byte parity in all memory locations and on all data paths, eight programmable direct memory access channels, vectored priority interrupt system, automatic program load from paper tape, and a programmer's control panel.

Comprehensive Instruction Set

The instruction set includes 118 instructions, fifteen of which are single-word, memory reference instructions. This latter group of instructions reduce the register housekeeping overhead associated with arithmetic and logical operations involving a word in memory and the contents of the arithmetic register.

Input/Output

Up to 64 input/output devices can be attached to and addressed by the I/O bus. Available peripheral equipment includes high-speed paper tape reader and punch, teletypewriter (whose interface may be used for any EIA RS-232C compatible device), fixed and moving-head disks, and an A-D subsystem. Program control over these and other devices is provided by a common set of I/O instructions which initiate, control and monitor all data transfers between the arithmetic register, device controllers, and the devices themselves.

Interrupt Processing

Facilities for handling program interrupts and direct memory access data transfers are standard PRIME 200 features. The interrupt system, which may use either a party line or vectored technique, facilitates processor control of the peripheral equipment by allowing any device to interrupt the normal program flow on a priority basis. The processor responds to an interrupt request by executing the instruction addressed by the contents of a particular memory location. In the party-line mode of operation there is only one such location and an interrupt service routine must determine which device to service; in the vectored mode a unique location is assigned to each device. In either mode interrupts can be armed and disarmed singularly or in groups, under program control.

Direct Memory Transfers

A high-speed device, such as a disk, can gain direct access to memory through a DMA channel without requiring the execution of additional instructions after the data transfer has been initiated. In general, DMA transfers are overlapped with processing; the processor pausing only when a transfer occurs between the memory and an I/O device. Any number of devices can be connected to the DMA system, and any eight can operate in this fashion at a given time.

Programmer's Control Panel

The computer is available for either rack-mount or table-top installation. Both versions provide a control panel containing a set of LED indicators through which the program can display information to the operator, and a set of data switches through which the operator can supply data words and addresses to the program. The data switches are also used as sense switches which can be interrogated by the program. Additional switches on the control panel allow the operator to start and stop the program, single-step through a program one instruction at a time, store the contents of the data switches in any memory location, display the contents of any location in the LEDs, and allow the

operator to load information automatically when there is no program in memory.

Microprogrammed Logic

Operation of the PRIME 200 is implemented entirely through microprogramming. Hence the whole system of stored program, program counting, and the like is duplicated at a "micro level" by a stored microprogram in read-only memory, with its own microinstruction counter and all the necessary facilities for executing microroutines—most of which correspond to user-level machine instructions. The microprogram system is completely invisible to the programmer, who sees only his own instructions and their results.

1.1 PROCESSOR ORGANIZATION

The central processor is the control unit for the entire system; it governs all peripheral I/O equipment, performs all arithmetic, logical, and data handling operations, and sequences the program. It is connected to the memory by a memory bus and to the peripheral equipment by an I/O bus. The processor handles words of sixteen bits. For error detection purposes two parity bits are appended to each word and all words are treated as byte pairs. The parity of each byte is checked not only when a word is read from memory, but on every transfer in the processor and over the I/O and memory busses.

Information Formats

The bits of a word are numbered 1 to 16, left to right, as are the bits in the registers that handle the words. An instruction operand is interpreted as a logical word, an address, a pair of 8-bit bytes, or a 16-digit signed or unsigned binary number. The arithmetic instructions operate on fixed-point binary numbers, either unsigned or the equivalent signed numbers using twos complement conventions. Addition, subtraction and shifting can be performed with double precision numbers.

Sequential Instruction Execution

The processor performs a program by executing instructions retrieved from consecutive memory locations as counted by the program counter P. As one instruction is being fetched, P is incremented by 1 so that the next instruction is normally taken from the next consecutive location. Sequential program flow is altered by changing the contents of P, either by incrementing it an extra time in a test skip instruction or by replacing its contents with the value specified by a jump instruction. The address for a memory access is held in register Y, and data read from memory or about to be stored in memory is held in register M.

Arithmetic Register

All computations are performed using the Arithmetic or A register. Data can be moved in either direction between A and any memory location, the contents of a memory location can be combined arithmetically or logically with the contents of A, and this register also serves as the data connection with the I/O bus. A secondary arithmetic register B serves as a right extension of A for double length operations. The processor also has a single-bit register C, which detects overflow in arithmetic operations and saves the last bit dropped out of A or B in shift operations.

Referencing Memory

Each memory reference instruction calculates an effective address. This calculation may include indirection, where an address calculated at an intermediate step is used to retrieve another address, and may include indexing, where a fixed quantity is added to a given address. An index register "X" as well as the A, B, and S (stack) registers may be used for storing the indexing quantity. The S register is used for push-pop stack operations as well as fully recursive reentry procedures. The recursive procedure is essentially an indexing technique which is performed inde-

pendently of and in addition to the indexing in the effective address calculation involving X, A or B.

1.2 MEMORY

The main memory is addressed as a set of contiguous word locations whose addresses range from 0 to '077777 or 32,767. (NOTE: throughout the remainder of this text, an apostrophe preceding a number indicates octal notation; no apostrophe indicates decimal notation. Memory locations are always specified by their octal addresses.) The number of words that can be addressed by an instruction, and the location of those words relative to the instruction depend on which of two addressing modes — sectored or relative — the machine is operating in. In either mode, contiguous word locations are organized into fixed-length groups called sectors.

Sectored and Relative Addressing Modes

In sectored mode addressing, all sectors are 512 words long and an instruction may directly address either the locations in sector 0 (locations 0-'777) or the locations in the sector in which the instruction is stored. Relative mode addressing permits direct references to locations in sector zero, as in sectored mode, or references to locations in a range relative to the contents of the program counter P (P-239 to P+256). Sixteen unused addresses from P-240 to P-256 are interpreted as special addressing codes which provide several additional methods of address formation such as stack register operation, base-plus-displacement and direct addressing of any location from 0 to '077777.

Automatic Memory Refresh

The computer's semiconductor memory is continually refreshed by a sequence of staggered refresh cycles, each of which refreshes 1/32 of the entire memory. Although refreshing does take some time from the program, the effect is usually negligible as the microprogrammed processor logic continues in operation while the refreshing is in progress.

High Speed Register File

The various processor registers discussed in the previous section and a number of other special processor registers can be addressed as memory locations although they are physically located in a high-speed register file within the CPU. Memory addresses 0-'37 are reserved for this purpose, and correspond to the registers in the register file as follows:

0	X
1	A
2	B
3	S
4-5	Reserved
6	Normalize shift count
7	P
10-13	Reserved
14-17	Reserved for microprogram
20-37	Word pairs for DMA channels 1-8 (address and word counts)

Reserved Memory Locations

Certain locations in sector zero are reserved for specific purposes. Location '61 is used as a counter by the real-time clock. Locations '60 and '62-'73 are used for interrupts; both internal (such as memory parity errors and illegal instructions) or external (peripheral device interrupts). Locations '100-'177 are set aside for vectored interrupts. (The location used for a particular interrupt is '100 plus the code of the device causing the interrupt.) All of the simpler device controllers work in this way while some of the more complex controllers are set up to interrupt to any desired location in the first 4K words of memory.

1.3 NUMBER AND INSTRUCTION FORMATS

To perform logical operations the hardware interprets operands as logical words. For arithmetic, the hardware operates on 16-bit unsigned numbers or signed numbers in twos complement notation. A 16-bit unsigned number is usually regarded as an integer and hence has a range of 0 to

2^{16}-1. In a signed number, bit 1 represents the sign (0 for plus, 1 for minus) and bits 2-15 represent the magnitude in twos complement notation. Signed numbers are generally regarded as having an arbitrary binary point, which the computer does not keep track of; the programmer must adopt a point convention and shift the magnitude of the result to conform to the convention used.

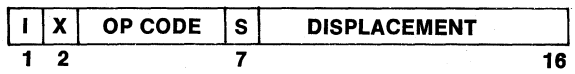
Two common conventions are to regard a number as an integer (binary point at the right) or as a proper fraction (binary point at the left); in these two cases the range of signed numbers represented by a single word is -2^{15} to 2^{15}-1, or -1 to $1-2^{-15}$.

For instructions that operate on double precision numbers, the high order word has the usual format, and the low order word has a 0 in bit 1 and a 15-bit low-order extension of the number in bits 2-16.

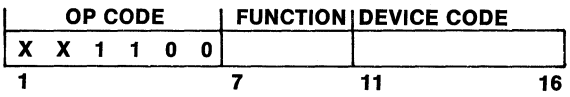
Instruction Groups

The instruction set can be conveniently divided into four major groups: generic, shift, I/O and memory-reference. In the generic group, the entire instruction word is treated as an op code. Instructions in the shift group consist of an op code plus a field which indicates the number of places to be shifted.

All memory reference instructions appear in the format shown below and consist of an op code plus additional fields of information which are used to develop an address for referencing memory. Bits 1 and 2 indicate how indirection and indexing will be used for address modification. The sector bit specifies how the displacement will be interpreted. If zero, the sector bit specifies that the displacement is the address of a location in sector zero. A one in the sector bit specifies that the displacement references a location in the sector in which the instruction is stored if the machine is in sectored mode, or a location in the range of P-239 to P+256 if in relative mode.



The format for I/O instructions is shown below. Bits 1 and 2 of the op code select among four subclasses of instructions for sending out control information, sensing



conditions in a device, or moving data in or out. Bits 7-10 specify the particular function within the subclass, and that function takes place using the device specified by bits 11-16. For example, a high-speed device that transfers data through a DMA channel would have one function in the output class for sending an interrupt address to the device, another for sending a word count, and so on.

1.4 PROGRAMMING CONVENTIONS

The 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 op codes (Appendix C).

For example, the mnemonic

```
LDA
```

assembles as '004000, and

```
LDA 3
```

assembles as '004003. The latter word, when executed as an instruction, loads the contents of memory location '3 (the stack register S) into the A register.

Free-Form Source Statements

The program in symbolic language for assembly is made up of source statements, each containing up to four variable length fields separated by spaces or tabs. The sequence of fields from left to right in a source statement line is label, operation, address, and comment. The operation field contains the op code or its mnemonic, and the address field contains the address used by a memory

reference instruction. The example above contains only operation and address fields. For other types of instructions the address field is used to specify whatever bits are not included in the op code; *e.g.*, the function and device code in an I/O instruction and the number of shifts in a shift instruction. In the example given above the number in the address field assembles directly into the displacement part of the instruction word because the location addressed is in sector 0 and the number has only one digit. If the instruction was written as

```
LDA 13
```

the assembler would generate '004015, because it interprets all unqualified numbers as decimal. On the other hand,

```
LDA '13
```

would assemble as '004013 and would actually access location '13.

An asterisk appended to an op code mnemonic indicates indirect addressing. For example,

```
LDA* '13
```

assembles '104013, and produces indirect addressing. Placing ",1" following the memory address causes modification of the address by the contents of the currently specified index register. Hence

```
LDA* '13,1
```

which assembles as '144013 and, depending on the addressing mode, the processor either indexes the initial address and then continues the effective address calculation, or post indexes the result.

In the above examples, addressing is in sector 0 so the displacement is equivalent to the address given. But the programmer can give any address in the available memory space; e.g., to load A from location '4000 the programmer would give

```
LDA '4000
```

whose assembled form would depend upon the current addressing mode and where location '4000 is in relation to the position of the instruction. In other words the programmer can give any address, and the assembler and loader together set up whatever effective address calculation is necessary to access the desired location.

Symbolic Addressing

Ordinarily the programmer dispenses with keeping track of numbers and uses symbolic addressing. One way to define a symbolic address is through use of the label field.

```
Q ADD '20
```

indicates that the location containing ADD '20 may be addressed symbolically as Q. Additional conventions for symbolic addressing are described in the PRIME 200 Macro Assembler manual.

1.5 SOFTWARE

To support its computers, Prime supplies a very extensive software package. This package includes assembler, link-loader, compiler, editor, input-output control system, and several operating systems, as well as numerous utility programs for various devices, debugging programs, data conversion, mathematical and interpretive routines, and a complete set of hardware verification and maintenance routines. Some of the major software items are the following.

The *Prime Macro Assembler (PMA)* includes not only all of the features one would expect in a well-designed symbolic assembler—pseudo-ops for assembly, listing and loader control; symbol and data definitions; storage allocation; program linking; conditional assembly—but also a means of generating application-oriented macro statements that can significantly simplify application programming.

With the *Desectorizing Link-Loader (LDR)*, relocatable or absolute program modules from any Prime translator can be loaded, linked and bound, and cross-sector references are automatically desectorized. Loader tables are preserved to provide for symbolic debugging; furthermore the Disk

Operating System can be used to save and restore partial loads.

The *Text Editor (ED)* is a full context editor with such features as line-by-line and character-by-character editing, and automatic execution of string buffers for multiple changes of the same text throughout a program. A bulk media converter, FILED, provides editing and transcription for large volumes of data.

The *Debugging Package (TAP)* includes an interactive trace routine for examining, patching, tracing, and so forth. Symbolic references to memory are permitted through the loader symbol table.

Extended FORTRAN IV (FRTN4), which is the basic Prime system programming language, is an implementation of ANSI FORTRAN IV with other extensions derived from the proposed Instrument Society of America FORTRAN for control applications. Significant extensions include embedded logical functions such as *AND, OR, NOT* and exclusive *OR*; run-time trace; octal constants; and an extended FORTRAN library.

The *Input-Output Control System (IOCS)* provides stream control routines and device drivers that make user programs device independent and centralize I/O control for all system software. A unique IOCS feature is the provision for source file editing and merging.

The *Disk Operating System (DOS)* is an interactive controller and file-handling system. It provides an extensive array of features to simplify program development and maintenance, including multiple-level file directories and multiple volume-control and file-access methods to simplify the creation, deletion and updating of source, object and data files; automatic job and data stream routing for batch processing; name addressing for files; and support of compressed ASCII files giving 5:1 character reduction for assembly language files and 4:1 reduction for FORTRAN files.

The *Real Time Operating System (RTOS)* is a compact multiprogramming system that schedules the three basic computer resources—processing power, main memory, mass memory—to control the execution of application programs in a real time environment. It provides interrupt handling, multiprogram scheduling, simultaneous input-output, and general supervisory functions. RTOS is available in two versions: a memory-only system for fast response applications, and a memory-disk system for multitask applications involving resource sharing, memory mapping, and large data bases.

RTOS and DOS use the same file management system, allowing maintenance of files off line. The Disk Operating System can run under control of the RTOS, and conversely, real time systems can be developed under DOS control.

The *Stand Alone Operating System* has exactly the same translators, loaders and IO controllers as the larger operating system configurations.

CHAPTER 2
CENTRAL PROCESSOR

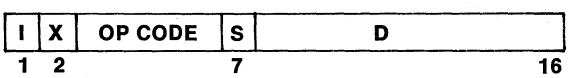
This chapter describes all computer instructions in detail and discusses the general effects of I/O instructions on processor elements including the control panel, priority interrupt system, real-time clock, and power monitor. Effects of I/O instructions on particular peripheral devices are described in the chapters pertaining to those devices.

For easy reference, all instruction descriptions are presented in the following format: mnemonic and instruction name at top left, and octal op code at top right over a box showing the binary word into which the mnemonic is assembled by the PRIME 200 Macro Assembler. The following letters are used in various instruction words to indicate variable information:

I Indirect Addressing Bit
X Indexed Addressing Bit
S Sector Bit
D Address Displacement

2.1 MEMORY REFERENCE

This section explains the procedure and instructions used to calculate the effective address of all memory reference instructions. The program controls the effective address calculation not only by the information given in instruction and address words, but also by selecting the register to be used for indexing (X, A, B, or S) and by specifying the addressing mode. The mode determines both the type of addressing and the size of the address space. Bits 1, 2, and 7-16 have the same format in every memory reference instruction whether the effective address is used for storage or retrieval of an operand or to alter program flow. (NOTE: in an instruction that loads or stores the index register, what would otherwise be the X bit is used instead as part of the op code). Bit 1 is the indirect or I bit, bit 2 is the index or X bit, bit 7 is the sector bit, and bits 8-16 are the displacement. The effective address of the instruction depends on the values of I,X,S, and D.



In general, an effective address, EA, is formed by adding a base address to a displacement field D and then conditionally adding the contents of an index register. If the S bit is zero, the base address is also zero. If S is one, the base address is specified by bits in the program counter: bits 2-7 if the machine is in sectored mode, bits 2-16 if in relative mode.

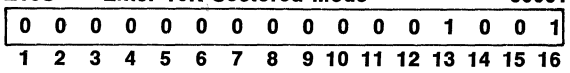
The base plus displacement plus conditional indexing calculation produces an effective memory address if I is zero, or an intermediate address if I is one (specifying indirect addressing). The intermediate address word may, depending on the addressing mode, also contain X and I bits and is processed in a manner similar to the original instruction word. Any number of levels of indirect addressing are permitted; the process continues until a location is found with a zero in the I bit. Note that while address computations are done in 16 bits, any address actually used to access memory is truncated to the number of bits appropriate to the specified addressing space (bits 3-16 for 16K, bits 2-16 for 32K).

Additional address formation flexibility is provided by dividing sectored and relative modes into three operating catagories: 16K sectored, 32K sectored and 32K relative. The 16K sectored mode (abbreviated as "16S" mode) is considered to be the normal mode of operation. When the machine is turned on or the computer is cleared from the control panel, the processor is automatically set up for 16S mode and all indexing is done using the index register X. In this mode, indexing may occur both before and after indirect references since an absolute address requires only 14 bits—leaving room in address words for both I and X bits. Note that when operating in this mode, effective addresses reference the first 16K of memory.

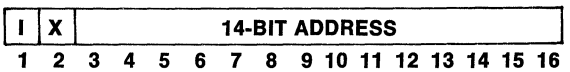
The sectored and relative modes for 32K (32S and 32R) extend the addressing range to 32K by using 15 bits in an address word to represent an address. Such address words have no X bit. Therefore, except for a special case, indexing must be done after one or more levels of indirect addressing has been performed. The special case permits indexing to occur prior to any memory access if the sector bit is zero and the displacement is less than '100.

The following instructions are used to change addressing modes or alter the source of indexing information.

E16S Enter 16K Sectored Mode '000011

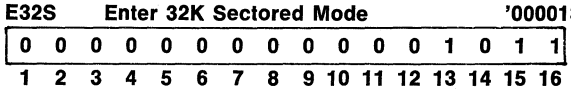


In subsequent effective address calculations use absolute sectors with $0 \leq D \leq$ '777 and interpret address words this way:

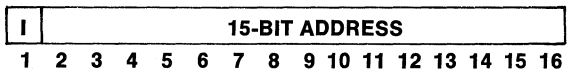


Indexing may be performed before or after indirect references.

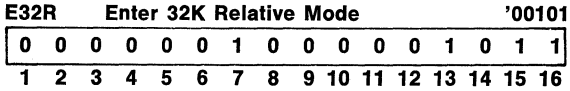
E32S Enter 32K Sectored Mode '000013



In subsequent effective address calculations use absolute sectors with $0 \leq D \leq$ '777, use postindexing when $S = 1$ or $D \geq$ '100 and $S = 0$, and interpret address word this way:



E32R Enter 32K Relative Mode '001013



In subsequent effective address calculations use relative sectors with $-240 \leq D \leq 255$ when $S = 1$, use postindexing when $S = 1$ or $D \geq$ '100 and $S = 0$, and interpret address words this way:

The following table lists the address word configurations and the calculation procedures for all cases in each mode. P is the contents of the program counter *prior* to the instruction fetch, the symbol PD represents the sectored address formed by concatenation, X is the contents of the currently selected index register, A is an absolute address, and I (ξ) is the result of the indirect chain beginning with access to the location addressed by ξ.

16K Sectored. $0 \leqslant D \leqslant$ '777

I	X	S	D	Address Word	EA
0	0	0			D
0	1	0			D + X
1	0	0		I, X, A	I(D)
1	1	0		I, X, A	I(D + X)
0	0	1			P|D
0	1	1			P|D + X
1	0	1		I, X, A	I(P|D)
1	1	1		I, X, A	I(P|D + X)

32K Sectored. $0 \leqslant D \leqslant$ '777

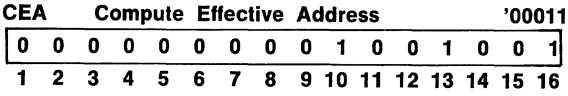
0	0	0			D
0	1	0			D + X
1	0	0		I, A	I(D)
1	1	0	$<$ '100	I, A	I(D + X)
1	1	0	$\geqslant$ '100	I, A	I(D) + X
0	0	1			P|D
0	1	1			P|D + X
1	0	1		I, A	I(P|D)
1	1	1		I, A	I(P|D) + X

32K Relative. $S = 0$: $0 \leqslant D \leqslant$ '777
$S = 1$: $-256 \leqslant D \leqslant 255$
(for $D < -240$ see § 2.9)

0	0	0			D
0	1	0			D + X
1	0	0		I, A	I(D)
1	1	0	$<$ '100	I, A	I(D + X)
1	1	0	$\geqslant$ '100	I, A	I(D) + X
0	0	1	$\geqslant -240$		P + 1 + D
0	1	1	$\geqslant -240$		P + 1 + D + X
1	0	1	$\geqslant -240$	I, A	I(P + 1 + D)
1	1	1	$\geqslant -240$	I, A	I(P + 1 + D) + X

In the description of any memory reference instruction, EA refers to the result of the effective address calculation, truncated to the appropriate number of bits. There is also an instruction that allows the program to calculate an effective address from an address word (not from an instruction word). This is used by the software in conjunction with the FORTRAN attach-arguments subroutine, whose calling sequence supplies address words for the arguments instead of the arguments themselves; thus the software can determine the positions of the arguments.

CEA Compute Effective Address '000111

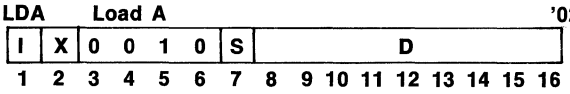


Calculate the effective address indicated by the contents of A interpreted as an address word in the current addressing mode, and place the result in A.

Load and Store Instructions

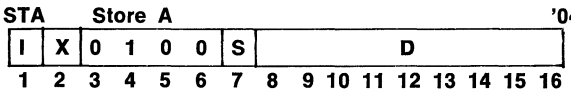
There are five instructions for moving data between memory and the A and index registers.

LDA Load A '02



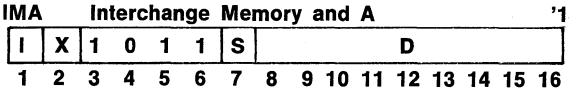
Load the contents of location EA into A. The contents of EA are unaffected, the original contents of A are lost.

STA Store A '04



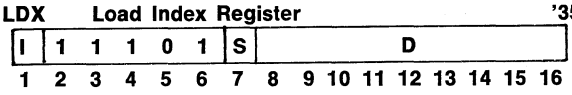
Store the contents of A in location EA. The contents of A are unaffected, the original contents of EA are lost.

IMA Interchange Memory and A '13



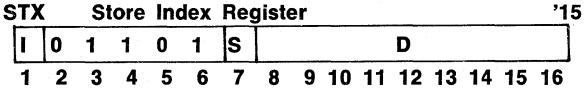
Store the contents of A in location EA and load the original contents of location EA into A.

LDX Load Index Register '35



Load the contents of location EA into the register currently selected for indexing. The contents of EA are unaffected, the original contents of the index register are lost. Note that this instruction cannot itself specify indexing, although an address word retrieved in the effective address calculation may do so.

STX Store Index Register '15



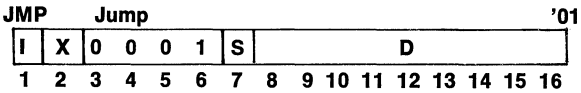
Store the contents of the register currently selected for indexing into location EA. The contents of the index register are unaffected, the original contents of EA are lost. Note that this instruction cannot itself specify indexing, although an address word retrieved in the effective address calculation may do so.

Jump Instructions

These two instructions allow the programmer to alter the normal program sequence by jumping to an arbitrary location.

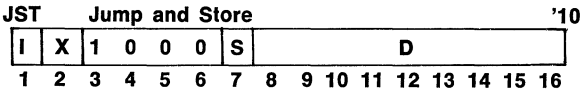
Jump Instructions

JMP Jump '01



Load EA into P. Take the next instruction from location EA and continue sequential operation from there.

JST Jump and Store '10



Store an address one greater than that in P into location EA (hence location EA receives the address of the location following the JST instruction). Load EA + 1 into P. Take the next instruction from location EA + 1 and continue sequential operation from there.

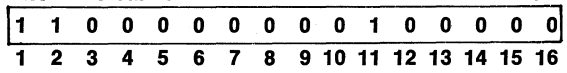
The address stored is truncated according to the mode (16K: bits 3-16, 32K: bits 2-16) and storage does not affect the other memory bits. Hence a location previously set up with I or I and X bits for use as an address word is ready as soon as the address is stored in it.

The usual procedure for calling a subroutine is to give a JST whose effective address is the starting location of the routine. Since P+ 1 is saved at the entry point, a subsequent return can be made to the location following the JST simply by giving a JMP that addresses the entry point indirectly.

2.2 REGISTER OPERATE

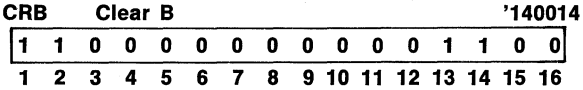
These instructions are simply for clearing the A and B registers and moving data between them.

CRA Clear A '140040



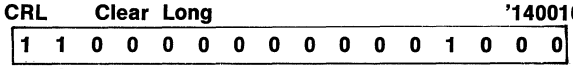
Change the contents of A to all 0s.

CRB Clear B '140014



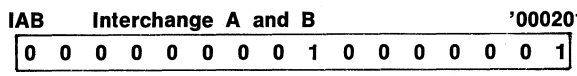
Change the contents of B to all 0s.

CRL Clear Long '140010



Clear A and B.

IAB Interchange A and B '000201

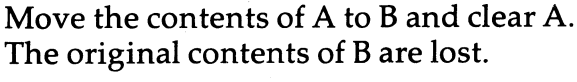


Move the contents of A to B and the contents of B to A.

XCA Transfer and Clear A '140104

Move the contents of A to B and clear A. The original contents of B are lost.

XCB Transfer and Clear B '140204

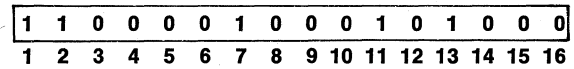


Move the contents of B to A and clear B. The original contents of A are lost.

2.3 BYTE MANIPULATION

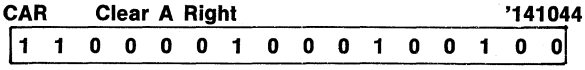
These instructions are for manipulating half words in A. They are useful for handling ASCII characters, 8-bit data bytes packed two to a word in memory, tables where half of each table location is used for the entry and the other half for a label, etc.

CAL Clear A Left '141050



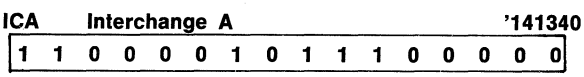
Clear A bits 1-8 without affecting bits 9-16.

CAR Clear A Right '141044



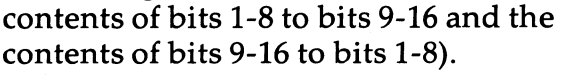
Clear A bits 9-16 without affecting bits 1-8.

ICA Interchange A '141340



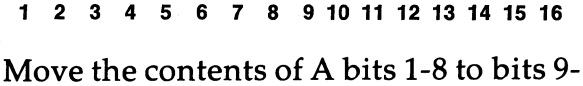
Interchange the two halves of A (move the contents of bits 1-8 to bits 9-16 and the contents of bits 9-16 to bits 1-8).

ICL Interchange and Clear Left '141140



Move the contents of A bits 1-8 to bits 9-16 and clear bits 1-8. The original contents of bits 9-16 are lost.

ICR Interchange and Clear Right '141240



Move the contents of A bits 9-16 to bits 1-8 and clear bits 9-16. The original contents of bits 1-8 are lost.

2.4 SHIFT GROUP

Shifting is the movement of the contents of a register bit-to-bit. The instructions in this group shift or rotate right or left the contents of A or the contents of A and B treated as a single register with A on the left. Although these instructions are similar in format and operation, functionally some are logical and others arithmetic, so they also belong to one or the other of the categories discussed in the next two sections.

A shift is logical or arithmetic simply in terms of the way the data word is interpreted: a logical shift treats it as a

logical word, whereas an arithmetic shift treats it as a signed number. In a logical shift the contents of the register or registers are moved bit-to-bit with 0s brought in at the end being vacated and information shifted out at the other end is lost. Rotation is a cyclic shift such that information rotated out at one end is put back in at the other.

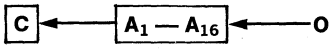
A right arithmetic shift fills the vacated left positions with the contents of the sign bit and does not change the sign. A left arithmetic shift includes the sign (A bit 1 only—B bit 1 is left out), but interprets a sign change as overflow and fills the vacated right positions with 0's. Hence arithmetic shifting is equivalent to multiplying the number by a power of 2 provided no information is lost. These operations also use the C bit to detect the loss of any bit of significance in a left arithmetic shift, and in all other cases to save the last bit shifted out.

In a shift instruction word bits 3-6 are all 0's and the group is indicated by 01 in bits 1 and 2. Bits 7-10 indicate the particular type of shift, and bits 11-16 specify the twos complement of the number of places to be shifted. Mnemonics are available for the individual types, so the op code may be regarded as the left four digits of the instruction word, with the word completed by adding the right two digits for the number of places. Note that the mnemonics are constructed using "logical" to mean a logical shift and "shift" to mean specifically an arithmetic shift.

ALL A Left Logical '0414

0	1	0	0	0	0	1	1	0	0	—N					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Shift the contents of A left *N* places, bringing 0s into bit 16; data shifted out of bit 1 is lost, except that the last bit shifted out is saved in C.

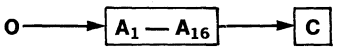


Note: The assembler recognizes ALL and LGL as equivalent.

ARL A Right Logical '0404

0	1	0	0	0	0	0	1	0	0	—N					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Shift the contents of A right *N* places, bring 0s into bit 1; data shifted out of bit 16 is lost, except that the last bit shifted out is saved in C.

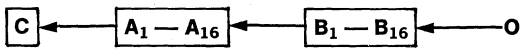


Note: The assembler recognizes ARL and LGR as equivalent.

LLL Long Left Logical '0410

0	1	0	0	0	0	1	0	0	0	—N					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

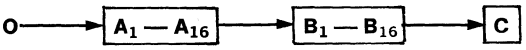
Shift the contents of A and B left *N* places, bringing 0s into B bit 16; B bit 1 is shifted into A bit 16; data shifted out of A bit 1 is lost, except that the last bit shifted out is saved in C.



LRL Long Right Logical '0400

0	1	0	0	0	0	0	0	0	0	—N					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

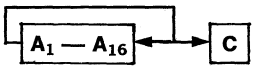
Shift the contents of A and B right *N* places, bringing 0s into A bit 1; A bit 16 is shifted into B bit 1; data shifted out of B bit 16 is lost, except that the last bit shifted out is saved in C.



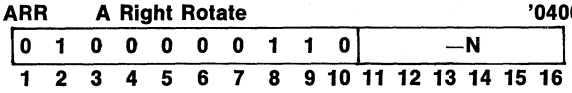
ALR A Left Rotate '0416

0	1	0	0	0	0	1	1	1	0	—N					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

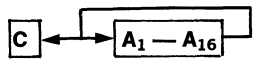
Shift the contents of A left *N* places, rotating bit 1 into bit 16. The last bit rotated back in at the right is also saved in C.



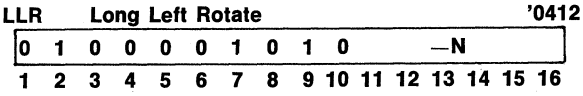
ARR A Right Rotate '0406



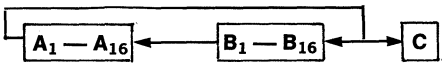
Shift the contents of A right *N* places, rotating bit 16 into bit 1. The last bit rotated back in at the left is also saved in C.



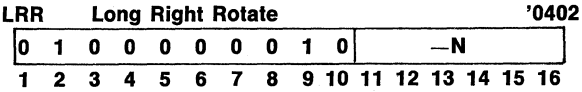
LLR Long Left Rotate '0412



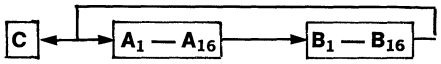
Shift the contents of A and B left *N* places, rotating A bit 1 into B bit 16; B bit 1 is shifted into A bit 16. The last bit rotated from A back to B is also saved in C.



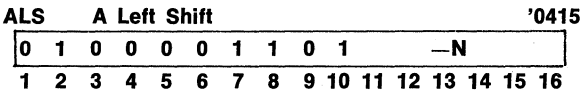
LRR Long Right Rotate '0402



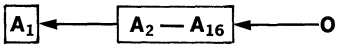
Shift the contents of A and B right *N* places, rotating B bit 16 into A bit 1; A bit 16 is shifted into B bit 1. The last bit rotated from B back to A is also saved in C.



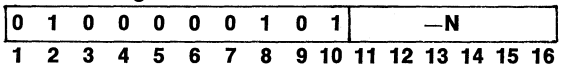
ALS A Left Shift '0415



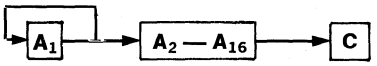
Shift the contents of A left arithmetically *N* places, bringing 0s into bit 16; data shifted out of bit 1 is lost. If the sign (bit 1) changes state, set C; otherwise reset it. A sign change indicates that a bit of significance (a 1 in a positive number, a 0 in a negative) has been shifted out of the magnitude part.



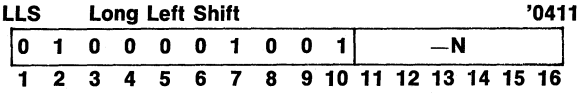
ARS A Right Shift '0405



Shift the contents of A right arithmetically *N* places, leaving the sign (bit 1) unaffected, but shifting it into the magnitude part (0s in a positive number, 1s in a negative); data shifted out of bit 16 is lost, except that the last bit shifted out is saved in C.

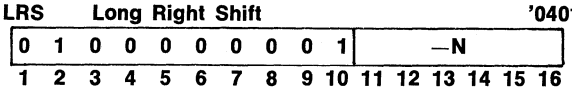


LLS Long Left Shift '0411

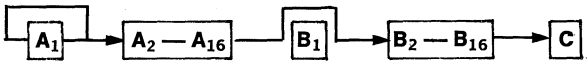


Shift the contents of A and B left arithmetically *N* places, bringing 0s into B bit 16 and bypassing B bit 1; B bit 2 is shifted into A bit 16; data shifted out of A bit 1 is lost. If the sign (A bit 1) changes state, set C; otherwise reset it. A sign change indicates that a bit of significance (a 1 in a positive number, a 0 in a negative) has been shifted out of the magnitude part.

LRS Long Right Shift '0401



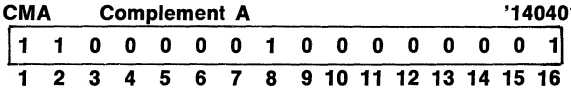
Shift the contents of A and B right arithmetically *N* places, leaving A bit 1 unaffected and bypassing B bit 1, but shifting the sign (A bit 1) into the magnitude part (0s in a positive number, 1s in a negative); A bit 16 is shifted into B bit 2; data shifted out of B bit 16 is lost, except that the last bit shifted out is saved in C.



2.5 LOGIC

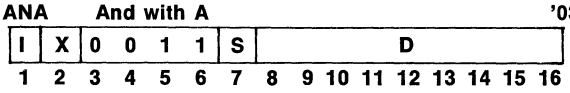
Besides the logical shift and rotate instructions described in the preceding section, the PRIME 200 repertoire includes instructions for performing the complement, AND, and exclusive OR functions (the latter two being memory reference), and a group of instructions that "logicize" numbers. A number is logicized by replacing it with a truth value that indicates the result of a comparison between the number and zero.

CMA Complement A '140401



Form the (logical) complement of the contents of A in A (replace all 1s in A with 0s, all 0s with 1s).

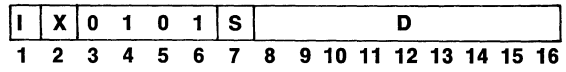
ANA And with A '03



Form the AND function of the contents of location EA with the contents of A and place the result in A. A given bit of the result is 1 if the corresponding bits of both operands are 1; otherwise the resulting bit is 0.

A Bit	Memory Bit	Resulting Bit
0	0	0
0	1	0
1	0	0
1	1	1

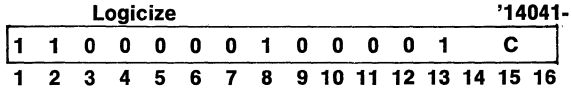
ERA Exclusive Or with A '05



Form the exclusive OR function of the contents of location EA with the contents of A and place the result in A. A given bit of the result is 1 if the corresponding bits of the operands differ; otherwise the resulting bit is 0.

A Bit	Memory Bit	Resulting Bit
0	0	0
0	1	1
1	0	1
1	1	0

Logicize '14041-



If the number in A satisfies the conditionC, replace it with 1; *i.e.*, with a 1 in bit 16 and 0s elsewhere; otherwise clear A. The conditions selected by bits 14-16 and the mnemonics and op codes for them are as follows.

Mnemonic	Bits 14-16	Condition	Op Code
LLT	0	<0	'140410
LLE	1	$\leqslant 0$	'140411
LNE	2	$\neq 0$	'140412
LEQ	3	$=0$	'140413
LGE	4	$\geqslant 0$	'140414
LGT	5	>0	'140415
LF	6	False*	'140416
LT	7	True*	'140417

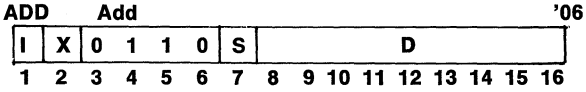
*These two instructions do not specify any condition: LF and LT simply set A to the truth values 0 and 1 respectively.

2.6 ARITHMETIC

The computer has memory reference instructions for performing addition, subtraction, multiplication and division of numbers in fixed point format [Appendix C]. The add and subtract instructions can operate on either single length or double length numbers; multiply supplies a double length product, and divide uses a double length dividend. The high and low order words respectively of a double length number are in registers A and B or memory locations EA and EA + 1, where the magnitude part is the 30-bit string in bits 2-16 of the two words; the sign is in bit 1 of the high order word, and bit 1 of the low order word should be 0.

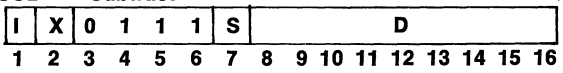
Besides the instructions for the basic operations and the arithmetic shift instructions described in §2.4, there are also instructions for manipulating signs, incrementing and decrementing numbers, negating, manipulating operands to allow multiplication and division using single length integers exclusively, and even instructions for facilitating floating point arithmetic. With many of the instructions the C bit detects overflow; *i.e.*, the condition where the magnitude of a number is larger than can be accommodated. In division, C indicates when the division process cannot generate a meaningful result.

ADD Add '06



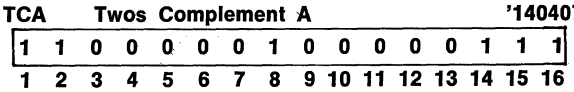
Add the contents of memory location EA to the contents of A and place the result in A. If the sum is $\geqslant 2^{15}$ or $\leqslant -2^{15}$, set C; otherwise reset it. In the first overflow case the result has a minus sign but a magnitude in positive form equal to the sum less 2^{15}; in in the second the result has a plus sign but a magnitude in negative form equal to the sum plus 2^{15}.

SUB Subtract '07



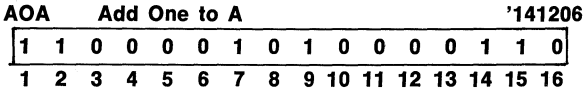
Subtract the contents of location EA from the contents of A and place the result in A. If the difference is $\geq 2^{15}$ or $< -2^{15}$, set C; otherwise reset it. In the first overflow case the result has a minus sign but a magnitude in positive form equal to the difference less 2^{15}; in the second the result has a plus sign but a magnitude in negative form equal to the difference plus 2^{15}.

TCA Twos Complement A '140407



Form the twos complement negative of the contents of A in A. If the number negated is -2^{15}, set C and give a result of -2^{15}; otherwise reset C.

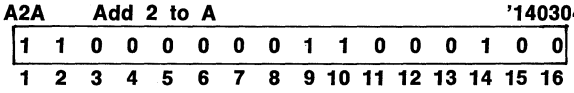
AOA Add One to A '141206



Add 1 to the contents of A and place the result in A. If the number incremented is 2^{15}-1, set C and give a result of -2^{15}; otherwise reset C.

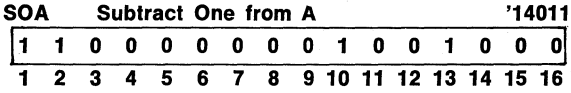
Note: The assembler recognizes the mnemonic A1A as equivalent to A0A.

A2A Add 2 to A '140304



Add 2 to the contents of A in A. If the number incremented is 2^{15}-2 or 2^{15}-1, set C and give a result of -2^{15} or $-(2^{15}-1)$; otherwise reset C.

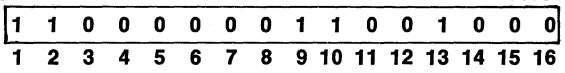
SOA Subtract One from A '140110



Subtract 1 from the contents of A in A. If the number decremented is -2^{15}, set C and give a result of 2^{15}-1; otherwise reset C.

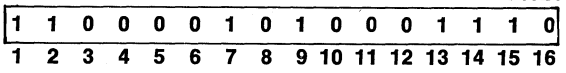
Note: The assembler recognizes the mnemonic S1A as equivalent to S0A.

S2A Subtract 2 from A '140310



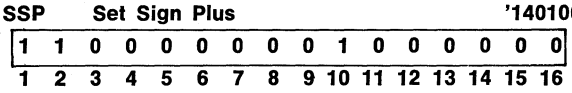
Subtract 2 from the contents of A in A. If the number decremented is $-(2^{15}-1)$ or -2^{15}, set C and give a result of 2^{15}-1 or 2^{15}-2; otherwise reset C.

ACA Add C to A '141216



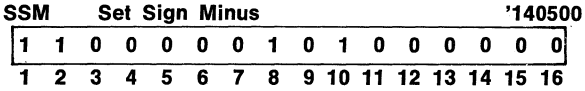
Add the contents of C to the contents of A in A (C is taken as being of the same order of magnitude as A bit 16). If the number originally in A is 2^{15}-1, set C and give a result of -2^{15}; otherwise clear C.

SSP Set Sign Plus '140100



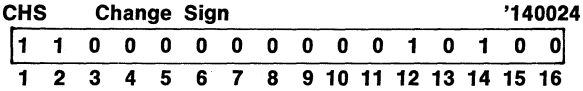
Reset A bit 1 without affecting the rest of the register.

SSM Set Sign Minus '140500



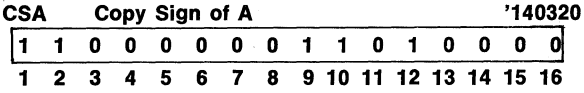
Set A bit 1 without affecting the rest of the register.

CHS Change Sign '140024



Complement A bit 1 without affecting the rest of the register.

CSA Copy Sign of A '140320



Make C equal to A bit 1, and reset A bit 1 (plus) without affecting the rest of the register.

Double Precision Arithmetic

The computer does not actually have separate instructions for handling double length operands—it has single and double

precision modes of operation. When power is turned on or the computer is cleared from the control panel, the processor is automatically in single precision mode, but the program can change the mode. When the processor is in double precision mode, the instructions that ordinarily load, store, add and subtract single length numbers instead operate on double length numbers. The op codes for these memory reference instructions are therefore the same as for their single precision counterparts, but the assembler recognizes unique mnemonics for them to facilitate clarity of documentation.

DBL Double Precision '000007

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Enter double precision mode so that subsequently every LDA, STA, ADD or SUB instruction handles double length operands *(i.e.,* is executed respectively as a DLD, DST, DAD or DSB as described below).

SGL Single Precision '000005

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Return to single precision mode so that subsequently any LDA, STA, ADD or SUB instruction handles single precision operands.

DLD Double Load '02

I	X	0	0	1	0	S	D								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Load the contents of location EA into A and location EA + 1 into B. The contents of memory are unaffected, the original contents of A and B are lost.

DST Double Store '04

I	X	0	1	0	0	S	D								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Store the contents of A in location EA and the contents of B in location EA + 1. The contents of A and B are unaffected, the original contents of the specified memory locations are lost.

DAD Double Add '06

I	X	0	1	1	0	S	D								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Add the double length contents of locations EA and EA + 1 to the double length contents of A and B, and place the result in A and B. If the sum is $\geq 2^{30}$ or $< -2^{30}$ set C; otherwise reset it. In the first overflow case the result has a minus sign but a magnitude in positive form equal to the sum less 2^{30}; in the second the result has a plus sign but a magnitude in negative form equal to the sum plus 2^{30}.

By definition, bit 1 of the low order part of a double precision number must be 0. However, this instruction produces a correct result as long as the sign bits (B_1 and bit 1 of EA + 1) are the same.

DSB Double Subtract '07

I	X	0	1	1	1	S	D								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Subtract the double length contents of locations EA and EA + 1 from the double length contents of A and B, and place the result in A and B. If the difference is $\geq 2^{30}$ or $< -2^{30}$, set C; otherwise reset it. In the first overflow case the result has a minus sign but a magnitude in positive form equal to the difference less 2^{30}; in the second the result has a plus sign but a magnitude in negative form equal to the difference plus 2^{30}.

Although bit 1 of the low order part of a double precision number should be 0, this instruction does produce a correct result if the sign bits of the low order parts are both 1. However the result is invalid if the low order sign bits are not the same.

To negate a double length number simply subtract it from zero. In a computer that lacks hardware for double precision, the single length instructions previously defined can be used to implement it. To

negate, take the twos complement of the low order part, but take the ones complement of the high order part unless the low order part is null, in which case twos complementing must be used for the high order part as well.

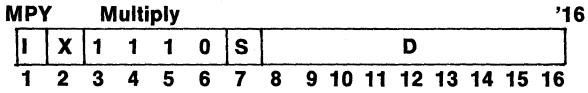
Suppose the high and low order parts of one double length operand are in locations AH and AL and another operand is similarly in BH and BL. The following sequence places the double length sum in CH and CL with appropriate compensation for any overflow from the low order addition.

```
LDA     AL      Get one low order part
ADD     BL      Add the other
CSA             Put carry in C, clear sign
STA     CL      Store low sum
LDA     AH      Get one high part
ACA             Add carry from low addition
ADD     BH      Add other high part
STA     CH      Store high sum
```

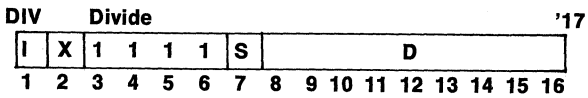
Of course SSP could just as well be substituted for the CSA, as C already contains the low overflow.

Multiply-Divide

The PRIME 200 has two basic instructions for performing multiplication and division of fixed point numbers. As previously mentioned, multiplication produces a double length product and division uses a double length dividend.



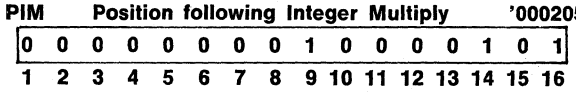
Multiply the contents of A by the contents of location EA, and place the double length result in A and B.



If the absolute value of the number in location EA is less than the absolute value of the number in A (taking both numbers as representing the same order of magnitude), set C; otherwise clear it. Then divide the double length contents of A and B by the contents of location EA, calculating a quotient of fifteen magnitude bits including leading zeros. Place the quotient in A and the remainder with the sign of the original dividend in B. The results in A and B are the correct quotient and remainder provided C is *not* set; otherwise they are meaningless.

The overflow condition for division requires that the quotient be a proper fraction or a single length integer. With fractions, the bits of the high order parts of dividend and divisor are of the same order of magnitude; hence the condition is that the divisor be larger in magnitude and the answer less than unity. With integers, the overflow test effectively treats the dividend as though the binary point were between the high and low parts, so that the actual dividend (with the binary point at the right end of B) is guaranteed to be greater than the divisor by no more than fifteen binary orders of magnitude, and hence the integral quotient will fit in one register. If the initial test is not satisfied, there is simply no way to determine the true position of the binary point in the result. Of course the program would compensate for this by shifting the operands and keeping track of the number of shifts (*i.e.*, the change in order of magnitude) required to produce a meaningful division.

As given above, the instructions are somewhat cumbersome for working entirely with single-length integers. In a multiplication of small integers, the significant bits of the result are all in B, whereas the sign is in A. Similarly it would be convenient to be able say to divide 15 by 3 and get an answer of 5 without having to use pairs of locations to hold the numbers. The following two instructions facilitate such operations.



Move the contents of A bits 2-16 to B bits 2-16 and reset B bit 1. Fill A bits 2-16 with the sign of A.

PID Position for Integer Divide '000211

0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Move the contents of B bits 2-16 into A bits 2-16. Clear B. The original contents of A bits 2-16 are lost, but A bit 1 is unaffected.

The first of these instructions is used following MPY to reduce the product to single length. However if there are more than fifteen significant bits in the product, the high order bits are lost. If there is any chance that the integers multiplied will produce a product larger than one word, the program should include a test to make sure A bits 2-16 are all null before giving the PIM.

The PID allows the programmer to use a single length dividend and guarantees the division to be meaningful (producing the integral part of the quotient) except in the obvious case of a zero divisor. Effectively the PID multiplies the given dividend by 2^{-15} so that the divisor is bound (unless it is zero) to satisfy the condition that it be greater in magnitude than the high order part in A. The result of a subsequent DIV is thus actually a proper fraction, which is multiplied by 2^{15} simply by interpreting it as an integer.

Floating Point Arithmetic

Operations in floating point format must be done using software. The standard format for floating point numbers is that in the high order word, bit 1 is the sign, but the rest of the word contains an 8-bit exponent and the seven high order bits of a fixed point fraction. For a positive number the sign is 0, as in fixed point format. But the contents of bits 2-9 are an integral exponent in excess 128 ('200) code. Exponents from -128 to +127 are therefore represented by the binary equivalents of 0 to 255 (0-'377). The contents of bits 10-16 are interpreted only as the high order part of a proper fraction, and additional words

Sign	Exponent	Fraction
1 2	9	10 16

0	Low Order Extension of Fraction
1	2 16

may be used to extend the precision of the fraction as desired. Floating point zero and negatives are represented in the same way as in fixed point: zero by a word containing all 0s (hence having the smallest possible exponent), and a negative by the twos complement. A negative number has a 1 for its sign and the twos complement of the fraction. In order to prevent excessive information loss, it is necessary to adopt a consistent procedure for keeping floating point numbers in a normal form. The usual procedure is to make the fractions as large as possible, thus keeping the exponents as small as possible. The processor has these two instructions to facilitate the manipulation of floating point numbers in normal form.

NRM Normalize '000101

0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Shift the contents of A and B left arithmetically, bringing 0s into B bit 16, bypassing B bit 1, leaving A bit 1 unaffected, and dropping bits out of A bit 2, until A bit 2 is in the state opposite that of A bit 1. Since the only data shifted out of A bit 2 is equal to the sign, no information is lost. Place the number of shifts performed in location 6 (the previous contents of location 6 are lost).

SCA Load Shift Count into A '000041

0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Load the contents of bits 9-16 of location 6 into A bits 9-16 and clear A bits 1-8.

By shifting until bit 2 differs from the sign,

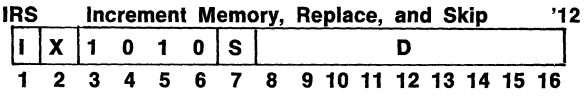
normalization produces a fraction in the range ½ to (1 − LSB) or -(½ + LSB) to -1. Saving the number of shifts allows the program to determine any change in the order of magnitude of a result due to a fixed point operation on the fractions of floating point operands. The program can then use the information stored in location 6 to adjust the exponent. Finally the result is put in proper format by shifting the fraction to the correct position and inserting the exponent in the high order word.

2.7 SKIP INSTRUCTIONS

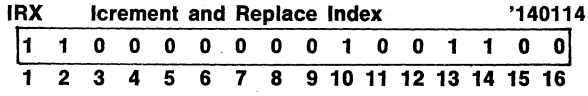
This group of instructions includes the entire skip group plus three instructions that increment or decrement a number and test the result, and two that compare one number with another. All instructions in the skip group have op codes beginning with 100000 in bits 1-6, whereas each of the other sets includes a memory reference instruction and instructions having op codes beginning with 110000.

Increment and Decrement

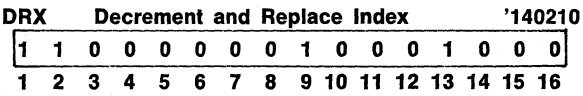
These instructions allow the program to keep a count in a memory location and to count the contents of the index register up or down. The skip test is always for a zero result. The instructions are used to count loop iterations or successively to modify a word for a series of operations.



Add 1 to the contents of location EA and place the result back in EA. Skip the next instruction in sequence if the result is zero.



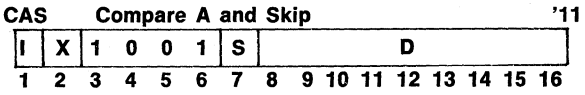
Add 1 to the contents of the presently selected index register and place the result back in that register. Skip the next instruction in sequence if the result is zero.



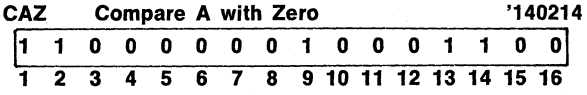
Subtract 1 from the contents of the presently selected index register and place the result back in that register. Skip the next instruction in sequence if the result is zero.

Compare

These two instructions do an algebraic comparison of the number in A with zero or a number in memory. They use a three-way test to allow skipping one or two locations as well as not skipping at all.



Compare the contents of A algebraically with the contents of location EA and act on the result as follows: if A is greater, go on to the next instruction in normal sequence; if the two are equal, skip the next instruction in sequence; if A is less, skip the next two instructions in sequence.



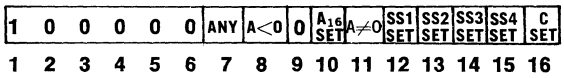
Compare the contents of A (fixed or floating) algebraically with zero and act on the result as follows: if A $\geq$ 0, execute the next instruction in sequence; if A = 0, skip the next instruction in sequence; if A $\leq$ 0, skip the next two instructions in sequence.

Skip Group

This group includes a number of miscellaneous skip instructions and also a combining set wherein skip conditions are selected by individual bits that may be combined to select several conditions at once. Bits 1-6 of all instructions are 100000. A 0 in bit 9 indicates the combining set, with individual conditions selected by 1s in bits 8 and 10-16. Bit 7 determines whether the condition is as given or is inverted; *i.e.*, a 1 in bit 7 indicates the condition is that specified by the remaining bits (any

of those specified in the combining set), whereas a 0 indicates the condition is opposite that specified (equivalent in the combining set to none of the specified conditions being satisfied). Any instruction can be given using the mnemonic SKP (which assembles as 100000) and giving the bit 7-16 configuration in the address field.

Combination Skip

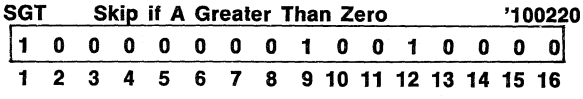


Skip the next instruction in sequence if bit 7 is 1 and any of the conditions specified by 1s in bits 8 and 10-16 is satisfied, or if bit 7 is 0 and none of the conditions specified by 1s in bits 8 and 10-16 is satisfied. (The conditions listed in the format box are those selected by 1s.) The various conditions, the bits that select them, and the mnemonics and op codes for them are as follows.

Mnemonic	Selector Bits	Bit 7	Skip on Condition	Op Code
NOP		1	None (no-op)	'101000
SKP		0	Skip unconditionally	'100000
SMI	8	1	A Minus ($A_1 = 1$)	'101400
SPL	8	0	A Plus ($A_1 = 0$)	'100400
SLN	10	1	LSB Nonzero ($A_{16} = 1$)	'101100
SLZ	10	0	LSB Zero ($A_{16} = 0$)	'100100
SNZ	11	1	A Nonzero	'101040
SZE	11	0	A Zero	'100040
SS1	12	1	Sense Switch 1 Set	'101020
SR1	12	0	Sense Switch 1 Reset	'100020
SS2	13	1	Sense Switch 2 Set	'101010
SR2	13	0	Sense Switch 2 Reset	'100010
SS3	14	1	Sense Switch 3 Set	'101004
SR3	14	0	Sense Switch 3 Reset	'100004
SS4	15	1	Sense Switch 4 Set	'101002
SR4	15	0	Sense Switch 4 Reset	'100002
SSS	12-15	1	Any of Sense Switches 1-4 Set	'101036
SSR	12-15	0	Any of Sense Switches 1-4 Reset	'100036
SSC	16	1	Set C	'101001
SRC	16	0	Reset C	'100001

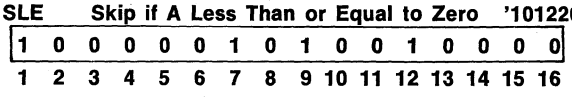
Skip conditions can be combined using SKP and giving the bit 7-16 configuration for the combination in the address field.

SGT Skip if A Greater Than Zero '100220



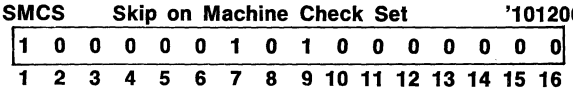
If the number contained in A (fixed or floating) is greater than zero, skip the next instruction in sequence.

SLE Skip if A Less Than or Equal to Zero '101220



If the number contained in A (fixed or floating) is less than or equal to zero, skip the next instruction in sequence.

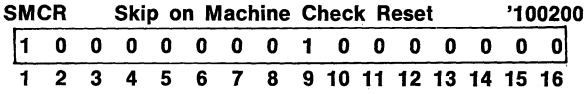
SMCS Skip on Machine Check Set '101200



If Machine Check is set (indicating a machine detected error), skip the next instruction in sequence. (When the processor is in machine check mode, this instruction has no meaning and executes as a NOP)

Note: The assembler recognizes the mnemonic SPS as equivalent to SMCS.

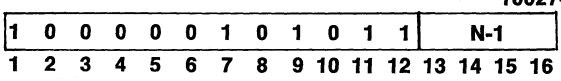
SMCR Skip on Machine Check Reset '100200



If Machine Check is reset (indicating no machine detected error), skip the next instruction in sequence. (When the proccessor is in machine check mode, this instruction has no meaning and executes as an absolute skip.)

Note: The assembler recognizes the mnemonic SPN as equivalent to SMCR.

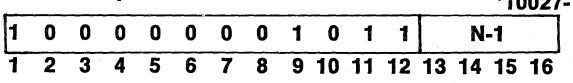
SAS Skip on A Bit Set '10026-'10027-



If A bit *N* is 1, skip the next instruction in sequence.

Note: The assembler will convert *N* to the octal equivalent of the bit number minus one.

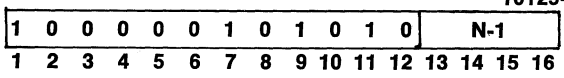
SAR Skip on A Bit Reset '10026-'10027-



If A bit *N* is 0, skip the next instruction in sequence.

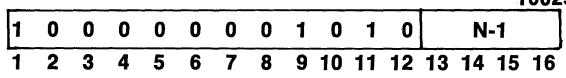
Note: The assembler will convert *N* to the octal equivalent of the bit number minus one.

SS Skip on Sense Switch Set '10124- '10125-



If sense switch *N* is on (up), skip the next instruction in sequence.

SR Skip on Sense Switch Reset '10024 '10025



If sense switch *N* is off (not up), skip the next instruction in sequence.

SS 1 — 101240	SR 1 — 100240
SS 2 — 101241	SR 2 — 100241
SS 3 — 101242	SR 3 — 100242
SS 4 — 101243	SR 4 — 100243
SS 5 — 101244	SR 5 — 100244
SS 6 — 101245	SR 6 — 100245
SS 7 — 101246	SR 7 — 100246
SS 8 — 101247	SR 8 — 100247
SS 9 — 101250	SR 9 — 100250
SS10 — 101251	SR10 — 100251
SS11 — 101252	SR11 — 100252
SS12 — 101253	SR12 — 100253
SS13 — 101254	SR13 — 100254
SS14 — 101255	SR14 — 100255
SS15 — 101256	SR15 — 100256
SS16 — 101257	SR16 — 100257

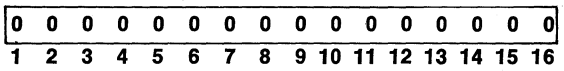
SAS 1 — 101260	SAR 1 — 100260
SAS 2 — 101261	SAR 2 — 100261
SAS 3 — 101262	SAR 3 — 100262
SAS 4 — 101263	SAR 4 — 100263
SAS 5 — 101264	SAR 5 — 100264
SAS 6 — 101265	SAR 6 — 100265
SAS 7 — 101266	SAR 7 — 100266
SAS 8 — 101267	SAR 8 — 100267
SAS 9 — 101270	SAR 9 — 100270
SAS10 — 101271	SAR10 — 100271
SAS11 — 101272	SAR11 — 100272
SAS12 — 101273	SAR12 — 100273
SAS13 — 101274	SAR13 — 100274
SAS14 — 101275	SAR14 — 100275
SAS15 — 101276	SAR15 — 100276
SAS16 — 101277	SAR16 — 100277

Summary of sense switch and bit test instruction op codes

2.8 CONTROL

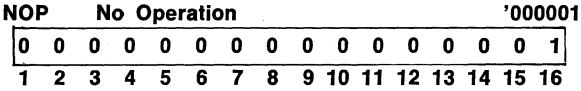
Many of the control instructions have already been described: those that select the index register and addressing mode are discussed with memory reference in §2.1, and those that choose between single and double precision arithmetic operations are treated in §2.6. Here we describe several miscellaneous control instructions and those associated with parity errors and the status keys.

HLT Halt '000000



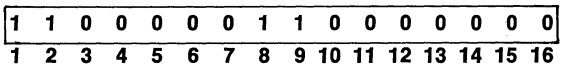
Halt the processor with the STOP indicator lit on the control panel and P pointing to the next instruction in sequence (the instruction that would have been executed had the HLT been replaced by a no-op). The data lights display the next instruction, and the address lights display the instruction OTA '1720 [§2.10]. (This latter instruction is part of the control panel microroutine; turning the function switch to any of the right five positions displays P instead.)

NOP No Operation '000001



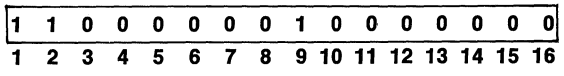
Do nothing but go on to the next instruction.

SCB Set C Bit '140600



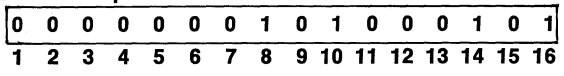
Set C.

RCB Reset C Bit '140200



Reset C.

SVC Supervisor Call '000505



Place the CPU in 16s mode and generate an interrupt through location '65.

To understand the actual implications of this instruction the reader must be familiar with the interrupt, which is presented in §2.11.

Parity Errors

Parity errors are classified as of two types: a parity error in a word read from memory is a memory parity error; a parity error in any other situation (in a register, over the I/O bus, etc.) is a machine check error. When the processor is in machine check mode, both types of errors are handled differently than when the processor is not in that mode, and for a machine check error the processor may actually execute a microprogram check of the machine to determine the source of error.

Occurrence of an error of either type sets the Machine Check flag, but this flag has meaning for the program only when the processor is *not* in machine check mode. When power is turned on or the computer is cleared from the control panel, the processor is automatically *not* in machine check mode; in this case the program can check for parity errors by sensing the flag (using the skip instructions defined in the preceding section) and can also reset the flag.

If the program places the processor in machine check mode, a parity error still sets the flag but the microprogram response always resets it, so it is irrelevant to the programmer and the skip instructions that sense it are meaningless. With the processor in machine check mode, a memory parity error produces an interrupt through location '67, and the response beyond that is entirely up to the service routine. For a machine check error, action by the microprogram depends upon the type of processor.

All processors except type 221 simply halt; and a halt for this reason is indicated by the series or the 218 series simply halts; and a halt for this reason is indicated by the fact that the data lights are all off, and if the operator turns the function selector to STOP/STEP, all the address lights go on. A type 221 processor responds by executing the microprogram verification routine to determine the source of error.

So long as the failure recurs, the processor continues to perform the verification routine indefinitely with the number of the failing test in the address lights. However should the failure disappear, the mircoroutine clears the keys (see below) and register file, and interrupts through location '70 to return control to the program. The program can also execute the verification routine at any time as an instruction to determine whether all is well.

EMCM Enter Machine Check Mode '000503

0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Enter machine check mode so that the micro program will respond to a parity error as described above.

LMCM Leave Machine Check Mode '000501

0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Leave machine check mode so that a parity error will simply set Machine Check.

RMC Reset Machine Check '000021

0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Reset the Machine Check flag.

Note: The assembler recognizes the mnemonic RMP as equivalent to RMC.

VIRY Verify '000311

0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

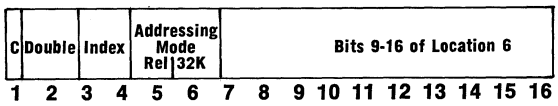
Execute the verification routine, and if there is a failure of any kind go on to the next instruction with the number of the test that failed in A. If there are no errors, skip the next instruction in sequence.

If the processor does not have the verification routine, this instruction executes as a no-op.

Status Keys

In order that the program be able to determine which register is being used for indexing, what the currently specified size

of the address space is, what the present type of addressing is, and so forth, a number of internal machine conditions, referred to as "keys," are available in a status word that can be read by the program. The format of this key word is as follows.



1 The state of C.

2 0 — Single precision, 1 — Double precision.

3-4 The currently selected index register as follows:

00	X
10	A
01	B
11	S

5-6 The current addressing mode as follows:

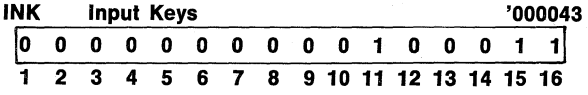
00	16K Sectored
01	32K Sectored
11	32K Relative
10	64K Relative

Note that a 1 in bit 5 indicates relative mode, a 1 in bit 6 indicates a 32K addressing space.

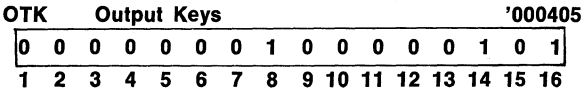
9-16 Bits 9-16 of location 6, which may contain a normalize shift count.

Not only can the program read the above information, but it can also set up the machine state according to a similar key word supplied by the program. *Eg* giving a key word with a 1 in bit 1 actually places the arithmetic logic in double precision mode, whereas giving the word with a 0 in bit 1 limits the basic arithmetic operations to single precision.

The processor has two instructions for reading and setting up the keys. The principle use of these instructions is for saving and restoring the keys in conjunction with program interrupts. Before doing its own operations, an interrupt service routine should save any parts of the register file it will use and should save the keys if it is going to make any change in the modes of operation. After completing its own task, the routine should restore the original machine state before returning to the interrupted program.



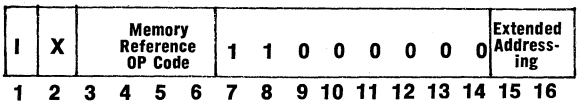
Read the key word defined above into A.



Set up C and the various machine modes according to A bits 1-6 as defined by the key word given above, and load A bits 9-16 into bits 9-16 of location 6 (shift counter). Clear bits 1-8 of location 6.

2.9 ADDRESS EXTENSIONS

In 32K relative mode with a 1 in the sector bit, the displacement must lie in the range -240 to +255. The other numbers representable by the displacement bits extend the addressing techniques. Special addressing is specified by the configuration of bits 15-16 in a memory reference instruction that is given in 32 K relative mode and in which bits 7-12 have the configuration 1 100 00.



Bits 15 and 16 in conjunction with the flag and tag bits, define the special types of addressing listed in the table below. The various types are based on three techniques: use of two-word instructions where the second word is an absolute address (making all of memory directly accessible to the instruction), use of S for an indexing procedure that is independent of and may be used in conjunction with standard indexing, and use of S to implement a push-pop stack.

The easiest way to keep track of the different procedures is to view them as divided into four sets determined by *I* and *X*, where each set has four cases determined by bits 15 and 16. The first set (where initial address equals effective address) contains the basic operations of which the other sets are mostly straightforward extensions. In case 0 of the first set, the effective address is simply the absolute address con-

32K Relative: Extended Effective Address Calculation
Bits 7-12 = 1 100 00 (S = 1, −256 ⩽ D ⩽ −241)

P Address of instruction location (contents of program counter before instruction fetch)
A Contents of location P + 1 (bit 1 is ignored)
S Contents of stack register
X Contents of currently selected index register
I(ξ) Result of indirect chain beginning with access to location addressed by ξ

I	X	Bits 15-16	EA	Ancillary Action	Type of Addressing
0	0	0	A		Address
0	0	1	A + S		Base plus displacement
0	0	2	S	S + 1 → S	Push/pop
0	0	3	S − 1	S − 1 → S	Pop/push
0	1	0	A + X		Address, indexed
0	1	1	A + S + X		Base plus displacement, indexed
0	1	2	I(S) + X	S + 1 → S	Push/pop indirect, postindexed
0	1	3	I(S − 1) + X	S − 1 → S	Pop/push indirect, postindexed
1	0	0	I(A)		Address indirect
1	0	1	I(A + S)		Base plus displacement, indirect
1	0	2	I(S)	S + 1 S	Push/pop indirect
1	0	3	I(S − 1)	S − 1 S	Pop/push indirect
1	1	0	I(A + X)		Address indexed, indirect
1	1	1	I(A +S + X)		Base plus displacement indexed, indirect
1	1	2	I(A) + X		Address indirect, postindexed
1	1	3	I(A + S) + X		Base plus displacement indirect, postindexed

tained in the location following the instruction—in other words the instruction is two words, of which the second is the address. Case 1 also involves a two-word instruction, where the second is a base that is changed by a displacement contained in the stack register (this is effectively an index procedure). Cases 2 and 3 provide the basic implementation of the push-pop stack. If the stack is regarded as made up of locations $N, N+1, N+2, \ldots$ and it is assumed that S always points to the next open location, then case 2 is push and case 3 is pop. On the other hand if the stack is viewed as locations $N, N-1, N-2, \ldots$ and S always points to the last filled location, then case 3 is push and case 2 is pop.

If I is 1 but X is 0 (the third set), the initial addresses from the four cases just discussed are taken as indirect; *i.e.*, as the beginning of an indirect chain. If I is 0 but X is 1 (the second set), indexing is used but not quite in the fashion implied by *I* and *X*. In cases 0 and 1 the initial addresses are simply indexed; but in cases 2 and 3 the stack address is taken as *indirect*, and indexing occurs at the completion of the address chain. In the fourth set, with I and X both 1, indexing and indirection are both used but in ways related only to cases 0 and 1 of the first set (the analogous push-pop procedure implied by *I* and *X* both 1 is actually implemented as cases 2 and 3 of the second set). In cases 0 and 1 the initial address is indexed and then used indirectly. In cases 2 and 3 the initial addresses are those otherwise associated only with cases 0 and 1; the initial address is used indirectly and indexing is put off until the end (in other words postindexing is substituted for the preindexing of cases 0 and 1).

2.10 INPUT-OUTPUT

Instructions in the I/O class govern the transfer of data to and from the peripheral equipment, and also perform some functions in the processor. The class comprises four types of instructions for sending control pulses out to a device, testing conditions in a device for a skip, and moving data or other information out to a device or in from it. An instruction in the I/O class is designated by 1100 in bits 3-6, and the type is indicated by bits 1 and 2; hence the four types of I/O instructions have op codes '14, '34, '54 and '74. Bits 7-10 specify the particular function the instruction is to perform, and bits 11-16 select the device that is to respond to the instruction. The format thus allows sixty-four codes for addressing devices ('00-'77) and sixteen

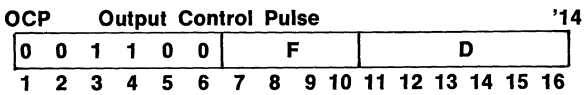
for specifying functions ('00-'77) that a given type of I/O instruction can perform using the addressed device.

Device code '20 is used for communication with the control panel and for controlling interrupts and the real time clock. The other sixty-three codes are available for external devices, but many are assigned to standard equipment. The meanings of the function codes differ with the type of instruction and the type of device, although some are common to all devices. With the control type of instruction, the function code 00 usually "turns on" or "starts" the device (with whatever meaning that term may have vis-a-vis the particular device), and code '17 initializes the device, making it ready for use. An I/O skip instruction invariably uses function code 00 to determine whether a device is ready and code '04 to determine whether it is requesting an interrupt. The data instructions, in and out, generally use code 00 specifically for real data—as against moving control information, word counts, addresses, or status. A table in Appendix lists all devices for which device codes have been assigned, and lists the function codes used with them.

Typically a device interface has a 6-bit device selection network, Ready and Interrupt Enable flags, and logic nets that supply the device code, the device identification, and the number of the slot in which the interface is mounted. The selection network decodes bits 11-16 of the instruction so that only the addressed device responds to signals sent by the processor over the I/O bus. The Ready flag indicates just that: the device is ready—meaning it has just completed a task requiring some response by the processor, or it is idle and may be used. Considering devices at the simplest level, the program places an output device in operation by giving a data-out instruction that resets Ready and sends the first unit of data—a word or character depending on how the device handles information. When the device has processed the unit of data, it sets Ready to indicate that it is ready to receive new data for output. With an input device, the program gives a control instruction to place the device in operation and reset Ready. When the device has read a unit of data, it sets Ready to indicate that it has data ready for the processor. The program responds by giving a data-in instruction that not only brings in the data but also resets Ready and tells the device to read more data; to end the process the program must actually issue a control command to stop the device. With either type of device, the setting of Ready requests an interrupt if the Interrupt Enable flag is set. If the program does not wish to use the device, it can reset Interrupt Enable to prevent the idle state of the device from continually requesting an interrupt.

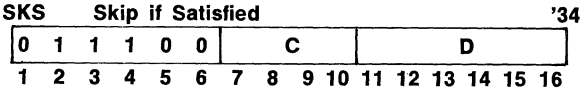
Every device can supply its device code for use by the interrupt system (although a more complex device may be set up to supply an interrupt address specified by the program rather than using its own device code). The program can read the slot number in order to determine the position of any device on the I/O bus (this determines priority with respect to the vectored interrupt) and can read the identification number of each device. The latter number not only identifies the type of device, but also indicates any modification from the standard, and indicates which one it is if several of the same type are connected to the bus.

In the discussions of the various I/O devices in Chapter 3 and beyond, all instructions described are special cases of these four I/O instruction types.



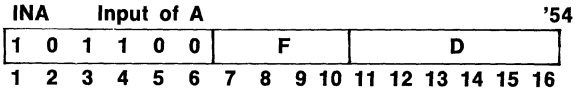
Send a control pulse for the function specified by *F* to device *D*.

SKS Skip if Satisfied '34



Skip the next instruction in sequence if the condition specified by *C* is satisfied in device *D*.

INA Input of A '54

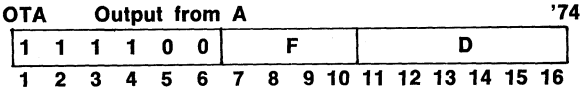


If the function *F* specifies a transfer for which Ready must be set, then if the Ready flag in device *D* is reset, do nothing but go on to the next instruction, whereas if Ready is set, perform the function *F* and skip the next instruction in sequence. To perform the function the processor reads the information specified by *F* from device *D* into A and performs whatever control operations are appropriate to the function and the device. Depending on *F*, the information read may be data, status, an address, a word count, or anything else.

The number of bits brought into A depends on the type of information, the size of the device register, the mode of operation, etc. Bits in A that do not receive information are cleared.

INA instructions for any device except device '20 use a ready test and skip the next instruction if the device was ready. When the INA is used to input a status register, the controllers are always ready.

OTA Output from A '74



If the function *F* specifies a transfer for which Ready must be set, then if the Ready flag in device *D* is reset, do nothing but go on to the next instruction, whereas if Ready is set, perform the function *F* and skip the next instruction in sequence. On the other hand if the state of Ready is irrelevant to the specified transfer, then perform the function *F* and go on to the next instruction in normal sequence without making any ready test at all. To perform the function the processor sends the contents of A to device *D* for the purpose specified by *F* and performs whatever control operations are appropriate to the function and the device. Depending on *F*, the information sent may go to a data buffer, a control or address register, a word counter, or any other destination.

The number of bits actually accepted by the device depends on the type of information, the size of the device register, the mode of operation, etc. The contents of A are unaffected.

An OTA instruction for any device discussed in the remainder of this manual uses a ready test and the skipping procedure as stated in the description of the instruction. An OTA to device '20 makes no test and cannot skip.

In the symbolic program an instruction is given using the defined mnemonic and placing the 4-digit octal code for function and device (with the function on the left) in the address field. *E.g.*, the device code for the paper tape reader is 01, and the function code for sensing whether a device is requesting an interrupt is '04; hence

```
SKS  '0401
```

is an instruction that skips if the reader is *not* presently requesting an interrupt.

The fact that the input and output instructions for data or other information include a ready test allows the program to give such an instruction without knowing whether the device is ready. If the program is ready to move data, it can just give an INA or OTA; if the device is *not* ready, the program can then go off to do something else and come back later to try again. Or the program can wait for say the reader to get a character from tape like this:

```
INA  '0001   If ready, read; otherwise
JMP  *-1     go back until ready,
...          then continue
```

The INA causes the device to read another frame, so if the program prefers not to

have the tape continue it must give

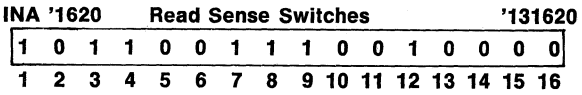
```
OCP  '0101
```

to stop the reader.

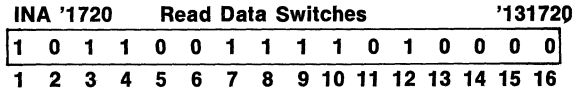
A device may require no transfers of real data at all, as is the case with the real time clock, but any device still uses at least three of the four instruction types. An output-only device or a device with no data requirements responds nonetheless to an INA for identification and generally recognizes another for supplying status information. Even a simple input-only device may recognize an OTA instruction for sending out control information. A high speed device, such as magnetic disk or tape, generally uses INA and OTA instructions only for status and control information with data moving directly between the device and memory via a direct memory channel. An instruction addressing a nonexistent device or specifying a function that is inapplicable to the addressed device is just a no-op.

Control Panel Communication

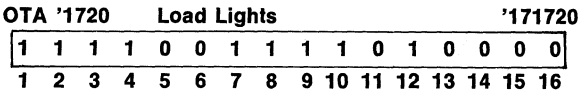
The program can communicate with the operator via the control panel by virtue of the fact that it can address the panel as an I/O device. With the following instructions the program reads the contents of the switch register as data or as sense switches and loads a data register whose contents can be displayed in the lights (in no case is a ready test necessary).



Read the contents of the control panel switch register as sense switch into A, where a switch that is on is read as a 1. A switch is on as a sense switch if it is up. This instruction does not skip.



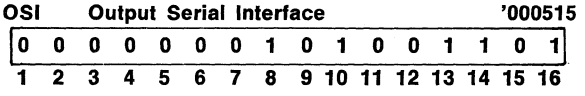
Read the contents of the control panel switch register as data switches into A, where a switch that is on is read as a 1. A switch is on as a data switch if it is in the momentary down position; in any other position it is regarded as off. This instruction does not skip.



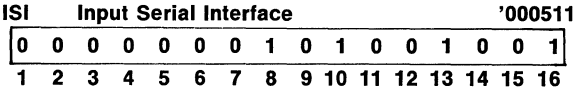
Load the contents of A into the control panel data register. If the address/data switch is set to DATA, the data register is displayed in the lights (a 1 turns on the corresponding light). This instruction does not skip.

Processor Serial Interface

Besides the many peripheral devices connected to the I/O bus and controlled by I/O instructions, there is a basic serial interface that is built right into the processor and is controlled by special instructions. By means of this device the program can control the transmission of serial data on four output lines and can receive serial data simultaneously over four input lines. The program handles output by periodically changing the contents of a 4-bit output register in which each bit is connected to a separate output line; thus successive changes in the register contents produce bit-by-bit serial transmission over the lines. Data is received by sampling the input lines to pick up bit-by-bit serial input. The device operates entirely on EIA standard levels and the lines are available at the back edge connector of the processor board. Output lines 1-4 are respectively at pins CF-41, CF-35, CF-39 and CF-37; signals can be supplied to input lines 1-4 at pins CF-36, CF-38, CF-40 and CF-42. The program supplies data to and receives data from the lines via A bits 13-16, where line 1 corresponds to bit 13. Input and output are handled by these two instructions.



Load the contents of A bits 13-16 into the 4-bit buffer whose contents are held on the serial interface output lines. Bit 13 supplies the data for line 1.



Read the contents of the serial interface input lines into A bits 13-16, with line 1 corresponding to bit 13.

The lines may be used for anything that involves transmission or reception of binary EIA signals. An output line could be used to control a light to signal the operator; an input line might be connected to a switch, allowing a person or a device to supply a binary signal that can be sampled at appropriate times by the program. The lines can also be used for standard data communication where the program is entirely responsible for all timing, for constructing characters with appropriate start and stop bits, and for stripping the data out of received characters. For output the usual procedure is simply to change the signal on the output line for each bit in a serial transmission. The program determines character length and transmission frequency, and can actually run the output lines at different rates—as would be the case were one line being used for serial transmission and another to control a signal light. Whenever any bit of the output register is changed, information previously given for the other lines must be repeated to keep the appropriate signals on them.

For input both the frequency and character length must be known. In conventional data communications, an idle line is constantly marking (continuous 1s) and the beginning of an asynchronous character is indicated by a starting space (a 0 bit). The usual procedure is to sample the line at five times the bit rate. Upon reading a 0 on a line that has been idle, the program should assume it has discovered only a possible space; if a 0 is still read at the next two sample times, it can be assumed that the line has a true space rather than a transient, and transmission has started. The program should then read the line at every fifth sample time so that reading is centered within each bit time. If a number of lines are operating, the program must keep track of them separately, *i.e.*, the program must keep the read times centered on each line independently of the others.

With sophisticated software the serial interface could actually be used for a complete data communication channel with even the automatic answering of incoming calls in a private network or the public dial telephone system. For such an arrangement one input line would be used for data and the others for modem control signals such as Ring Indicator, Clear To Send, Carrier Detected, and Data Set Ready. Output would require three lines: one for data, and two for the control signals Request To Send and Data Terminal Ready.

2.11 EXTERNAL INTERRUPT

Many I/O devices must be serviced infrequently relative to the processor speed and only a small amount of processor time is required to service them, but they must be serviced within a short time after they request it. Failure to do so within the specified time (which varies among devices) can result in loss of information and certainly results in operating the device below its maximum speed. The external priority interrupt is designed with these considerations in mind, *i.e.*, the use of interruptions in the current program sequence facilitates concurrent operation of the main program and a number of peripheral devices. The interrupt system also allows conditions internal to the processor to interrupt the program [§2.12], but here we are concerned only with external interrupts.

Interrupt requests by a device are governed by its Interrupt Ready and Interrupt Enable flags. When a device completes an operation it sets the Ready flag, and this action

requests an interrupt if Interrupt Enable is set—if Interrupt Enable has been reset by the program, the device cannot request an interrupt. The program controls the enabling flags by means of OCP instructions; moreover the flags in some devices are also connected to the I/O bus data lines, so the program can set up the enabling flags in all such devices at once by means of a mask sent over the bus.

At appropriate times the processor synchronizes any requests that are then being made. Once a request has been synchronized the device that made it must wait for an interrupt to start. Although the interrupt signal on the bus is disabled once an interrupt starts, the request made by the device remains until the program resets Ready or Interrupt Enable. If the program does reset Interrupt Enable in a device, that device not only cannot request an interrupt when its Ready flag sets, but any request it has already made is voided, so it is no longer waiting for an interrupt (and no I/O skip instruction can determine that it had requested one). However, if Ready is left set, setting Interrupt Enable restores the request.

Before beginning each instruction the processor takes care of all direct memory requests, including any additional requests that are made while direct memory transfers are being handled [see §§2.12, 2.13]. When no more devices are requesting access, the processor starts an interrupt if the external interrupt system is enabled and a device that has priority is requesting an interrupt. The way in which the hardware handles an interrupt and the way in which the program should respond depends upon the interrupt mode.

Standard Interrupt Mode

In standard mode any device that can make an interrupt request has priority to interrupt any program, even an interrupt service routine, unless the interrupt system is inhibited. The processor starts to service an interrupt by inhibiting the interrupt system so no further interrupts can be started, saving P (which points to the next instruction) in the location addressed by the contents of location '63, and begins the interrupt service routine by resuming normal instruction execution at the location following that in which P was stored.

Caution

The contents of any interrupt location ('63 for the standard interrupt) are always interpreted as a 16-bit absolute address. Therefore, when setting up interrupt locations, the program must make sure not to use addresses larger than available memory.

The service routine should determine which device requires service, save the keys and any parts of the register file that it will use, and service the device. The device can be identified by means of SKS instructions that test for interrupt requests. The program may leave the interrupt inhibited while servicing the device (or devices), or it can enable interrupts and establish a priority structure to allow higher priority devices to interrupt the current routine.

There are two ways in which the program can structure device priority. The service routine establishes a basic priority by the order in which it tests the devices. It can also define higher and lower priorities by setting up the Interrupt Enable flags in the devices and then reenabling the interrupt. In this way any device whose Interrupt Enable flag is reset cannot interrupt the current routine and is therefore defined as being of lower priority, whereas a device that is allowed to interrupt is defined as being of higher priority.

After servicing a device (or all devices found to be interrupting by an SKS chain), the routine should restore the preinterrupt states of the keys and the register file,

enable the interrupt, and return to the interrupted program by jumping indirect through the location in which P was stored. If the routine allows interrupts by higher priority devices, then before returning to the interrupted program it should reenable lower priority devices that were not allowed to interrupt the current routine but will be allowed to interrupt the program to which the processor is returning.

Vectored Interrupt Mode

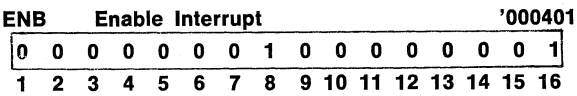
In vectored mode the processor responds to an interrupt request from a specific device and has a built-in priority structure such that lower priority devices cannot interrupt while the processor is holding an interrupt for a device of higher priority. The conditions for starting an interrupt are therefore the same as those given for the standard case with one exception: if the processor is already in an interrupt routine, it will go on to the next instruction even if interrupts are enabled, unless the requesting device is of higher priority than that for which the current interrupt is being held. When an interrupt is started and several devices are making requests simultaneously, the processor responds to that requesting device that has the highest priority (mounted in the lowest-numbered slot).

As in standard mode the processor inhibits further interrupts, saves P as specified by the contents of an interrupt location, and proceeds with the service routine at the position following that in which P was stored. However, unlike a standard interrupt, here there is no fixed interrupt location—instead the location is specified by the device to which the processor is responding. In most cases the device specifies an address '100 greater than its device code, but a complex device may have an address register for this purpose so that the program can specify the location through which the device will interrupt. Since the system uses a location unique to each device, there is no need for testing, and the service routine acts only for the interrupting device (it should of course save keys and registers as usual). There is also a built-in priority determined by bus position, so even if the routine allows interrupts, no device higher on the bus can do so (in other words all devices in higher-numbered slots are of lesser priority). Moreover the program can still pick and choose among the nearer devices by adjusting the individual Interrupt Enable flags. Hence in vectored mode devices of higher interrupt priority can interrupt a given routine once interrupts are re-enabled.

When returning to the interrupted program, the routine must restore the preinterrupt state and either reenable interrupts or reestablish the appropriate priority structure. Furthermore a routine for a vectored interrupt must also give a specific instruction (CAI, defined below) to clear the presently active interrupt so the processor can then respond to requests from devices of lower interrupt priority.

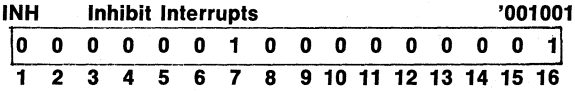
Interrupt Programming

The instructions that control the interrupt system are all of the type with a full word op code, but associated with the system are two I/O instructions that deal with the mask used for setting up the Interrupt Enable flags in certain devices. When power is turned on or the computer is cleared from the control panel, the processor is automatically in standard interrupt mode with interrupts inhibited.



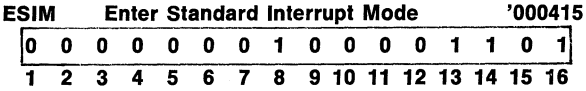
Enable the external interrupt system so the processor will respond to interrupt requests over the I/O bus. This instruction becomes effective following execution of the next sequential instruction.

INH Inhibit Interrupts '001001



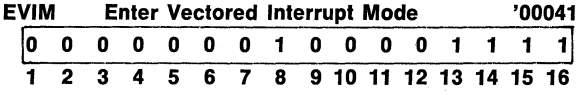
Inhibit the external interrupt system so the processor will not respond to interrupt requests over the I/O bus. This instruction takes effect immediately.

ESIM Enter Standard Interrupt Mode '000415



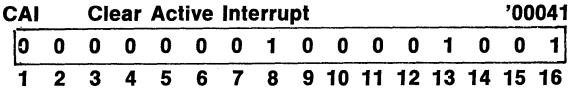
Enter standard interrupt mode so that all interrupts are made through location '63.

EVIM Enter Vectored Interrupt Mode '000417



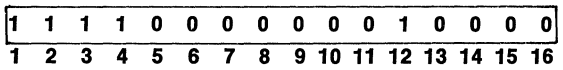
Enter vectored interrupt mode so that for interrupt purposes the priority of a device is determined by its position on the I/O bus (with lower devices having higher priority) and each interrupt is made through the location specified by the sole interrupting device.

CAI Clear Active Interrupt '000411



Terminate the presently active interrupt so that the processor can recognize interrupt requests from devices in higher slots than the device for which the current interrupt is being held. This instruction is of use only in vectored interrupt mode.

SMK Send Mask '170020

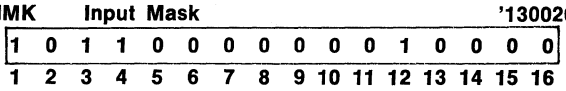


Set up the Interrupt Enable flags in the devices according to the mask in A (a 1 in a mask bit sets the flag in the device corresponding to that bit; a 0 resets it). Note that this instruction is equivalent to OTA '0020; and it never skips.

The bits in the mask and the devices assigned to them are as follows (note that the mask does not necessarily control the Interrupt Enable flags in all devices).

1	
2	
3	
4	Moving head disk
5	
6	
7	
8	Fixed head disk
9	Paper tape reader
10	Paper tape punch
11	Teletypewriter
12	
13	
14	
15	
16	Real time clock

IMK Input Mask '130020



For those devices associated with the mask that can be supplied by an SMK, read the states of their Interrupt Enable flags into A (the correspondence of devices to mask bits is the same as given above). Note that this instruction is equivalent to INA '0020; and it never skips.

Timing. The time a device must wait for an interrupt to start depends on how many devices are using interrupts, how long the service routines are for devices of higher priority, and whether the direct memory channels are in use. In vectored mode a single device will shut out all others of lower priority until a CAI instruction is executed; and the direct memory channels shut out all interrupts when they operate at the maximum rate. If the DMA channels are not in use and 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. Without delays caused by indirect addressing, the maximum interrupt waiting time is the latency given in the table at the end of Appendix.

Programming Suggestions. If the program has little computing to do and is using only one or two fast I/O devices or several slow ones, it may not be necessary to use the interrupt at all. On the other hand, if there are many calculations to perform and the program is using a fast device or data is

being processed using several slower devices, then the interrupt is necessary. The critical factors in determining whether to use the interrupt, and in what ways the program should determine priority, are what the program is doing besides input-output and the time required by the service routines.

A convenient method for handling a large number of priority levels is to use a push-pop stack for saving the machine state. This obviates setting aside so many specific locations for saving registers, and makes it very easy for a routine at any level in a sequence of nested routines to restore the state for the interrupted program.

For those who do program interrupt routines, there are several rules to remember.

■ An interrupt cannot be started until the current instruction is finished. Therefore do not use lengthy indirect address chains if a device that requires very fast service can request an interrupt.

■ The service routine should save the keys and any parts of the register file that it will use.

■ The principle function of an interrupt routine is to respond to the situation that caused the interrupt. *E.g.*, computations that can be performed outside the routine should not be included within it.

■ Before returning to the interrupted program, the routine should restore the keys and the register file, and in vectored mode it must give a CAI.

2.12 INTERNAL INTERRUPTS

Besides the use of interrupts to handle the peripheral equipment, there are a number of internal processor situations that can interrupt the program. The action taken in response to an internal interrupt is essentially the same as for an external interrupt, but many of the conditions associated with the latter are not applicable to the former. All internal interrupts are vectored regardless of the mode of the external interrupt.

Although a particular type of internal interrupt may be inhibited at its source, it is never affected by the enabling or inhibiting of external interrupts as a class; *e.g.*, a memory parity error can cause an interrupt only if the processor is in machine check mode, but with that mode in effect, an error always causes an interrupt even if external interrupts are inhibited. All internal interrupts have priority over external interrupts by virtue simply of the circumstances they represent; and among internal interrupts, priority is a straightforward function of logical necessity.

In response to a particular internal interrupt, the processor vectors through a specific location. If the 16-bit absolute address in this location is zero, the processor halts. If the address is nonzero, the processor inhibits external interrupts, saves P in the location addressed by the contents of the interrupt location, and resumes normal program execution at the location following that in which P was stored. Since an internal interrupt has nothing to do with the bus priority structure, the service routine need not give a CIA upon completion.

Internal interrupts are used to monitor the hardware and aid in software execution. Interrupt locations and the conditions that generate interrupts through them are as follows.

'60 Power Failure—incoming power is not up to specification. This vector should be left unimplemented (zero) unless the processor has the memory save option [§2.14].

'61 Real Time Clock Counter—this is not an internal interrupt at all, but is used as a counter by the real time clock [§2.14].

'62 Reserved

'63 External interrupts use this location.

'64 Reserved

'65 Supervisor Call—an interrupt to this

location is produced by the SVC instruction [§2.8].

'66 Unimplemented Instruction—the program has given an op code that is reserved for instruction use, but whose execution is not implemented in the microcode. This interrupt may be used to call a routine that implements such instructions through software; to further this objective, the stored program address points to the instruction that caused the interrupt.

'67 Memory Parity Error—the processor is in machine check mode and has detected an error in data just read from memory. This interrupt automatically resets Machine Check.

'70 Machine Check Recovery—the processor is in machine check mode and has been executing the verification routine in response to detection of a machine check error, but in the last iteration of the routine no errors were discovered.

'71 Missing Memory Module—either the memory board containing the addressed location is not installed or a software bug has generated an address outside of the available memory space. This interrupt automatically resets Machine Check. It may be used to determine memory size.

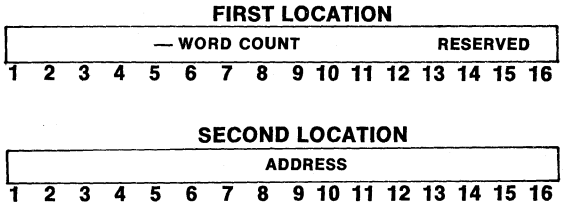
'72 Illegal Instruction—the program has given an op code that is not now and never will be implemented as an instruction in the microcode. This feature allows the user to design software to implement such codes with assurance that they will never come to be used for PRIME-implemented instructions. To further this objective, the stored program address points to the instruction that caused the interrupt.

'73 Reserved

2.13 DIRECT MEMORY ACCESS

Handling data transfers between external devices and memory under programmed I/O control requires the execution of several instructions for each word transferred. To allow greater transfer rates the processor contains eight direct memory channels through which devices, at their own request, can gain direct access to memory using a minimum of processor time. At rates lower than the maximum, the channels free the processor to allow execution of a program concurrently with data transfers for high speed devices such as disk and magnetic tape.

To control a direct memory transfer, the program sets up a device to use a particular channel and sets up a pair of memory locations to define the channel. The channels use locations '20-'37 in the register file, with locations '20 and '21 governing channel one, '22 and '23 governing channel two, and so on to '36 and '37*. To set up the device, the program gives an OTA that supplies the controller the address of the first channel location to be used. The program places a 12-bit word count in the first location, and the address of the first word to



be transferred in the second. The word count is in bits 1-12 and is the twos complement of the number of words to be transferred; the maximum number of words in a single block on one channel is therefore 4096, produced by a negative count of zero (a single device can handle larger blocks by stepping through successive channels). The contents of the second

*The processor permits any contiguous pair of locations in the register file to be used, although some locations, such as the program counter or those reserved for microprogram functions, are obviously not appropriate for this purpose. The programmer can use X, A, B, S, and certain other locations when necessary.

address are interpreted as a 16-bit absolute address regardless of memory size.

When the device requires data service, it requests access to memory via its channel. Between instructions and at various points within an instruction, the processor can pause to handle a transfer. If several devices are waiting for service simultaneously, the first to receive it is the one that is mounted in the lowest-numbered slot. Whenever the processor pauses to handle a DMA request, it handles all pending requests before resuming the instruction, starting an interrupt, or going on to the next instruction.

To service a channel request the processor accesses the location specified by the channel address, sends its contents out over the bus or stores in it a word taken from the bus as specified by the device, and increments both the address and the word count by one. When the word count overflows (goes to zero), the processor signals the device that the block is complete. Typically, complex device controllers such as those for fixed and moving head disks can automatically chain DMA channels thereby facilitating scatter/gather data transfers.

Timing. The time a device must wait for channel access depends on when its request is made within an instruction and how many devices of higher priority are also requesting access; a given device must wait until all devices of higher priority have been serviced, so the highest priority device can preempt all processor time if it requests access at the maximum rate. The microprogram must save certain registers to service the channel, and although it can pause within an instruction it cannot take direct memory requests while starting an interrupt, so the worst case waiting time for the highest priority device is 3-4 μs for an isolated transfer. But once an initial transfer has taken place, continuous transfers can be handled at the rate of one every 1.2 μs; this allows a maximum of 833,333 words per second, but at this rate all other processing activity is suspended.

2.14 CONTROL OPTIONS

Equipment in this category is mostly processor options, such as the memory save with automatic restart after power failure and the automatic program load. Also included is the real time clock, which is a device connected to the I/O bus but which is used by the processor for control purposes rather than for the movement of data in and out of the system.

Memory Save and Automatic Restart

If ac power should fail for any reason (including being turned off), the processor automatically interrupts through location '60; this is an internal interrupt, which is not affected by the inhibit for external interrupts. Without the memory save option the program should put a zero vector in location '60 so that the processor just halts. But with the option, the interrupt routine should save the keys and the entire register file in MOS memory, shut down the I/O equipment, and then give an HLT. The program has about a millisecond in which to do this before the power monitor issues a master clear and turns control over to the battery-operated memory save unit. This unit continues to refresh memory until power is again within specifications, at which time it restarts normal program execution at location '1000 provided the function switch is in the RUN position. Naturally the program must set up location '1000 ahead of time in whatever way is appropriate for restarting.

The memory save unit, which automatically recharges its battery while power is on, is equipped with one 20 ampere hour battery and has space for installing another. The unit can be mounted at the front or back of the rack; it takes 7 inches of panel space and is 8 inches in depth. On its panel are a full charge indicator,the battery

terminals, a switch to disconnect the battery, and terminals for a meter.

Automatic Program Load

The automatic program load option enables the operator to reload programs from devices such as fixed-and moving-head disks, and paper tape simply by initiating a hardware bootstrap from the control panel. This feature saves considerable time and effort by eliminating the tedious and error-prone procedure of manually keying in a bootstrap loader one word at a time.

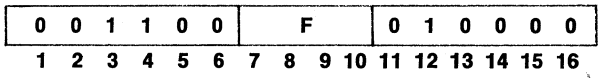
On a machine with the program load option, the LOAD position of the function switch calls a microcode loader. There are two versions of this loader, one for the fixed-and moving-head disks, and one for the ASR and high speed paper tape readers. The operator uses the sense switches to select which version is to run and to specify whatever other information is needed by the microprogram. The disk version reads 460 words from sector 0 of the selected disk, storing the words beginning at location '770; after reading the data, the processor begins normal program execution at location '1000 (the program executed—*i.e.*, the data read in from the disk—is entirely at the discretion of the programmer). The reader microprogram reads any Prime self-loading tape. Tapes of the assembler, linking loader, text editor and other basic programs are available from Prime in self-loading format. Also, any tape punched by the memory dump and load program (MDL) is in the self-loading format and its data is stored in the same part of memory from which it was punched.

Real Time Clock

This option can be used to signal the end of a specified real time interval or to measure te real time taken by an event; with appropriate software the clock can easily be used to keep the time of day. Although it is for internal control purposes, the clock is handled like an I/O device and is mounted on the same board as the reader, punch and teletypewriter controllers. The device is based on a clock source, which is usually the ac line (60 or 50 Hz), but an alternate source can be mounted on the board or supplied externally. While the clock is on, it responds to each pulse from the source by incrementing the contents of location '61—in other words the clock uses memory location '61 as a counter. Whenever the counter overflows to all Os the clock requests an interrupt if the Interrupt Mask bit is set.

The clock has device code '20 and uses seven OCP instructions, two INAs and one SKS. The Interrupt Mask bit is controlled both by OCPs and by interrupt priority mask bit 16.

OCP 'F20 Output Control Pulse to Clock **030-20 031-20**



Perform the function specified by *F* in the clock as follows.

OCP 0020 Start—start the clock if it is not already running, also acknowledge an interrupt request (equivalent to resetting a ready flag). This instruction does not disturb the clock count if it is already counting.

'0020 Start—start the clock if it is not already running, also acknowledge an interrupt request (equivalent to resetting a ready flag). This instruction does not disturb the clock count if it is already counting.

'0220 Stop—turn off the clock interrupt request and stop the clock.

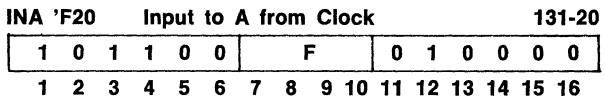
'0420 Select Line—select the ac line frequency as the c ock source.

'0520 Select Alternate Source.

'1520 Set Interrupt Mask bit

'1620 Reset Interrupt Mask bit

'1720 Initialize—stop the clock, turn off the clock interrupt request, and select the ac line as the clock source.



Read the information specified by *F* from the clock into A as follows.

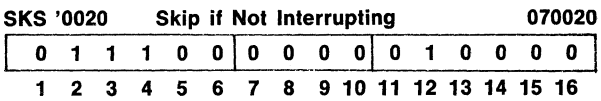
INA '1120 Input Identification—read zeros into A bits 1-3, the slot in which te clock is mounted into A bits 4-8, and the device address code '20 into A bits 9-16.

'1320 Input Status—read the status of the clock into A bits 13-15 as follows.

13 Clock On—a 1 indicates the clock is running.

14 Source—o, ac line; 1, alternate

15 Interrupt Enable.



If the clock is not now requesting an interrupt, skip the next instruction in sequence.

When the clock is turned on, the first count can come at any time up to the clock period. Counting location '61 is done by a memory increment function whose priority lies between that of direct memory access and the external interrupt; at each pulse from the clock source, the clock increments memory after the current instruction and all pending direct memory access are finished, but before the processor handles any interrupt. When the counter overflows, the clock requests an interrupt if the Interrupt Mask bit is set. Once one count has occurred, further counting is steady and is not disturbed by external interrupts or by the action of enable-and inhibit-interrupt instructions.

The ac line clock is used primarily for low-resolution timing (compared to processor speed) but it has high long-term accuracy. Turning on power or clearing the computer from the control panel automatically selects the ac line as the source. A full count of 2^{16} at 60 Hz takes 18.4 minutes.

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

$$T(C + nI)$$

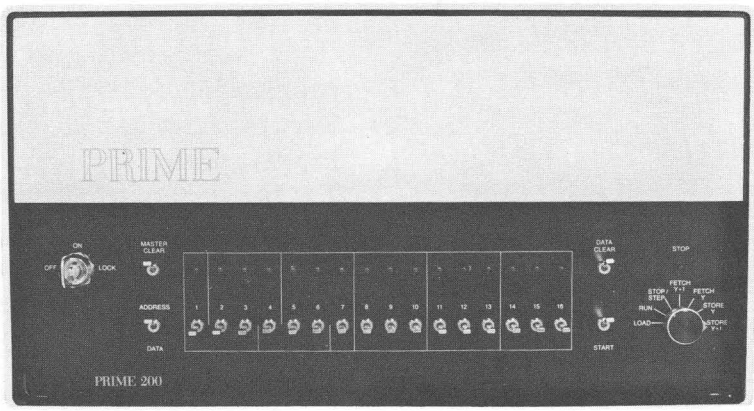
where *T* is the period of the source, *n*, is the number of clock interrupts since the clock was started, and *I* is the interval count selected by the program at each interrupt. To cause the clock to request an interrupt after $T \times n$ ms, where $n \leq 2^{16}$ and *T* is the period of the source in milliseconds, load location '61 with *-n* expreessed in binary. There is an average indeterminacy of half a count every time the clock starts and stops.

For keeping the time of day, the program can use a memory location to keep 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 location '61. If the other location is to be used by itself as a low resolution clock kept in hours, minutes, and perhaps seconds, the programmer should pick an appropriate interval setting the counter each time to '177704 produces an interrupt every second.

2.15 OPERATION

At the left on the control panel is a three-position key-operated rotary switch that controls power and locks the console. Turning it to ON turns on power and generates master clear *(see below)*. Turning it to LOCK keeps power on and disables the operating switches so no one can interfere with the operation of the processor (the operator can still use the data and sense switches in conjunction with a program). When power is on and within specifications, the microprogram is running even

though the program may not be; and the processor is in the stop or run state as incicated by whether or not the STOP light at the upper right is on (the light blinks when memory refresh is running on the optional backup battery). With the processor in the run state, the microprogram executes the program, *i.e.*, it performs instructions, handles interrupts, etc.; in the stop state the microprogram executes a control panel routine, which monitors the panel in order to respond to action by the operator.



The operator determines the state of the processor principally by the two switches at the lower right: a six-position rotary function selector and a momentary-contact START switch. The selector does not itself initiate any operations, but rather selects a function to be performed, and the microprogram performs that function in response to the operator pressing START. Turning the selector to either of the left two positions (LOAD, RUN) and pressing START places the processor in the run state, turning off the STOP light; if the processor halts for any reason, it is then in the stop state even if the selector is still in RUN or LOAD (which we shall refer to as the run positions). Turning the selector to any of the right five positions (the stop positions) normally halts the processor, returning it to the stop state.

Power turnon always generates the master clear, and this function can be produced by the operator pressing the switch at the upper left, provided the panel is not locked and the function selector is in a stop position. The master clear initializes the control panel, the processor and the I/O interfaces (resetting various critical control circuits); clears C and the register file except for P (location '7), which it sets to '1000 and displays in the address lights; and places the processor in the stop state, out of machine check mode, in 16K sectored addressing mode with indexing from X, in single precision arithmetic mode, and in standard interrupt mode with external interrupts inhibited. If the processor has the optional verification routine, master clear executes that routine and reiterates it as long as any failure continues (the number of the failing test is displayed in the address lights instead of P if the function selector is in or is turned to a run position).

In the middle of the panel are a row of lights and a register of switches. The former is address or data lights depending upon the position of the toggle switch at the lower left. The register switches have three positions: a locking up position, a neutral center position, and a momentary down position. The up positions are used only as sense switches with a program or the automatic program load; the down positions may be used as either data or address switches again depending upon the position of the same toggle that defines the lights. However the exact meaning of the switches and the lights depends on which state the processor is in, so we shall discuss the operation of the control panel for these two states separately.

Run State

With the processor in the run state, a program actually running, and the function selector in a run position, the address lights display the contents of the memory address register and the data lights display the contents of the control panel data register. In ordinary circumstances the data register holds information loaded into it by an OTA '1720, but it may also be loaded by the microprogram as discussed below. The memory address register is used for supplying addresses to memory, but it is also used to hold all I/O instructions and some of the generic op codes (if the program is very heavy in I/O, the operator will actually be able to see such instructions in the lights). In any event the information displayed in the address lights is superimposed on a dim glow in the right five lights caused by the refresh logic.

The up positions of the switch register are used for sense switches: any switch that is locked up can be read as a 1 into the corresponding bit of the A register by an INA '1620 or tested as set by instructions in the skip group. The down positions of the switches have the same meaning regardless of the setting of ADDRESS/DATA: a switch held down can be read as a 1 into the A register by an INA '1720 (the programmer must realize that there is contact bounce associated with pressing a switch down and should compensate accordingly).

The two run-state functions that can be performed by setting the function selector and pressing START are as follows.

RUN Turn off STOP and begin normal program execution in the run state beginning at the location presently addressed by P.

While the processor is running, it can be returned to the stop state by the program giving an HLT, by the operator turning the selector to a stop position, by the microprogram encountering a zero vector in an internal interrupt location, or by a machine check error in a computer that lacks verification capability. When the program halts, turning the selector to a stop position displays P in the address lights with the next instruction in the data lig ts. Turning the selector to a stop position displays P in the address lights with the next instruction in the data lights. A halt by a zero vector is the same as a program halt, but the "next instruction" is the contents of X (location 0). A machine check halt sets the memory address register to all 1s.

If the verification routine is executed and finds a consistent error, there is no further program execution, and the number of the failing test is displayed in the address lights.

If turning the selector to a stop position does not turn on STOP, then the processor is hung up in the run state, and the address and data lights continue to display the same information they do when the program is running. The hang-up may be an infinite indirect address chain, continuous direct memory access, a missing memory vector which points to missing memory, or the verification routine encountering a solid machine check error. When the processor is hung up it does not monitor the selector; however the machine can always be stopped by turning the selector to a stop position and pressing MASTER CLEAR (unless of course the panel is locked).

LOAD In a processor without the optional program load, this function is a no-op. Otherwise it turns off STOP and performs the opera-

tion selected by the contents of sense switches 14-16 as follows.

0 Begin normal program execution at location 1000.
1 Execute the program load microroutine for the ASR reader.
2 Execute the program load microroutine for the high speed paper tape reader.
3 Execute the program load microroutine for the fixed-head disk.
4 Execute the program load microroutine for the moving-head disk.

Once this function has begun it is indistinguishable from RUN.

Stop State

The processor enters the stop state when it halts even if the function selector remains in a run position; and unless there is a hangup, it goes to the stop state when the selector is turned to a stop position. Moreover the functions other than RUN and LOAD are executed in their entirety with the processor remaining in the stop state. While the operator is using the stop-state functions, the address lights always display the contents of the memory address register and the data lights display the contents of the memory data register (in both cases the display is actually done by the microprogram through the control panel data register). The up positions of the switches can still be used as sense switches but have meaning only as read by instructions in a program—in the stop state a program can be executed only in single step. Pressing a switch down sets the corresponding bit of the memory address register or memory data register depending on the setting of ADDRESS/DATA; the DATA CLEAR switch resets the register selected by the ADDRESS/DATA switch.

The five stop-state functions that can be performed by setting the function selector and pressing START are as follows.

STOP/STEP Begin program execution at the location specified by P, but with STOP on so the processor halts at the completion of the first instruction. With the selector in this position, the operator can single step through a program to debug it; but the processor handles no interrupts and the program cannot display information in the data lights (information loaded by an OTA '1720 is lost as soon as the instruction is finished).

FETCH THIS Load the data register with the contents of the location specified by the address register.

FETCH NEXT Add 1 to the contents of the address register, and load the data register with the contents of the location specified by the incremented address.

STORE THIS Store the contents of the data register in the location specified by the address register.

STORE NEXT Add 1 to the contents of the address register, and store the contents of the data register in the location specified by the incremented address.

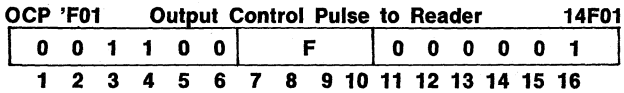
CHAPTER 3
BASIC PERIPHERAL DEVICES

This chapter discusses the basic peripheral devices: teletypewriter, paper tape reader and paper tape punch.

The program can type out characters on the teletypewriter and can read characters that have been typed at the keyboard. This device has a relatively slow transfer rate, but it provides a convenient means of man-machine interaction. KSR teletypewriters are equipped with a keyboard and printer; ASR models also have a low-speed reader and punch. The controllers for the teletypewriters, high-speed reader and punch, and real-time clock are all mounted on a single etched circuit board. Although this controller board is usually refered to as the teletypewriter controller, it is actually a flexible asynchronous serial line controller having four multiplexed input/output ports, eight program-selectable baud rates, and facilities for handling control signals associated with a data communication lines following EIA RS 232C conventions.

3.1 PAPER TAPE READER

The high-speed paper tape reader operates at speeds up to 200 characters per second and is available either separately or combined with a high-speed punch in a single unit. The controller has an eight-bit buffer to hold a complete eight-bit character read from tape. The INA instruction is used to transfer the character from the buffer to the computer. The reader device code is '01, and the controller interrupts through location '101 in vectored mode (in relation to the other controllers on the same board, the reader is of second-highest priority). Interrupt Mask enable is controlled by OCP instructions and by bit 9 in an SMK '20 instruction.



Perform the function *F* in the reader as follows:

OCP '0001 Start—reset Ready and move tape in the selected direction to read one character from tape into the reader buffer.

'0101 Stop—stop tape motion.

'0201 Forward—select the forward tape direction. Note: give this function *only* when the tape is stopped.

'0301 Reverse—select the reverse tape direction. Note: give this function *only* when the tape is stopped.

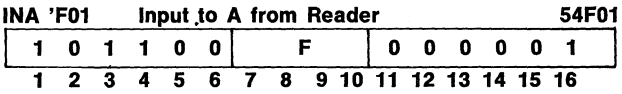
'1201 Normal Mode—return the reader and punch to normal operating mode (see function 13).

'1301 Diagnostic Mode—place the reader and punch controllers in diagnostic mode, which is discussed below.

'1501 Set Interrupt Mask.

'1601 Reset Interrupt Mask.

'1701 Initialize—stop tape motion, select the forward direction, reset Interrupt Mask, and place the controller in normal mode.



If Ready is reset, do nothing but go on to the next instruction. If Ready is set, perform the function specified by *F* as given below, and skip the next instruction in sequence. An asterisk indicates a function for which the controller simulates Ready being set so the function and skip always occur.

INA '0001 Input Data—OR the contents of the reader buffer into A bits 9-16, reset Ready and start the reader to read the next character.

'1001 Input Data—clear A and then perform function 00.

'1101* Input Identification—read the slot number in which the reader controller is mounted into A bits 4-8 and the octal identification code X01 into A bits 9-16. Zeros are read into bits 1-3.

'1301* Input Status—read the status of the reader into A bits 9-16 as follows.

Diagnostic Mode	Run	Forward	Interrupt Mask	Power OK	Data Bit 3	Data Bit 2	Data Bit 1
9	10	11	12	13	14	15	16

A 1 in any bit position indicates the condition is "true" or "set". A 1 in bit 13 indicates reader power is on and all voltages are at operating levels. Bits 1-8 of A are always zeros.

SKS 'C01 Skip if Reader Condition Satisfied 34C01

0	1	1	1	0	0	C				0	0	0	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Skip the next instruction in sequence if the reader condition specified by C is satisfied as follows.

SKS '0001 Ready

'0401 Not Interrupting

The reader comes on in the ready state, so if it is not to be used the program should give an Initialize (OCP '1701) to prevent its requesting an interrupt. To read an initial character from tape, give an OCP '0001. Starting the reader causes it to read all eight channels from the next frame on tape into the buffer (the presence of a hole produces a 1 in the buffer). When the operation is complete, the controller sets Ready, requesting an interrupt if the Interrupt Mask bit is set. When the character is brought into A, tape channel 1 corresponds to A bit 16. Giving an INA '0001 or INA '1001 not only reads the buffer, but also resets Ready and starts the reader for reading the next frame.

To allow the computer to stop the reader on the character read, the interface allows up to one millisecond to issue an OCP '101 stop command. If this command is issued within one millisecond from the time the Ready flag is set, the reader will stop on the character just read. If the one millisecond interval is exceeded, the reader advances to the next frame and waits for an INA instruction. This feature prevents loss of data regardless of the time the processor takes to service the reader, and allows the program one millisecond to input data while maintaining the maximum 200 character per second transfer rate.

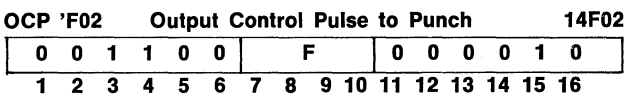
Diagnostic Mode. An OCP '1301 places the controllers for both the reader and punch in a diagnostic mode, in which both devices are disconnected from their controllers, the punch buffer output is connected to the reader buffer input, and internal timing replaces the signals normally supplied by the devices. With this arrangement an OTA '0002, which loads a character into the punch buffer and resets Punch Ready, also causes the controller to "punch" the character into the reader buffer, setting Reader Ready to request an interrupt if Reader Interrupt Mask bit is set. Giving an INA '1001 not only retrieves the character, resetting Reader Ready, but also sets Punch Ready. Status registers may be tested as in normal operation.

Operation. On the combination unit, the controls for the reader are the left four of the five buttons at the upper left of the front panel. The leftmost button controls power to the reader/punch combination. The second and third buttons slew tape through the reader at high speed in the direction indicated by the arrows on the buttons. The fourth button is labeled LOAD and is used to prevent the controller from operating the reader (button will be

illuminated) when a tape is being loaded. Detailed operating procedures for this unit will be found in the PRIME 200 OPERATOR'S GUIDE.

3.2 PAPER TAPE PUNCH

The punch perforates 8-channel tape at speeds up to 75 characters per second; each character is supplied to the punch from an 8-bit buffer in the controller. The punch device code is 02 and the controller interrupts through location '102 in vectored mode (in relation to the other controllers on the same board, the punch is of third highest priority). Interrupt Mask is controlled by OCPs and interrupt mask bit 10.



Perform the function specified by *F* in the punch as follows.

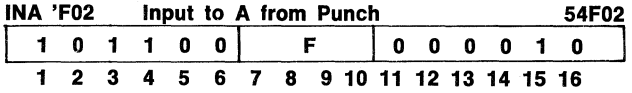
OCP '0002 Enable Punch—enable Ready to request an interrupt if Interrupt Mask bit is set. Note: this function is independent of Interrupt Mask and must be given to allow Ready to interrupt even when the Interrupt Mask bit is set.

'0102 Disable Punch—Reset punch Ready. The punch will no longer accept data from the computer.

'1502 Set Interrupt Mask. This allows an interrupt request to be generated when the punch is ready.

'1602 Reset Interrupt Mask. Inhibit an interrupt request if the punch is ready.

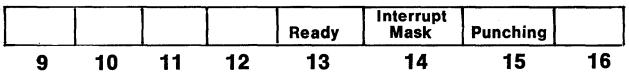
'1702 Initialize—stop the punch (making it ready) and reset Interrupt Mask.



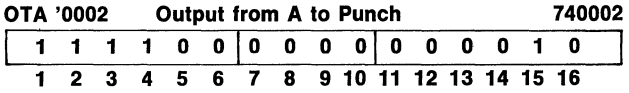
Perform the function specified by *F* as given below, and skip the next instruction in sequence (the controller simulates Ready being set for these functions).

INA '1102 Input Identification—read zeros into A bits 1-3, read the slot number in which the punch controller is mounted into A bits 4-8 and the octal identification code X02 into A bits 9-16.

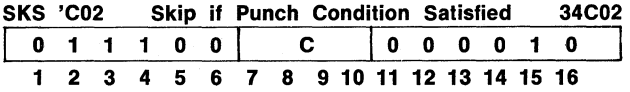
'1302 Input Status—read 0's into A1-A12, and the status of the punch into A bits 13-15 as follows.



A 1 in bit 15 indicates the punch is busy punching the contents of the buffer into tape.



If Ready is reset, execute the next instruction in sequence. If Ready is set, reset it, load the contents of A bits 9-16 into the punch buffer, start the punch to punch the contents of the buffer into the next frame position in the tape, and skip the next instruction in sequence.



Skip the next instruction in sequence if the punch condition specified by C is satisfied as follows.

SKS '0002 Ready

'0102 Power On

'0402 Not Interrupting

The punch comes on in the Ready state, but it cannot request an interrupt unless it is enabled by an OCP '0002, so the programmer can leave the Interrupt Mask bit

set without worrying about interrupts while the punch is not in use. Giving an OTA '0002 both supplies a character from A to the buffer (A bit 16 corresponds to tape channel 1) and starts the device to punch the contents of the buffer (a 1 produces a hole in the tape). After punching is complete, the device sets Ready, requesting an interrupt if the Interrupt Mask bit is set and the punch has been enabled.

Timing. At 75 frames per second the punch takes 13.3 ms per character. After Ready sets, the program has about .3 ms to give a new OTA to keep punching at the maximum rate.

NOTE

The punch controller has a diagnostic mode that is controlled in conjunction with the diagnostic mode for the reader controller, *i.e.*, both controllers are put into or taken out of diagnostic mode together by the same reader OCP instructions as described in the preceding section. It is important to remember that the punch device will not punch characters into tape if it has been left in diagnostic mode by a reader OCP. (OCP '1301).

3.3 TELETYPEWRITER

Communication with the console teletypewriter is handled by an asynchronous serial line interface, which can handle any EIA device or communication line at various bit frequencies. The interface is capable of full-duplex operation, *i.e.*, data can be transferred in both directions at the same time; it actually acts like two interfaces that share a common device code, Ready flag and Interrupt Mask bit (there are also separate Ready flags for transmitter and receiver). The program can select from among eight operating frequencies, (110 baud to 9600 baud), but at any given time reception and transmission must be at the same rate. Characters can be received from any one of four devices at a time; ports 2, 3 and 4 handle only EIA RS 232-C signals, but port 1 handles either EIA signals or a device using a 20 mA current loop. All four output ports handle signals of either type, and the single output data stream can be transmitted over any set of ports simultaneously. The character format in all cases is a single start bit, eight data bits, and one or two stop bits (as specified by the program). There are also four input control lines that can be sensed by the program and four output control lines that can be set by the program to provide control signals at the four device connectors. These control lines may be used for any purpose, such as handling a full-dress communication channel with automatic answering of incoming calls. The program can also select half-duplex operation in which characters transmitted are automatically sent back in through the receiver, and all characters received are simultaneously retransmitted.

The four teletypewriter models regularly available for use with the computer are the ASR33, ASR35, KSR33 and KSR35, all of which are capable of speeds up to ten characters per second and use a dc current loop (port 1). The program can output characters to the page printer and can read in the characters from the keyboard. With an ASR the program can also punch characters in a tape and read characters from a tape.

The entire ASCII character set is listed in Appendix D. Character codes received from the keyboard have eight bits wherein the most significant is always 1. Lower case characters are not available on the Model 33 or 35, but transmitting a lower case code to the teletypewriter causes it to print the corresponding upper case character. There are, of course, no restrictions on the codes that can be punched in or read from tape.

The input and the output parts of the interface each have a character buffer separate from the shift register used to receive or

transmit a character, so the program has an entire character time to respond to Ready at any given bit frequency. The device code is 04 and the controller interrupts through location 104 in vectored mode (in relation to the other controllers on the same board, the serial interface is of highest priority). Interrupt Enable is controlled by interrupt mask bit 11.

OCP 'F04 Output Control Pulse to Serial Interface **14F04**

0	0	1	1	0	0	F				0	0	0	1	0	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Perform the function specified by *F* in the serial interface as follows.

OCP '0004 Select Half-Duplex Input—set up the interface for half-duplex reception at 110 baud with two stop bits, using receiver port 1 and transmitter port 1. Note: give this function *only* when the interface is not busy. Do not attempt to transmit while the interface is in this mode as transmission may be garbled by input being retransmitted. Completion of transmission does not request an interrupt.

'0104 Select Half-Duplex Output—set up the interface for half-duplex transmission at 110 baud with two stop bits, using transmitter port 1 and receiver port 1. Note: give this function *only* when the interface is not busy. Completion of reception does not request an interrupt.

'1004 Select Full-Duplex—restore the interface to full-duplex operation. Note: give this function *only* when the interface is not busy.

'1204 Normal Mode—return the interface to normal operating mode (see function 13).

'1304 Diagnostic Mode—place the interface in full-duplex operation where all devices are disconnected and the transmitter output is fed into the receiver.

'1504 Set Interrupt Mask bit.

'1604 Reset Interrupt Mask bit.

'1704 Initialize—reset Interrupt Mask bit, reset all error and ready flags, reset all output control flags, and set up the interface in normal mode for half-duplex input (as per function 00).

OTA 'F04 Output from A to Serial Interface **74F04**

1	1	1	1	0	0	F				0	0	0	1	0	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

If Transmitter Ready is reset, do nothing but go on to the next instruction. If Transmitter Ready is set, perform the function specified by *F* as given below, and skip the next instruction in sequence. An asterisk indicates a function for which the controller simulates Transmitter Ready being set so the function and skip always occur.

OTA '0004 Output Data—load the contents of A bits 9-16 into the output buffer; reset Ready and Transmitter Ready.

'0104* Output Control Word—set up the output control signals and select the operating characteristics of the interface according to the contents of A as follows.

OUTPUT CONTROL SIGNALS				FULL DUPLEX	CLOCK FREQUENCY		
1	2	3	4				
1	2	3	4	5	6	7	8

	2 STOP BITS	RECEIVER PORT		TRANSMITTER PORTS			
				1	2	3	4
9	10	11	12	13	14	15	16

1-4 1s turn on the indicated control signals; 0s turn them off.

5 1 selects full-duplex, 0 selects half-duplex.

6-8 These bits select the clock frequency as follows.

Bits 6-8	*Frequency (Hz)*
000	110
001	150
010	300
011	600
100	1200
101	2400
110	4800
111	9600

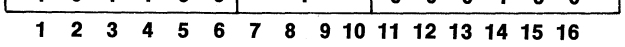
10 1 selects character format with two stop bits; 0 selects one stop bit.

11-12 These bits select the single receiver port as follows.

Bits 11-12	*Receiver Port*
00	1
01	2
10	3
11	4

13-16 1s select transmitter ports as indicated.

INA 'F04 Input to A from Serial Interface 54F04



If Receiver Ready is reset, do nothing but go on to the next instruction. If Receiver Ready is set, perform the function specified by *F* as given below, and skip the next instruction in sequence. An asterisk indicates a function for which the controller simulates Receiver Ready being set so the function and skip always occur.

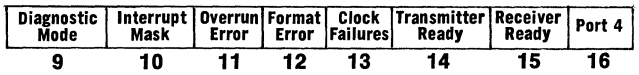
INA '0004 Input Data—OR the contents of the input buffer into A bits 9-16; reset Ready and Receiver Ready.

'0104 Input Data—clear A and then perform function 00.

'1104* Input Identification—read the slot number in which the serial interface is mounted into A bits 4-8 and the octal identification code X04 into A bits 9-16.

'1204* Input Setup Status—read the status of the output control signals and operating characteristics of the interface into A. The status bits read have the same meaning as the control bits sent out by an OTA '0104; hence this status reflects the way the interface has been set up by an OTA '0104 and the OCP functions 00, 01, 10, 13 and 17.

'1304* Input Flag Status—clear A bits 1-8 and read the status of the mode, error and ready flags into A bits 9-16 as follows.

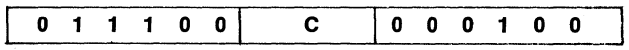


11 Reception of a character was completed before the program retrieved one or more previously received characters from the input buffer.

12 The interface has received a character with the wrong number of data or stop bits.

16 This bit is for diagnostic purposes only.

SKS 'C04 Skip if Serial Interface Condition Satisfied 34C04



Skip the next instruction in sequence if the interface condition specified by C is satisfied as follows.

SKS '0004 Ready.

'0104 Busy—the interface is busy while a character is being transmitted or received.

'0404 Not Interrupting

'0604 Receiver Ready

'0704 Transmitter Ready

'1104 Input Control Signal 1

'1204	Input Control Signal 2
'1304	Input Control Signal 3
'1404	Input Control Signal 4

When the transmitter is inactive, loading a character into the output buffer causes the transmitter to take the character and begin sending it through the selected output ports. When the transmitter is already sending a character, a new character loaded into the output buffer must wait until the previous transmission is complete, at which time the transmitter automatically takes the waiting character. In any event movement of a character from buffer to transmitter sets Transmitter Ready and Ready, the latter requesting an interrupt if the Interrupt Mask bit is set; the program may then send a new character while the transmitter is busy with the previous one. To each data character the transmitter adds a start bit and one or two stop bits as selected by the program, and sends out the full character serially at the selected bit frequency. Transmitting a code to the teletypewriter causes it to print the character or perform the indicated control function. If the punch is on, the character is also punched in the tape with A bit 16 corresponding to channel 1 (a 1 in A produces a hole in the tape).

Reception from the keyboard requires no initiating action by the program; striking a key transmits the code for the character serially to the receiver. As soon as the receiver assembles a character, it sends the data part to the input buffer and sets Receiver Ready and Ready, the latter requesting an interrupt if the Interrupt Mask bit is set. The receiver is then free for more reception while waiting for the program to take the character from the buffer. Should a new character be received before the program has taken the previous one, the new is loaded into the buffer destroying the old and setting the Overrun flag. An ASR tape reader is usually set up so that the operator must make it read and send the first character; but with the reader on, each subsequent INA that takes a character from the input buffer also causes the reader to read another. The presence of a hole in the tape produces a 1 in the buffer, and when the character is brought into A, tape channel 1 corresponds to A bit 16.

The program has no inherent control over incoming information, so when responding to Ready the program should check Receiver Ready first, giving priority to the receiver in case the interface is ready for transfers in both directions simultaneously. When the program is making a transfer and the ready flag for the other direction is on or comes on, the INA or OTA resets the ready flag for the direction being handled but not the main Ready, which is held set to request a subsequent interrupt for the other direction.

Timing. Although characters contain eight data bits, the character rate is actually one-tenth or one-eleventh the bit rate depending upon whether there is one or two stop bits. Because of the double buffering on both sides, the program has a full character time at the selected bit rate (ten or eleven times the bit period) to respond to a ready indication without fear of losing received data or running the transmitter below maximum rate.

The Model 33 and 35 teletypewriters operate at ten characters per second, 100 ms per character. The sequence carriage return-line feed, when given in that order, allows sufficient time for the type block to get to the beginning of a new line. After tabbing, the program must wait for completion of the mechanical function by sending one or two rubouts. If the time is critical, the programmer should measure the time required for his tabs. Tabs are normally set every eight spaces (columns 9, 17, . . .) and require one rubout.

APPENDIX A
TWOS COMPLEMENT CONVENTIONS

The signed numbers used as relative displacements in referencing memory and as operands for the arithmetic instructions utilize the twos complement representation for negatives. In a word or byte used as a signed number, the leftmost bit represents the sign, 0 for positive, 1 for negative. In a positive number the remaining bits are the magnitude in ordinary binary notation. The negative of a number is obtained by taking its twos complement, with the sign bit included in the operation as though it were a more significant magnitude bit. If x is an n-digit binary number, its twos complement is 2^n-x, and its ones complement is $(2^n -1)$-x, or equivalently $(2^n$-$x)$-1. Subtracting a number from 2^n-1 (*i.e.*, from all 1s) is equivalent to performing the logical complement, *i.e.*, changing all 0s to 1s and all 1s to 0s. Therefore, to form the twos complement one takes the logical complement—usually referred to simply as the complement—of the entire word including the sign, and adds 1 to the result. A displacement of 173 and its negative would look like this in bits 8-16 of an instruction word where bit 8 is the sign.

$+173_{10} = +255_8 =$ | | | | | | | | 010 | 101 | 101 |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | 8 | | 16 |

$+173_{10} = -255_8 =$ | | | | | | | | 101 | 010 | 011 |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | 8 | | 16 |

The same numbers used as operands in memory or the A register would look like this.

$-173_{10} = +255_8 =$ | 0 | 000 | 000 | 010 | 101 | 101 |
|---|---|---|---|---|---|

$-173_{10} = -255_8 =$ | 1 | 111 | 111 | 101 | 010 | 011 |
|---|---|---|---|---|---|
| 1 | | | | | 16 |

Bit 1 is now the sign and bits 2-7 are not significant. It is thus evident that expanding an integer into a full word is accomplished simply by filling out the word to the left with the sign.

The arithmetic instructions manipulate operands as 16-bit unsigned numbers, but the program can interpret them as signed numbers in twos complement notation. It is a property of twos complement arithmetic that operations on signed numbers using twos complement conventions are identical to operations on unsigned numbers; in other words the hardware simply treats the sign as a more significant magnitude bit (although overflow is detected as though the numbers were signed). Regarding the above 16-bit examples as unsigned numbers, the positive form would still represent 173, but the negative form now represents 65,363 ('177523). Insofar as processor operations are concerned, it makes no difference which way the programmer interprets the contents of registers provided only that he is consistent.

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 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 as a 16-bit number is 100000 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). Assuming the binary point to be stationary, 1s may be discarded at the left in a negative integer, just as leading 0s may be dropped in a positive integer; equivalently an integer can be extended to the left by prefixing 1s or 0s respectively (*i.e.*, by prefixing the sign). In a negative (proper) fraction, 0s may be discarded at the right; as 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. Truncation of a negative number thus increases its absolute value. 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.

Since each bit position represents a binary order of magnitude, shifting a number is equivalent to multiplication by a power of 2, provided of course that the binary point is assumed stationary. Shifting one place to the left multiplies the number by 2. A 0 should be entered at the right, and no information is lost if the sign bit remains the same—a change in the sign indicates that a bit of significance has been shifted out. Shifting one place to the right divides by 2. Truncation occurs at the right, and a bit equal to the sign must be entered at the left.

APPENDIX B
ADDRESSING

P Address of instruction location (contents of Program counter before instruction fetch)
P|D Sectored address formed by concatenation of the left seven bits of P with the right nine bits of D
A For standard addressing: an absolute address of 14 or 15 bits; for extended addressing: specifically the 15-bit absolute address in location P + 1 (bit 1 is ignored)
S Contents of stack register
X Contents of currently selected index register
I(ξ) Result of indirect chain beginning with access to locations addressed by ξ

16K Sectored $0 \leqslant D \leqslant$ '777

I	X	S	D	Address Word	EA
0	0	0			D
0	1	0			D + X
1	0	0		I,X,A	I(D)
1	1	0		I,X,A	I(D +X)
0	0	1			P|D
0	1	1			P|D + X
1	0	1		I,X,A	I(P|D)
1	1	1		I,X,A	I(P|D + X)

32K Sectored $0 \leqslant D \leqslant$ '777

0	0	0			D
0	1	0			D + X
1	0	0		I,A	I(D)
1	1	0	$<$ '100	I,A	I(D + X)
1	1	0	$\geqslant$ '100	I,A	I(D) + X
0	0	1			P|D
0	1	1			P|D + X
1	0	1		I,A	I(P|D)
1	1	1		I,A	I(P|D) + X

32K Relative $S = 0$: $0 \leqslant D \leqslant$ '777
$S = 1$: $-240 \leqslant D \leqslant 255$
[for $D < -240$ see § 2.9]

0	0	0			D
0	1	0			D + X
1	0	0		I,A	I(D)
1	1	0	$<$ '100	I,A	I(D + X)
1	1	0	$\geqslant$ '100	I,A	I(D) + X
0	0	1	$\geqslant -240$		P + 1 + D
0	1	1	$\geqslant -240$		P + 1 + D + X
1	0	1	$\geqslant -240$	I,A	I(P + 1 + D)
1	1	1	$\geqslant -240$	I,A	I(P + 1 + D) + X

32K Relative: Extended Effective Address Calculation
Bits 7-12 = 1 100 00 ($S = 1$, $-256 \leqslant D \leqslant -241$)

I	X	Bits 15-16	EA	Ancillary Action	Type of Addressing
0	0	0	A		Address
0	0	1	A + S		Base plus displacement
0	0	2	S	S + 1 → S	Push/pop
0	0	3	S − 1	S − 1 → S	Pop/push
0	1	0	A + X		Address, indexed
0	1	1	A + S + X		Base plus displacement, indexed
0	1	2	I(S) + X	S + 1 → S	Push/pop indirect, postindexed
0	1	3	I(S − 1) + X	S − 1 → S	Pop/push indirect, postindexed
1	0	0	I(A)		Address indirect
1	0	1	I(A + S)		Base plus displacement, indirect
1	0	2	I(S)	S + 1 S	Push/pop indirect
1	0	3	I(S − I)	S − 1 S	Pop/push indirect
1	1	0	I(A + X)		Address indexed, indirect
1	1	1	I(A + S + X)		Base plus displacement indexed, indirect
1	1	2	I(A) + X		Address indirect, postindexed
1	1	3	I(A + S) + X		Base plus displacement indirect, postindexed

APPENDIX C
INSTRUCTION MNEMONICS AND TIMING

The table beginning on this page lists the instruction mnemonics in alphabetical order, and for each gives the octal op code, the effect of the instruction, the execution time in microseconds, and the number of the page on which the full description appears in Chapter 2. For skip instructions two or three times are given, where the first is the time for no skip, the second for a single skip, and the third (where applicable) for a double skip. The time for a shift instruction is a function of the number of shifts *N*. Following the alphabetic listing is a table of the mnemonics in numerical order.

			Time	Page
A1A	141206	= AOA	1.36	2-8
A2A	140304	Add two to A	1.36	2-8
ACA	141216	Add C to A	1.36	2-8
ADD	06	Add C(E) to A	1.96	2-7
ALL	041400	A left logical	1.08 + .24*N*	2-5
ALR	041600	A left rotate	1.08 + .24*N*	2-5
ALS	041500	A left shift	1.28 + .24*N*	2-6
ANA	03	AND C(E) to A	1.96	2-7
AOA	141206	Add one to A	1.36	2-8
ARL	040400	A right logical	1.08 + .24*N*	2-5
ARR	040600	A right rotate	1.08 + .24*N*	2-6
ARS	040500	A right shift	1.08 + .24*N*	2-6
CAI	000411	Clear active interrupt	1.28	2-24
CAL	141050	Clear A left	1.36	2-4
CAR	141044	Clear A right	1.36	2-4
CAS	11	Compare A with C(E) and skip	2.64-2.92-3.20	2-12
CAZ	140214	Compare A with zero and skip	1.52/1.80/2.08	2-12
CEA	000111	Compute effective address	2.04	2-3
		Plus for each indirect level	.88	
CHS	140024	Change sign of A	1.36	2-8
CMA	140401	Complement A	1.36	2-6
CRA	140040	Clear A	1.28	2-4
CRB	140014	Clear B	1.28	2-4
CRL	140010	Clear long (A,B)	1.48	2-4
CSA	140320	Copy sign of A to C, set sign plus	1.36	2-8
DAD	06	Double add C(E,E+1) to A,B	3.56	2-9
DBL	000007	Enter double precision mode	1.28	2-9
DIV	17	Divide A,B by C(E)	13.68-14.72[1]	2-10
DLD	02	Double load A,B from E;E+1	2.96	2-9
DRX	140210	Decrement and replace index, skip	2.64	2-12
DSB	07	Double subtract C(E,E+1) from A,B	3.32 *or* 3.6	2-9
		(Either time may result from algorithm)		
DST	04	Double store A,B into E,E+1	3.04	2-9
DXA	000011	= E16S	1.28	2-2
E16S	000011	Enter 16K sectored addressing mode	1.28	2-2
E32R	001013	Enter 32K relative addressing mode	1.28	2-2
E32S	000013	Enter 32K sectored addressing mode	1.28	2-2
EMCM	000503	Enter machine check mode	1.28	2-15
ENB	000401	Enable external interrupts	1.28	2-23
ERA	05	Exclusive OR C(E) to A	1.96	2-7
ESIM	000415	Enter standard interrupt mode	1.28	2-24
EVIM	000417	Enter vectored interrupt mode	1.28	2-24
EXA	000013	= E32S	1.28	2-2
HLT	000000	Halt	1.68	2-14
IAB	000201	Interchange A and B	1.88	2-4
ICA	141340	Interchange bytes of A and clear left	1.36	2-4
ICL	141140	Interchange bytes of A and clear left	1.36	2-4
ICR	141240	Interchange bytes of A and clear right	1.36	2-4
IMA	13	Interchange memory and A	2.88	2-3
IMK	54-20	Input interrupt mask to A	2.94	2-14
INA	54	Input to A		2-19
		Not ready	2.04	
		Ready, OR to A	3.10	
		Ready, clear A	3.38	
INH	001001	Inhibit external interrupts	1.28	2-24
INK	000043	Input keys to A	2.44	2-16
IRS	12	Increment and replace C(E), skip	1.56/1.84	2-12
IRX	140114	Increment and replace index, skip	1.56/1.84	2-12
ISI	000511	Input serial interface to A	1.56	2-21
JMP	01	Jump to E	1.28	2-3
JST	10	Jump and store P in E	2.64	2-3
LDA	02	Load A from E	1.88	2-3
LDX	35	Load index from E	1.88	2-3
LEQ	140413	Logicize A equal to zero	1.54	2-7
LF	140416	Logicize false	1.54	2-7
LGE	140414	Logicize A greater than or equal to zero	1.54	2-7
LGT	140415	Logicize A greater than zero	1.54	2-7
LLE	140411	Logicize A less than or equal to zero	1.54	2-7
LLL	041000	Long left logical	1.08 + .48*N*	2-5
LLR	041200	Long left rotate	1.08 + .68*N*	2-6
LLS	041100	Long left shift	1.28 + .48*N*	2-6
LLT	140410	Logicize A less than zero	1.54	2-7
LMCM	000501	Leave machine check mode	1.28	2-15
LNE	140412	Logicize A not equal to zero	1.54	2-7
LRL	040000	Long right logical	1.08 + .48*N*	2-5
LRR	040200	Long right rotate	1.08 + .68*N*	2-6
LRS	040100	Long right shift	1.08 + .48*N*	2-6
LT	140417	Logicize true	1.54	2-7
MPY	16	Multiply A by C(E)	10.48	2-10
NOP	000001	No-op	1.68	2-14
NRM	000101	Normalize	3.24 + .76*N*	2-11
OCP	14	Output control pulse	2.16	2-18
OTA	74	Output from A		2-19
		Not ready	2.04	
		Ready	3.00	
OTK	000405	Output keys from A	2.12	2-16
OSI	000515	Output serial interface from A	1.28	2-20
PID	000211	Position for integer divide	2.08	2-11
PIM	000205	Position from integer multiply	1.84	2-10
RCB	140200	Reset C bit	1.36	2-14
RMC	000021	Reset machine check	1.48	2-15
RPM	000021	= RMC	1.48	2-14

[1] 2.62 for divide overflow (normal case)

			Time	Page
S1A	140110	= SOA	1.36	2-8
S2A	140310	Subtract two from A	1.36	2-8
SAR	10026n	Skip on A bit n reset	2.04/2.32	2-13
SAS	10126n	Skip on A bit n set	2.04/2.32	2-13
SCA	000041	Load shift count to A	2.16	2-11
SCB	140600	Set C bit	1.36	2-14
SGL	000005	Enter single precision mode	1.28	2-9
SGT	100220	Skip on A greater than zero	2.04/2.32	2-13
SKP	100000	Skip	2.32	2-13
SKS	34	Skip on I/O condition	2.16/2.44	2-19
SLE	101220	Skip on A less than or equal to zero	2.04/2.32	2-13
SLN	101100	Skip on A LSB nonzero	2.04/2.32	2-13
SLZ	100100	Skip on A LSB zero	2.04/2.32	2-13
SMCR	100200	Skip on machine check reset	2.04/2.32	2-13
SMCS	101200	Skip on machine check set	2.04/2.32	2-13
SMI	101400	Skip on A minus	2.04/2.32	2-13
SMK	74XX20	Set interrupt mask from A	2.59	2-24
SNZ	101040	Skip on A nonzero	2.04/2.32	2-13
SOA	140110	Subtract one from A	1.36	2-8
SPL	100400	Skip on A plus	2.04/2.32	2-13
SPN	100200	= SMCR	2.04/2.32	2-13
SPS	101200	= SMCS	2.04/2.32	2-13
SNR	10024n	Skip on sense switch n reset	2.04/2.32	2-14
SR1	100020	Skip on reset sense switch 1	2.04/2.32	2-13
SR2	100010	Skip on reset sense switch 2	2.04/2.32	2-13
SR3	100004	Skip on reset sense switch 3	2.04/2.32	2-13
SR4	100002	Skip on reset sense switch 4	2.04/2.32	2-13
SRC	100001	Skip on reset C bit	2.04/2.32	2-13
SNS	10124n	Skip on sense switch n set	2.04/2.32	2-14
SS1	101020	Skip on set sense switch 1	2.04/2.32	2-13
SS2	101010	Skip on set sense switch 2	2.04/2.32	2-13
SS3	101004	Skip on set sense switch 3	2.04/2.32	2-13
SS4	101002	Skip on set sense switch 4	2.04/2.32	2-13
SSC	101001	Skip on set C bit	2.04/2.32	2-13
SSM	140500	Set sign of A minus	1.36	2-8
SSP	140100	Set sign of A plus	1.36	2-8
SSR	100036	Skip on all of sense switches 1-4 reset	2.04/2.32	2-13
SSS	101036	Skip on any of sense switches 1-4 set	2.04/2.32	2-13
STA	04	Store A in E	1.96	2-3
STX	15	Store index in E	1.96	2-3
SUB	07	Subtract C(E) from A	1.96	2-8
SVC	000505	Supervisor call	3.24	2-14
SZE	100040	Skip on A zero	2.04/2.32	2-13
TCA	140407	Twos complement A	1.64	2-8
VIRY	000311	Verify		2-15
XCA	140104	Transfer A to B and clear A	1.68	2-4
XCB	140204	Transfer B to A and clear B	1.68	2-4
*	100000	Indirect	.88	
,1	040000	Index	—	

NUMERIC LISTING

When the programmer is inspecting the lights to determine what instruction is in a particular memory location, he should first interpret bits 1-6 as a pair of octal digits for identification of memory reference and IO instructions. If lights 3-6 are all off, the instruction is in the generic class and the whole word should then be interpreted as six octal digits.

Memory Reference and Input-Output

00/20/40/60	*Generic*	10/30/50/70	JST/,1/*/* ,1
01/21/41/61	JMP/,1/*/* ,1	11/31/51/71	CAS/,1/*/* ,1
02/22/42/62	LDA/,1/*/* ,1	12/32/52/72	IRS/,1/*/* ,1
02/22/42/62	DLD/,1/*/* ,1	13/33/53/73	IMA/,1/*/* ,1
03/23/43/63	ANA/,1/*/* ,1	14/34/54/74	*Input-output*
04/24/44/64	STA/,1/*/* ,1		14 OCP
04/24/44/64	DST/,1/*/* ,1		34 SKS
05/25/45/65	ERA/,1/*/* ,1		54 INA
			74 OTA
06/26/46/66	ADD/,1/*/* ,1	15/55	STX/*
06/26/46/66	DAD/,1/*/* ,1	35/75	LDX/*
07/27/47/67	SUB/,1/*/* ,1	16/36/56/76	MPY/,1/*/* ,1
07/27/47/67	DSB/,1/*/* ,1	17/37/57/77	DIV/,1/*/* ,1

Generic

000000	HLT	040500	ARS	101400	SMI
000001	NOP	040600	ARR	140010	CRL
000005	SGL	041000	LLL	140014	CRB
000007	DBL	041100	LLS	140024	CHS
000011	E16S	041200	LLR	140040	CRA
000011	DXA	041400	ALL	140100	SSP
000013	E32S	041500	ALS	140104	XCA
000013	EXA	041600	ALR	140110	SOA
		100000	SKP	140110	S1A
		100001	SRC	140114	IRX
000021	RMC	100002	SR4	140200	RCB
000021	RMP	100004	SR3	140204	XCB
000041	SCA	100010	SR2	140206	AOA
000043	INK	100020	SR1	140206	A1A
000101	NRM	100036	SSR	140210	DRX
000111	CEA	100040	SZE	140214	CAZ
000201	IAB	100100	SLZ	140216	ACA
000205	PIM	100200	SMCR	140304	A2A
000211	PID	100200	SPN	140310	S2A
000311	VIRY	100220	SGT	140320	CSA
000401	ENB	100240	SR	140401	CMA
000405	OTK	100260	SAR	140407	TCA
000411	CAI	100400	SPL	140410	LLT
000415	ESIM	101001	SSC	140411	LLE
000417	EVIM	101002	SS4	140412	LNE
000501	EMCM	101004	SS3	140413	LEQ
000503	LMCM	101010	SS2	140414	LGE
000505	ISI	101020	SS1	140415	LGT
000515	OSI	101036	SSS	140416	LF
001001	INH	101040	SNZ	140417	LT
001013	E32R	101100	SLN	140500	SSM
		101200	SMCS	140600	SCB
		101200	SPS	141044	CAR
040000	LRL	101220	SLE	141050	CAL
040100	LRS	101240	SS	141140	ICL
040200	LRR	101260	SAS	141240	ICR
040400	ARL			141340	ICA

APPENDIX D
INPUT-OUTPUT CODES

The following table lists the complete ASCII code, with information pertaining to its use with Teletype Models 33 and 35. The lower case character set (codes 140-176) is not available on these models, but giving one of these codes causes the teletypewriter to print 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 computer teletypewriter, and the definitions bear no necessary relation to the use of the codes in conjunction with the software. Following the ASCII table is a complete listing of the Prime I/O devices with their device and identification codes and mask bit assignments.

8-Bit Octal Code	Character	Remarks
200	NUL	Null, tape feed. Control shift P.
201	SOH	Start of heading; also SOM, start of message. Control A.
202	STX	Start of text; also EOA, end of address. Control B.
203	ETX	End of text; also EOM, end of message. Control C.
204	EOT	End of transmission (END); shuts off TWX machines. Control D.
205	ENQ	Enquiry (ENQRY); also WRU, "Who are you?" Triggers identification (Here is. . .") at remote station if so equipped. Control E.
206	ACK	Acknowledge; also RU, "Are you. . . ?" Control F.
207	BEL	Rings the bell. Control G.
210	BS	Backspace; also FEO, format effector. Backspaces some machines. Repeats on Model 37. Control H.
211	HT	Horizontal tab. Control I.
212	LF	Line feed or line space (NEW LINE); advances paper to next line. Control J.
213	VT	Vertical tab (VTAB). Control K.
214	FF	Form feed to top of next page (PAGE). Control L.
215	CR	Carriage return to beginning of line. Control M.
216	SO	Shift out; changes ribbon color to red. Control N.
217	SI	Shift in; changes ribbon color to black. Control O.
220	DLE	Data link escape. Control P (DC0).
221	DC1	Device control 1, turns transmitter (reader) on. Control Q (X ON).
222	DC2	Device control 2, turns punch or auxiliary on. Control R (TAPE, AUX ON).
223	DC3	Device control 3, turns transmitter (reader) off. Control S (X OFF).
224	DC4	Device control 4, turns punch or auxiliary off. Control T (TAPE, AUX OFF).
225	NAK	Negative acknowledge; also ERR, error. Control U.
226	SYN	Synchronous idle (SYNC). Control V.
227	ETB	End of transmission block; also LEM, logical end of medium. Control W.
230	CAN	Cancel (CANCL). Control X.
231	EM	End of medium. Control Y.
232	SUB	Substitute. Control Z.
233	ESC	Escape, prefix. Control shift K.
234	FS	File separator. Control shift L.
235	GS	Group separator. Control shift M.
236	RS	Record separator. Control shift N.
237	US	Unit separator. Control shift O.
240	SP	Space.
241	!	
242	"	

8-Bit Octal Code	Character	Remarks
243	#	
244	$	
245	%	
246	&	
247	'	Accent acute or apostrophe.
250	(	
251	)	
252	*	
253	+	
254	,	
255	−	
256	.	
257	/	
260	0	
261	1	
262	2	
263	3	
264	4	
265	5	
266	6	
267	7	
270	8	
271	9	
272	:	
273	;	
274	<	
275	=	
276	>	
277	?	
300	@	
301	A	
302	B	
303	C	
304	D	
305	E	
306	F	
307	G	
310	H	
311	I	
312	J	
313	K	
314	L	
315	M	

8-Bit Octal Code	Character
316	N
317	O
320	P
321	Q
322	R
323	S
324	T
325	U
326	V
327	W
330	X
331	Y
332	Z
333	[

334	/	
335	]	
336	△	
337	—	
340	`	Accent grave.
341	a	
342	b	
343	c	
344	d	
345	e	
346	f	
347	g	
350	h	
351	i	
352	j	
353	k	
354	l	
355	m	
356	n	
357	0	
360	p	
361	q	
362	r	
363	s	
364	t	
365	u	
366	v	
367	w	
370	x	
371	y	
372	z	
373		
374	|	
375		
376	~	
377	DEL	

On early versions of the Model 33 and 35, either of these codes may be generated by either the ALT MODE or ESC key.
Delete, rub out.

Keys That Generate No Codes

REPT	Causes any other key that is struck to repeat continuously until REPT is released.
LOC LF	Local line feed.
LOC CR	Local carriage return.
	Opens the line (machine sends a continuous string of null characters).
BRK RLS	Break release (not applicable).
HERE IS	Transmits predetermined 20-character message.

IO DEVICES

Device Code	Identification	Mask Bits	Device
00			
01	X01	9	High speed reader
02	X02	10	High speed punch
03			
04	X04	11	Teletypewriter
05			
06			
07			
10			
11			
12			
13			
14			
15			
16			
17			
20	X20	16	RTC interrupt mask
20			Control panel
21			
22	X22-	4,8	Fixed head disk
23			
24			
25	X25	4,8	Moving head disk
26			
27			
30			
31			
32			
33			
34			
35			
36			
37			
40			
41			
42			
43			
44			
45			
46			
47			
50			
51			
52			
53			
54			
55			
56			
57			
60			
61			
62			
63			
64			
65			
66			
67			
70			
71			
72			
73			
74			
75			
76			
77			

PRIME

PRIME COMPUTER, INC., 17 STRATHMORE ROAD, NATICK, MASSACHUSETTS 01760

Printed in U.S.A.

012