MAN1671

SYSTEM REFERENCE
User Guide

Revision A
July 1975

MAN1671

SYSTEM REFERENCE
User Guide

Revision A
July 1975

PRIME
Computer, Inc.
145 Pennsylvania Ave.
Framingham, Mass. 01701

Performance characteristics are
subject to change without notice.

CONTENTS

CONTENTS (Cont)

ILLUSTRATIONS

List of Tables

SECTION 1

GENERAL DESCRIPTION

INTRODUCTION

All Prime computers share a common, general-purpose, microprogrammed architecture. Within this basic framework are three, program-compatible performance steps: The Prime 100, 200 and 300 central processors. From the 100 to the 300, each processor provides an increasingly powerful combination of standard computation and control features, plus various performance enhancement options that can be selected by the user to satisfy specific application requirements.

A single 16" X 18" circuit board contains the logic for a complete processor, and upgrading the processor performance of a system usually requires little more than unplugging one board and plugging in another. For example, by substituting a Prime 300 board for a 200, the following resources are made available as standard features: virtual memory capability, main memory addressing up to 262,144 16-bit words, hardware multiply/divide as well as over two dozen additional machine-language instructions, expanded direct memory access features supporting a maximum data transfer rate of 1.25M words/second, automatic program loaders for disk, magnetic tape and paper tape, and micro-verification routines. The new processor will work with the original chassis, power supply, peripherals and controllers, even the original processor's main memory. Equally important, the new processor will run all existing software without modification, and will also support virtual memory versions of Prime's disk and real-time operating systems for multi-user configurations. (See the Prime Computer Software Guide for a 48-page summary of all Prime system software, including operating systems, FORTRAN IV, BASIC, Macro Assembler, Micro Assembler, debugging aids, loaders and libraries.)

THE PRIME COMPUTER USER PLAN

To encourage users to take advantage of the ease with which Prime systems can be upgraded and expanded, Prime's unique "Computer User Plan" provides a two-year trade-in policy which permits upgrading of central processors, processor enhancements, chassis, power supplies and memories - all with guaranteed trade-in allowances toward the purchase of new equipment. (This and other features of the Plan are described in a Computer User Plan booklet available from any Prime sales office.)

SUMMARY DESCRIPTION

The following tables summarize standard and optional features and compare the performance characteristics of all processors. Certain processor-related features listed in these charts, such as real-time clocks and watch-dog timers, are available on a separate System Option controller

board and are described in detail in the SOC Users Guide. Peripheral devices, data communication interfaces, and digital and analog input/output systems are described in the appropriate user guides.

Table 1-1. Central Processor Features

Central Processor	100		200		300	
Feature Availability	Std.	Opt	Std.	Opt.	Std.	Opt.
1. Multi-level, vectored priority interrupt system.	●		●		●	
2. Four-channel, bit-serial, full-duplex interface.	●		●		●	
3. Eight channel, programmable DMA system.	●		●		●	
4. Full control panel.	●		●		●	
5. Unimplemented instruction trap.	●		●		●	
6. Illegal instruction trap.	●		●		●	
7. Power supply	●		●		●	
8. High-speed register file (32 addressable 16-bit regs.)	●		●		●	
9. Hardware multiply/divide, and double precision arith.		●		●	●	
10. Extended direct memory access (DMC,DMT).		●		●	●	
11. Automatic program loaders (standard devices).		●		●	●	
12. Processor byte parity.			●		●	
13. Memory byte parity.			●		●	
14. Microverification routines.				●	●	
15. Single-and double recision fl. pt. arith				●		●
16. Virtual memory capability.					●	
17. Stack processing instructions					●	
18. Field exerciser panel.		●		●		●
19. Automatic program loader (custom).		●		●		●
20. Power monitor, with battery backup for MOS mem.		●		●		●
21. Real-time clock (see Product Description 2).		●		●		●
22. Watch-dog timer (see Product Description 2).		●		●		●
23. Interprocessor controller (see Product Description 6).		●		●		●
24. Writable control store.						●

Table 1-2. Performance Comparison

Central Processor	100	200	300
1. Word Size	16 bits	16 bits plus 2 parity bits	16 bits plus 2 parity bits
2. Instruction Size	basic format 16 bits; extended format, 32 bits		
3. Addressing	direct, indexed and indirect in sectored and relative modes; extended (double word format); and stack processing.		
4. Minimum-maximum main memory	4-64K	4-64K	8-256K
5. Memory access time	680 nanosec.	600 nanosec.	600 or 400 nanosec.
6. Memory increment per board	4K,8K,32K,	4K,8K,32K	8K,32K
7. Maximum direct addressing range	64K words per program (K = 1,024)		
8. Maximum virtual memory space per user.	-	-	64K words
9. I/O data path	16 bits	16 bits plus 2 parity bits	16 bits plus 2 parity bits
10. Four-channel, bit-serial interface.		110-9600 baud	
11. Maximum DMT I/O rate	694K words/ sec.	1.0M words/sec.	1.25M words/sec.
12. External interrupts	party line or multi-level vectored priority		
13. Typical interrupt latency	8.2 microsec.	6-8 microsec.	5.4 microsec.
14. Addressable registers in high speed register file	32 (includes index register, accumulators, stack register, DMA addresses, etc.)		
15. Standard instructions	112	117	141
16. Optional instructions	9	37	31
17. Instruction types	memory reference, input/out, generic, shift		
19. Typical instruction times	all times in microseconds, P300 times with 600 nsec. memory		
Add to memory	2.44	1.96	1.56
Skip on condition	2.84-3.30	2.04-2.32	1.80-2.00
Shift A right, n bits	1.4+.36n	1.08+.24n	.92+.2n
Complement A	1.76	1.36	1.12
Hardware multiply	14	10.48	8.72
Hardware divide	18.2-19.6	13.68.-14.72	10.95
Fl. pt. load	-		4.04
Fl. pt. add	-		7.8
Fl. pt. multiply	-		20
Fl. pt. divide	-		40

Table 1-3. Electromechanical Specifications

Chassis Capacity	5 boards	10 boards	17 boards
Chassis dimensions (WXHXD)	19"x10-1/2"x19-1/2"	19"x15-3/4"x19-1/2"	19"x26-1/4"x19-1/2"
Weight (including fans and power supply)			
Operating temp. range °F	50° - 104°	50° - 104°	50° - 104°
Max. rel.humidity (no cond.)	95%	95%	95%
Mounting	table tope or rack	rack	rack
Typical heat dissapation (BTU/hr)	2,000	3,600	4,000
Voltage range (VAC)	95-125 or 190-250	95-125 or 190-250	95-125 or 190-250
Hz (single phase)	47-63	47-63	47-63
Amps (typical	5	9	10
Power Supply	40 amp. main supply, chassis-mounted.	40 amp. main supply, chassis mounted; 40 amp aux. supply rack mounted.	two 40 amp. main supplies, each chassis mounted; one 40 amp aux. supply, rack mounted; one power dist. unit.
Typical battery backup time (one battery, 32K memory)	6 hours	6 hours	6 hours

Microprogrammed Logic

A Prime central processor can be thought of as a processor within a processor. The outer processor is visible to the user, executes the machine language instructions of user programs, and interfaces with the main memory and I/O bus. The inner processor, a microprogrammed controller, is transparent to the user, and through a series of micro-instructions stored in a read-only memory, provides the logic to control and monitor the outer processor's activities. A writable control store option on the Prime 300 makes the inner processor accessible to the user by providing 256 words of random access control memory (RAM) in which to store and execute user-prepared microprograms. This feature allows the user to apply the speed (240 nanosecond typical instruction cycle) and large word size (52 bits) of the inner processor directly to the execution of application programs. By microprogramming frequently used subroutines, algorithms and special-purpose instructions, the user can improve the computer's efficiency in terms of both increased speed and reduced main memory storage requirements.

32 Addressable Registers

A general-purpose arithmetic unit and high-speed register file are the basic tools used by an outer processor to perform machine-language functions and store transient data and control information. The arithmetic unit performs arithmetic and logical operations while the 32 addressable registers in the register file handle such functions as address indexing, stack processing, program sequenceing, and control of direct-to-memory I/O data transfers. All processing is done on 16-bit words, with all bits in a word processed in parallel.

System Integrity Features

Prime 200 and 300 processors include a comprehensive array of error detection and fault location features to monitor the integrity of data as it is stored, moved and processed within the system, and to help the user manage a quick and orderly recover in the event of a system malfunction.

Memory Parity: For parity checking purposes, every word in main memory is treated as a pair of eight-bit bytes, each with a ninth parity bit. Memory parity is checked when a word is read and generated when a word is written. The response to a memory parity error is determined by the user via program control. If the machine has been set to operate in "check mode", a parity error causes an interrupt and automatic program branch to a reserved memory location. If the check mode has not been enabled, a parity error sets a flag that can be tested under program control.

Processor Parity: Byte parity is checked or generated for all data transfers on the bus between the main memory and processor, among all processor registers, on all internal processor busses, and on the bus between the processor and all I/O devices. If the machine is operating in machine check mode and is also equipped with the microverification option, a processor parity error will automatically initiate entry to a series of microverification routines. Without microverification, a processor parity error will set a flag that can be tested under program control.

Microverification: The microverification routines are a series of microprograms that can verify the integrity of the processor components other than basic clocks and control circuits. In addition to being activated by a processor parity error or a system master-clear, the microverification routines can also be entered under control of a user's program. When one of the microverification routines detects a fault condition, the identity of the routine is displayed on the control panel and the processor stalls by looping on this routine. If the fault condition clears, the processor automatically resumes normal program execution. The stalled condition of the processor can be detected automatically by a watch-dog timer (see System Option Controller User Guide) that in turn can initiate some remedial action.

Comprehensive Instruction Set

The machine-language instruction set permits manipulation of data on a bit, byte, word and multiword basis. Included in the standard instruction set for all processors is a group of memory reference instructions that minimize register housekeeping overhead, and a group of "logicize" instructions that enhance compiler efficiency by converting comparison relationships directly to truth values.

Prime 200 and 300 processors can be equipped with optionally available single- and double-precision floating point arithmetic hardware which has been optimized to run FORTRAN IV real arithmetic operations. Unique to the Prime 300 are a set of stack procedure instructions for recursive and reentrant programming, and powerful conditional jump instructions that combine condition testing and branching in a single instruction.

While each central processor provides a different mix of standard and optional instructions, instruction set compatibility among all processors is maintained by unimplemented-instruction interrupt hardware and a virtual instruction package containing software equivalents of unimplemented machine-language instructions. Thus, programs written to utilize the full instruction set of a Prime 300 (virtual memory management instructions excluded) can usually be run on a Prime 200 or 100 without modification.

All processors automatically trap illegal instructions. This feature reserves a subset of op codes for the user (they will not be implemented in Prime system software) and permits automatic branching to user-prepared subroutines when these op codes are encountered in a program.

Automatic Program Loaders

Automatic program loaders are available, either as standard or optional features (See Table 1-1), with all Prime processors. Loaders for media such as paper tape, magnetic tape, and disk are stored in a read only memory (ROM), physically located on the processor's control panel circuit board. This board can also be equipped with a user-specified custom loader that Prime will produce using up to 512 X 16 bits of programmable read-only memory (PROM).

MOS Memory

Instructions and data are stored in MOS main memory. Prime systems use MOS memory exclusively and, depending on which processor is used (see Table 1-1), memory access times of 680, 600 or 440 nanoseconds, and maximum capacities of 64K or 256K words are available. A system's main memory can be expanded in modular increments of 4K, 8K or 32K words on a single 16" X 18" circuit board. (Using 32K word memory boards, a Prime 300 processor with 256K word memory and seven controller and interface boards can be housed in a chassis only 26-1/4" high.)

Automatic Power Monitor With Battery Backup

Any configuration of processor and MOS memory can be equipped with an optional power monitor system to preserve the memory's contents if AC power is interrupted, and automatically restart program execution when power is restored. Four major functions are handled by the power monitor option: sensing line voltage not within operational limits, issuing an interrupt at the onset of power failure, battery refreshing of MOS memory, and automatic restart when power is restored.

The battery backup system includes onr or two 20 Amp-hour, sealed gel-electrolyte cells and automatic charger. The batteries are housed on a rack-mountable panel that can be installed in any position in a system rack or cabinet.

Direct-To-Memory Data Transfers

Data transfers between the main memory and high-speed devices can be performed through the use of the programmable, eight-channel direct memory access (DMA) system standard on all Prime processors. The number of direct memory channels and the maximum data rate can be expanded with DMC and DMT modes of operation (optional on the Prime 100 and 200, and standard on the 300). DMC which provides up to 2,000 direct memory channels, is similar to DMA, except that where DMA uses registers in the high-speed register file to store control information, DMC uses main memory locations. DMT is used with certain high-speed device controllers in which the controllers themselves monitor direct memory transfers with minimal processor intervention.

Processor Serial I/O Ports

In addition to the data transfers handled via the I/O bus, EIA binary signals up to 9600 baud can be handled by a four-channel, bit-serial, full-duplex interface which is an integral part of all Prime processors. By means of programmed control of this interface, serial data can be transmitted on four output lines and simultaneously received on four input lines. The interface operates on EIA standard levels, and all lines are easily accessible at the back edge-connector strip of the processor board.

Vectored Priority Interrupts

A flexible interrupt processing capability is a standard feature on all Prime processors and augments programmed control of I/O data transfers. I/O processing on an interrupt basis frees the central processor for other activities between data transfers, and automatically resolves processing priorities when multiple activities require servicing at the same time. An interrupt vectoring technique minimizes interrupt response time by assigning each interrupt source a program selectable memory location for subroutine entry. Interrupt priorities are established by the physical sequence in which device controllers are plugged into the back plane.

Prime 300 Virtual Memory Capability

The Prime 300 includes as a standard feature a virtual memory capability for secure and automatic allocation of processor and memory resources for multi-user and multi-task operations. Virtual memory expands the processing and storage capability of the Prime 300 by:

1. extending the maximum main memory addressing range to 262,144 words,

2. providing hardware memory protection for all users and tasks running simultaneously,

3. automating memory management so that programs can be written without any special consideration of the fact that they may be executed in a virtual memory environment.

With virtual memory, a Prime 300 can support multi-user time shared disk operating systems providing each of up to 15 users with a 65,536 word virtual memory space. It can also support multi-tasking real-time operating systems, foreground/background systems with real-time multi-user or multi-task processing in the protected foreground, and processing of single programs larger than 65,536 words.

The implementation of virtual memory is transparent to the system's users. As a result, each user is free to create, test, modify and execute programs without concern for how system resources will be managed to perform these functions. Furthermore, programs written for single-user Prime 100 or 200 systems can be executed by a virtual memory Prime 300 without modification, and vice versa.

Here is a simplified overview of the four key features involved in virtual memory operation.

Paging: Information is stored in main memory in fixed-length, 512-word pages. All memory accesses are automatically intercepted and translated from their normal 16-bit format to 18-bit address fields, permitting the maximum addressing range to be expanded from 65,536 words to 262,144 words. The correspondence between the original "virtual" addresses and the "real", or physical, addresses which they are translated into is maintained by a 128-entry page map stored in main memory. A separate map is stored for each user in a multi-user system.

Page map referencing overhead is held to an absolute minimum through the use of four high-speed, content-associative memory registers (CAMs). The CAMs are continually updated with copies of the four most recently used page map entries so that in over 96% of all subsequent memory accesses, one of the registers will contain the required map entries. This reduces the need to actually access the user's page map in memory and cuts address translation overhead from the memory cycle needed to access the page mape to only 80 nanoseconds to access the CAMs.

Page Turning: In a virtual memory system, the computer's main memory and disk storage appear to be one continuous memory space. Physical and logical differences between the two types of memory, as well as the actual location of the pages in a user's program are transparent to the user. These and other memory management functions are handled automatically by the Virtual Memory Disk Operating System. When a task requires execution of a program page not currently located in main memory, the disk operating system is notified via a "page fault" interrupt and automatically performs a page turning operation in which an inactive page in main memory and the required page in disk storage are swapped.

Write Protection: A user's page map, in addition to keeping track of where program pages are stored, can also keep track of which pages are permitted to be altered (unprotected) and which are not (protected). Thus, the page map is consulted whenever a write operation is attempted; writing into unprotected pages is permitted, writing into protected pages is automatically inhibited.

Restricted Execution: User-level programs operate in a restricted execution mode in which input/output, interrupt and control instructions are not executed directly. These instructions, when encountered in a program, cause control to be turned over to the operating system which then either performs the necessary operations or returns error messages if execution of the instruction (such as a Halt) would alter the machine state. By allowing the operating system to provide standardized procedures for handling these functions, programs originally written for memory-resident or single-user disk operating systems can run in a multi-user virtual memory system without modification.

Virtual memory operation is initiated and terminated via program control, using the following group of four, double-word control, instructions: Enter Paging Mode and Jump (EPMJ), Leave Paging Mode and Jump (LPMJ), Enter Restricted Execution Mode and Jump (ERMJ), Enter Virtual Memory and Jump (EVMJ). The last instruction combines the functions of EPMJ and ERMJ.

ELECTROMECHANICAL PACKAGE

Each central processor, memory increments of up to 32K words, peripheral device controllers, I/O interfaces, and integral power supplies are fabricated on individual 16" X 18" circuit boards. This function-on-a-board packaging simplifies system configuration and, when combined with error detection and fault location features such as byte parity, microverification and controller loop-back, permits accurate isolation of a system malfunction and quick board-replacement maintenance.

Configuration flexibility is further enhanced by a choice of three standard chassis and power supply combinations: a 5-board chassis with integral 40 amp power supply, a 10-board chassis with integral 40 amp supply and externally mounted 40 amp auxiliary supply, and a 17-board chassis with two integrally mounted 40 amp supplied and one externally mounted 40 amp auxiliary supply with power distribution unit. By referring to the Prime Computer Configurator, a user can quickly determine the best chassis/power supply combination for his application. Also, should system expansion require additional chassis capacity, the Prime Computer User Plan offers a two-year trade-in policy with guaranteed allowances toward the purchase of larger chassis/power supply combinations.

Except for board capacity, all chassis are essentially the same, with all circuit boards and integral power supplies plugged-in through the rear to an etched interconnect-plane at the front of the chassis. All external cabling is attached to rear-edge connector strips on the processor and controller circuit boards. Side-mounted fans provide air circulation within the chassis. All systems use a common control panel assembly which plugs into the front of any chassis.

EXTENDED CONTROL STORE FEATURES

The performance of systems with Prime 200 or 300 processors can be enhanced through the addition of an optional Extended Control Store (XCS) board. When used in conjunction with the Prime 200, the XCS board provides the microprogrammed logic for single- and double-precision, floating point arithmetic. With the Prime 300, the XCS board can be used to provide either floating-point hardware, writable control store, or a combination of both features.

Floating Point Arithmetic: The floating point arithmetic feature provides direct hardware execution of the 27 floating-point arithmetic instructions in the Prime instruction repertoire. (On systems without floating-point hardware, these instructions are automatically trapped and emulated via software subroutines in the Virtual Instruction Package.) The floating-point hardware produces direct in-line coding of floating-point instructions in FORTRAN and assembly language programs, eliminating the idle time associated with branching to and from floating-point subroutines. Floating point arithmetic can be performed in either single- or double-precision formats. In the single precision format, two words are used to store the mantissa and characteristic, and accuracy is maintained to seven significant digits. The double-precision format uses four words and maintains accuracy to 14 significant digits.

Writable Control Store: This feature provides user-access to the control store of the Prime 300 and adds 256 words of random-access memory for storage of user-prepared microprograms. Microprograms can be written symbolically using Prime's Micro Assembler language, and loaded from main memory into the control store using a Prime-supplied loader. The assembler and loader operate under control of the Disk Operating System. For details of microprogramming techniques, refer to the Microprogrammers Handbook.

A special group of four jump instructions are provided with this feature to transfer control to the writable control store. These instructions use a double-word format, with the first word containing a pointer to a main memory location containing the address of a micro-instruction in the control store. When operating in the restricted execution mode, the machine traps these instructions so that the operating system can determine how to handle user-program requests for access to the writable control store.

DATA STRUCTURE

Prime instruction and data word formats are discussed in Section 3. The bits of a word are numbered 1 to 16, left-to-right, and can be interpreted as an instruction, a logical word, an address, a pair of 8-bit bytes, or a 16-bit signed or unsigned number.

HIGH-SPEED REGISTER FILE

All processor registers are physically located in a high-speed register file and logically addressed as if they were MOS memory locations. Memory addresses 0-37 are reserved for this purpose and correspond to the following registers:

Memory Address	Register Designation	Function
0	X	Index Register
1	A	Arithmetic Register
2	B	Extension Arithmetic Register
3	S	Stack Register
4	FLTH	Floating Point
5	FLTL	Accumulators
6	VSC	Visible Shift Count
7	P	Program Counter
10	PMAR	Page Map Address Register
11	Reserved for microprogram	
12	PFAR	Page Fault Address Register
13-17	Reserved for microprogram	
20-37	DMA 1-8	Word pairs for DMA channels (address and word counts)

MEMORY ADDRESSING

The main memory is addressed as a set of continuous word locations. 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 groups called sectors.

Sectored and Relative Addressing Modes

In sectored mode addressing, all sectors are 512 words long and an instruction may address either the locations in sector 0 (locations $0\text{-}777_8$) 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-256 to P+255). The sixteen values from P-241 to P-256 are interpreted as special address formation such as stack register operation, base-plus-displacement, and direct addressing of the entire memory space.

Indexed and Indirect Addressing

Each memory reference instruction calculates an effective address. This calculation may include one or more levels of indirect addressing, as well as pre- and post-indexing. A stack register is available for use in push-pop stack operations and fully recursive reentry procedures.

Summary of Memory Addressing Modes

Addressing Mode	S (bit 7)	Displacement (bits 8-16)	Addressing Class	No. of Words
16 Sectored 32 Sectored	0 or 1	$0\text{-}777_8$	Sectored	1
32 Relative 64 Relative	0	$0\text{-}777_8$	Sectored	1
32 Relative 64 Relative	1	-240 to +255	Relative	1
32 Relative 64 Relative	1	-256 to -241 (op code extension rather than address value)	Extended Addressing	
			Long Reach	2
			Stack Relative	2
			Stack Post-increment	1
			Stack Pre-decrement	1

INSTRUCTION SUMMARY

Listed in Appendix C are execution times for all machine-language instructions. Two sets of timing data are shown for the Prime 300: one for a processor with 600 nanosecond access memory, the other for 440 nanosecond memory.

SCOPE OF USER GUIDE

This document provides information to evaluate the capabilities and program the Prime 100, 200 and 300 computers . The following Prime documents should be available for reference:

Prime CPU Operators Guide (Control panel and paper tape device operation)	MAN1672
Prime Macro Assembler Lanaugage Reference Manual (Assembly language syntax and pseudo operation)	MAN1673
General Purpose Interface Manual	MAN1676
Prime Installation and Maintenance (CPU physical characteristics, maintenance practices)	MAN1677
Prime Microcoders Handbook (Microprogramming techniques)	MAN1857
Magnetic Tape Controller User Guide (Operation, maintenance, and programming for systems with magnetic tape)	MAN1940
System Option Controller User Guide (Controller to interface custom devices on communications line)	MAN1944

SECTION 2

CENTRAL PROCESSOR ORGANIZATION

This section contains a CPU block diagram, and a brief description of processor organization, description of standard CPU functions, including microprocessor control, instruction execution, memory cycling, interrupt and trap handling, input/output, and data integrity features. The block diagram (Figure 2-1) is an overview of processor functions. A more detailed diagram in given in the Microprogrammer's Handbook.

PROCESSOR ORGANIZATION

From the user's point of view, the central processor is the control unit for the entire system; it performs all arithmetic, logical, and data handling operations, manages address calculations, and sequences the program. It is connected to the memory by a memory bus and to the peripheral equipment by an I/O data, address and control busses. The processor (Figure 2-1) consists of a set of high speed hardware registers addressed by a register file, an Arithmetic Logic Unit, and other registers such as the Y and M memory buffers that are connected to the memory and I/O busses. Microprocessing logic manipulates data contained in these system elements to execute each instruction.

Micropgoram Control

Processor arithmetic operations are performed by manipulating data contained in the register file in conjunction with the arithmetic logic unit. Processor arithmetic operations, data transfers to and from main memory and peripheral I/O operation are all controlled by a microprogram stored in read-only memory. The microprogram is a separate program stored in increments of 256 52-bit micro-instructions. The microprocessor executes one or two micro-instructions during each nominal machine cycle to execute user-level instructions, calculate addresses, accomplish interrupts, oversee I/O transfers, and in general perform internal system control functions.

High Speed Register File

The first 32 memory locations (0-'37) are high speed hardware that permits multi-step instruction op-codes, (e.g., multiply, double-precision) to proceed at several times the memory cycle time under microprogram control. The X, A and B registers can be addressed symbolically or as memory locations.

A detailed discussion of microprogramming and the associated registers is given in the Microprogrammers handbook.

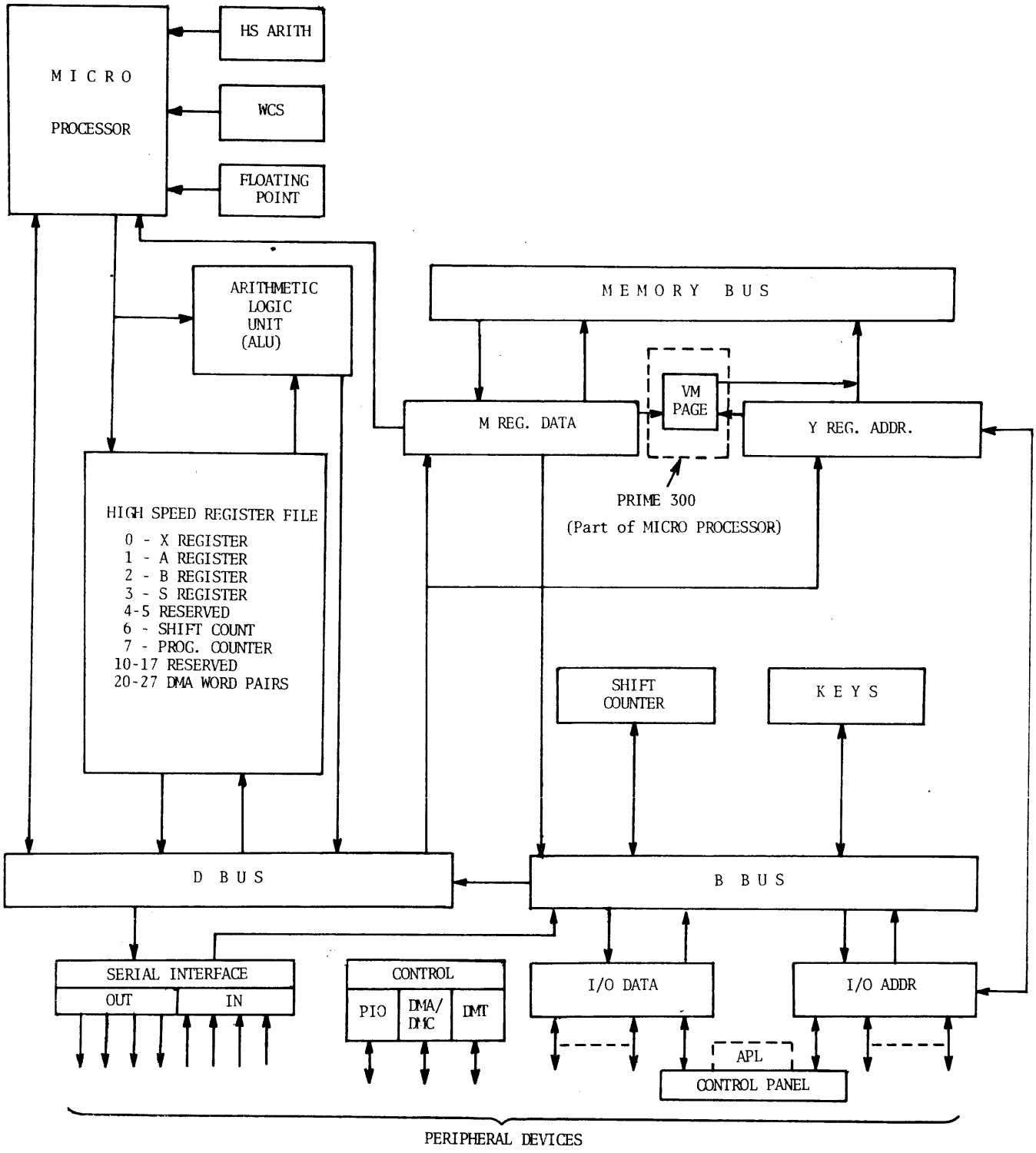


Figure 2-1. CPU Block Diagram

CENTRAL PROCESSOR DESCRIPTION

As stated in Section 1, Prime's microprogrammed computer can be thought of as having a processor within a processor. From the user standpoint, the outer processor is a stored program digital computer and consists of a control unit, main memory, arithmetic, and I/O logic. In a micro-programmed computer, however, the function of the control unit is implemented with an inner control memory containing an orderly arrangement of instruction sub-elements. These sub-elements, called micro-instructions, are arranged into a series of steps (a micro-program) to execute a user level or outer processor level instruction.

The inner processor or micro-processor also contains a control unit, memory, and I/O facilities. It too contains a program address register, fetches instructions, and executes them. It is even capable of being interrupted from normal instruction level sequences in order to handle DMA, Power Failure, Machine Checks, etc.

In order to achieve speed in executing user level instructions and minimize random discrete logic, the micro-instruction word is 52 bits wide and is expandable to 64 bits wide. The micro-instruction is divided into 12 fields, with each field controlling a portion of the processors operation.

The microprogram resides in a read-only control memory (ROM), which makes it impervious to power outages and programming errors.

Every function that the outer processor would normally perform is controlled by a series of micro-instruction steps. This includes fetching user-level instructions from memory, incrementing the program address register, and executing the instruction. Unlike the outer processor, the microprogram never stops. Even when the outer processor is executing a HALT instruction, the microprogram is monitoring the control panel and is ready to respond to control panel input. On Prime systems, the control panel is an I/O device and the switch settings are interpreted by the CPU as data words, each bit having a particular function. The microprogram decodes the sense switch data and then controls user program execution, and displays data and program addresses on the control panel displays.

STANDARD CPU FUNCTIONS

Sequenctial Instruction Execution

The address for a memory access is held in register &, and data read from memory or about to be stored in memory is held in register M. The processor performs a program by executing instructions retrieved from consecutive mememory locations as counted by the program counter (P), Register file 7 on the block diagram. 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.

Addressable Registers

A general-purpose arithmetic unit and high-speed register file are used by the outer processor to perform machine-language functions and store transient data and control information. The arithmetic unit performs arithmetic and logical operations while the 32 addressable registers in the register file handle such functions as address indexing, stack processing, program sequencing, and control of direct-to-memory I/O data transfers. All processing is done on 16-bit words, with all bits in a word processed in parallel.

Arithmetic Register

All computations are performed using the ALU and the Arithmetic or A register. Data can be moved in either direction between A and any memory location via the D Bus and the M Register. The contents of a memory location can be combined arithmetically or logically with the contents of A. The A register also serves as the data connection with the programmed I/O bus, via the D Bus and B Bus. A secondary arithmetic register, the B Register, serves as a right extension of A for double length operations. The processor also has a single-bit register, the C Register (or Carry Bit), that is set on overflow in arithmetic operations and is loaded with the last bit dropped out of A or B in shift operations.

Referencing Memory

Each memory reference instruction calculates an effective address that is stored in the Y register 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. The index register (X) as well as the S (stack) register 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 that is performed independently of and addition to the indexing in the effective address calculation involving X.

MOS Memory

Instructions and data are stored in MOS main memory. Prime systems use MOS memory exclusively and depending on which processor is used, memory access times of 680, 600 or 440 nanoseconds, and maximum capacities of 64K or 256K words are available. A system's main memory can be expanded in modular increments of 8K or 32K words on a single 16" X 18" circuit board.

Automatic Power Monitor with Battery Backup

Any configuration of processor and MOS memory can be equipped with an optional power monitor system to preserve the memory's contents if AC power is interrupted, and automatically restart program execution when power is restored. Four major functions are handled by the power monitor option: sensing line voltage not within operational limits, issuing an interrupt at the onset of power failure, battery refreshing of MOS memory, and automatic restart when power is restored.

The battery backup system includes one or 20 Amp-hour, sealed gel-electrolyte cells and automatic charger. The batteries are housed on a rack-mountable panel which can be installed in any position in a system rack or cabinet.

Direct-To-Memory Data Transfers

Data transfers between the main memory and high-speed devices can be performed through the use of the programmable, eight-channel direct memory access (DMA) system, standard on all Prime processors. The number of direct memory channels and the maximum data rate can be expanded with DMC and DMT modes of operation (optional on the Prime 100 and 200, and standard on the 300). DMC, which provides up to 2,000 direct memory channels, is similar to DMA, except that where DMA uses registers in the high-speed register file to store control information, DMC uses main memory location. DMT is used with certain high-speed device controllers in which the controllers themselves monitor direct memory transfers with minimal processor intervention.

Processor Serial I/O Ports

In addition to the data transfers handled via the I/O bus, EIA binary signals up to 9600 baud can be handled by a four-channel, bit-serial, full-duplex interface which is an integral part of all Prime processors. By means of programmed control of this interface, serial data can be transmitted on four output lines and simultaneously received on four input lines. The interface operates on EIA standard levels, and all lines are easily accessible at the back edge-connector strip of the processor board.

Vectored Priority Interrupts

A flexible interrupt processing capability is a standard feature on all Prime processors and augments programmed control of I/O data transfers. I/O processing on an interrupt basis frees the central processor for other activities between data transfers, and automatically resolves processing priorities when multiple activities require servicing at the same time. An interrupt vectoring technique minimizes interrupt response time by assigning each interrupt source a program selectable memory location for subroutine entry. Interrupt priorities are established by the physical sequence in which device controllers are plugged into the back plane.

Virtual Memory

The Prime 300 includes, as a standard feature, a virtual memory capability. For details, refer to Section 7.

INSTRUCTION EXECUTION

Refer to the block diagram, Figure 2-1 to supplement reading of this discussion of instruction execution.

High-Speed Register File

All processor registers are physically located in a high-speed register file and logically addressed as if they were MOS memory locations. Memory addresses 0-37 are reserved for this purpose and correspond to the following registers:

Memory Address	Register Designation	Function
0	X	Index Register
1	A	Arithmetic Register
2	B	Extension Arithmetic Register
3	S	Stack Register
4	FLTH	Floating Point
5	FLTL	Accumulators
6	VSC	Visible Shift Count
7	P	Program Counter
10	PMAR	Page Map Address Register
11	Reserved for microprogram	
12	PFAR	Page Fault Address Register
13-17	Reserved for microprogram	
20-37	DMA-18	Word Pairs for DMA channels (address and word counts)

Transfer of Information

The simplest CPU operation is the transfer of information from one register to another register or a series of registers; for example, to transfer the contents of the A-Register in the register file to Register M and thence to the memory bus. To do this, the A Register must be selected; the register file must be allowed on the Bus D; the resultant data on Bus D must be put into the M Register and its effective address must be calculated and stored in the Y Register, then the M Register and Y Register address must be transferred to the Memory Bus. Prior to transfer of data from the Memory Bus to MOS memory, if the machine is a Prime 300; the paging hardware must be utilized to (1) check if the page is in memory and (2) if not, bring the page into memory. Finally, the data in the memory bus must be transferred to memory at the specific memory address. The program Counter (Register file Register 7) must have been incremented when the instruction was fetched. This process is roughly a Store A (STA) instruction.

Conversely, information may be taken from memory and moved back down to a register in the register file. This process is roughly equivalent to a Load A (LDA) instruction. To do this, information must be transferred from the memory to Memory Bus to the M Register which must be selected as the source of Bus D via a transfer through Bus B. Then the register file must be used as the source of the information on Bus D. Finally, the P counter (Register File 7) must have been incremented when the instruction was fetched. These operations are accomplished by selecting and setting the proper microcode fields then executing the microcode. For details of the fields that are set and how to construct microcode information, refer to the Microcoders Handbook (MAN 1940).

Transfers Using the ALU

To add two values, the first of which is in the M Register and the second of which is in the A Register and then load the result into the A Register; it is necessary to first get the correct data to the inputs of the Arithmetic Logic Unit (ALU). This is done by selecting the M Register as the source of the B Bus and the A Register as the register file register. Next, the ALU must be conditioned to add. This is done by selecting the microcode fields for addition (Refer to the Microcoder's Handbook). After the add operation, the results have to be loaded back into the A register by selecting the ALU as the source of Bus D and the A Register as the destination of the information on Bus D.

Shifting

Shifting is controlled by microcode. This includes both the type of shift and the end conditions. It is accomplished by using the information in the S Register (Register File Register Number 6) as the source for the information on the D Bus. Each output of the S register is shifted (right or left) one place before being placed into the D Bus; the Shift Counter is used to keep count of the number of shifts. This counter is created outside of the register file and can be loaded from Bus B and read in as the low order half of Bus B. The shift counter incrementation takes place at the end of the shift cycle.

MEMORY CYCLING

MOS memory provides an optimum combination of high speed, simple plug-in expansion and high density packaging. Memory cycle times are either 600 or 750 nanoseconds on the Prime 300, 750 nanoseconds on the 200 and 1 microsecond on the 100. A single etched circuit board provides 8K words available in increments up to 32K per board with integral byte parity. Memory capacity is expandable to 64K in 8K increments on all Prime computers, and to 256K in 32K increments of 750 ns memory on 300-series machines.

The main memory is addressed as a set of contiguous word locations whose addresses range from 0 to '177777 or 65,536. (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 that provide additional methods of address formation such as stack register operation, base-plus-displacement and direct addressing of any location from 0 to '177777.

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.

Reserved Memory Locations

Locations '40 through '777 of sector zero are reserved for the specific purposes listed in Table 3-1. ??

Locations '40-'57 are reserved for eight Direct Memory Channel (DMC) data words and eight channel control words. Locations '60 through '74 are dedicated for specific interrupts, both internal (i.e., memory parity errors and illegal instructions) or external (peripheral device interrupts). Locations '100-'177 are set aside for vectored interrupts from peripheral devices (i.e., the locations used for a particular interrupt is typically '100 plus the code of the device causing the interrupt).

INTERRUPT AND TRAP HANDLING

Traps and Interrupts

Traps result in branching in the microcode. Interrups result in branching in the executing program. Some traps also cause interrupts.

There are external and internal interrupts. Internal interrupts are those caused by traps, such as unimplemented instruction interrupts, etc. External interrupts are caused by real-time interrupt requests from device controllers plugged into the backplane. External interrupts can be enabled or disabled by the INH and ENB instructions.

External interrupts have two modes, vectored and standard, selected by the EVIM and ESIM instructions. In standard mode, an indirect JST through location '63 is executed. In vectored mode, the indirect JST is through a vector address provided by the interrupting controller. In both modes, interrupt priority is determined by the backplane.

Interrupts

There are 13 different interrupt vectors now allowed for in the Prime processors. They fall into several broad classes: hardware monitoring, external, and software aids.

All of these interrupts have some properties in common. First, all of the interrupts check their vector location to see if it is zero before going indirect through it. If it is zero, a HALT or HLT is executed. Second, the vector is interpreted as a 16 bit absolute address independent of address mode in force. Third, the program counter is deposited at the address pointed to by the vector and execution begins at the next address. Fourth, the non-visible keys are changed by clearing out the 'system clear' and 'permit external interrupts' flops. Fifth, all vectors do an absolute vector.

Hardware Monitoring

These interrupts as a class check on the operability of the system and give the user warning of past or approaching failures:

1. Missing Memory Module

   The memory does not exist at a location accessed. This interrupt may be used to determine memory size. It may result from the CEA instruction as well as any memory reference instruction.

   The interrupt cannot be inhibited and deposits the P counter pointing to the next instruction to be executed. The Machine Check Flag is cleared by this interrupt.

2. Memory Parity

   An error has been detected in the memory data most recently read.

3. Machine Check

   An internal data transfer or I/O bus transfer generated a parity error.

4. Parity Fail

   This uninhibitable interrupt is taken when a failure of system power is detected. The interrupt is through location '60 and is given 1 millisecond before an internal system clear signal is given. If location '60 has an address other than zero in it, an interrupt to that location will be executed. If the contents of location '60 is zero, a halt occurs.

   Systems with battery backup can mainimize the effect of power loss by saving applicable data registers, terminating peripheral transfers and setting up for an auto restart at location '1000 when power is restored.

External Interrupts

These interrupts serve as the normal asynchronous sources of external stimuli to the processor. Included in this class are all of the normal peripheral interrupts.

1. Real Time Clock (Increment)

   This interrupt does not interrupt program execution. However, it does increment location '61 of memory every 16.6 milliseconds (20 milliseconds for 50 Hz systems). On incrementing to zero, an external interrupt through location '63 is requested.

   Incrementing the clock is not affected by the ENB and INH instructions, but can be started and stopped using programmed I/O.

2. Real Time Clock (Overflow)

   This is a standard external interrupt. (See 3.)

3. Interrupt (Compatible Mode)

   This interrupt is for all external devices. It can be enabled or inhibited using the ENB and INH instructions, respectively. The actual device interrupting must be determined by a polling method. External interrupts are automatically inhibited by this interrupt. External interrupts come here if the processor is in compatible mode.

4. Interrupt (Vectored Mode)

   Identical in function to compatible mode, this method is used if the processor has been put into the vectored interrupt mode. ENB and INH word as before.

   This time, however, each interrupt uses a vector specified by the controller (normally '100 + Device Address) and the vector can be anywhere in the first 64K memory.

Software Aids

These vectors serve as a link to tie user developed software to Prime developed software along a clearly defined path. In addition, standard software can use these traps to run efficiently on large Prime machines while still running successfully on smaller Prime machines.

SVC (SerVice Call): This interrupt is a convenient way unambiguously demanding the attention of the executive software. Argument transfer will typically be done using the computer words in memory that follow the SVC.

Prime executive software defines the SVC calls. The advantage of using SVC is: an SVC works the same in normal, restricted, or virtual execution mode. Thus standard software is able to run in different execution environments.

Restricted Execution Violation: This interrupt is enabled by executing an ERM and disabled by any interrupt (including SVC).

If enabled, this interrupt occurs whenever a restricted user executes any I/O (including ISI and OSI) instructions, or machine mode change of any non-visible key, or over n levels of indirection (n = a convenient number), or execution of a HALT. This feature is found only in systems with virtual memory.

UII (Unimplemented Instruction): To permit upward compatible software, Prime has reserved octal codes that when executed cause an unimplemented instruction interrupt. On the Prime 100 and 200, Multiply and Divide are examples of instructions that cause this unimhibitable interrupt.

As a result, a package that decodes and software-implements these instructions, can be added. To help this unimplemented instruction (UII) package, the program counter contents is saved so that a deposited program counter always points to the instruction that caused the interrupt.

ILL (Illegal Instruction): To permit customer use of special op codes which act as UII's, Prime has defined many codes as illegal. Execution of these causes an interrupt similar to the UII (Unimplemented Instruction package). The difference is that an instruction that is unimplemented can easily become implemented in the future by microcode changes. Illegal instructions, however, will remain illegal.

INTERNAL INTERRUPTS

Besides the use of interrupts to handle the peripheral equipment, a number of internal processor situations 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 simply by virtue of the circumstances they represent among internal interrupts, priority is a function of logical necessity.

In response to an 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 the P register in the location and resumes normal program execution at the location following that in which the P register 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 must be left unimplemented (zero) unless the processor has the memory save option.

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

'62 Restricted execution in VM.

'63 External interrupts use this location.

'64 Page Fault.

'65 Supervisor Call - an interrupt to this Service.

INPUT/OUTPUT

As shown on the block diagram, a Prime computer system can be connected to a variety of peripheral and/or terminal devices. A more complete discussion of I/O is given in Section 4.

Generally, I/O Data is transferred to and from the B Bus from the serial interface or AMLC or SMLC devices. Device types other than the serial interface (bit banger) interact with the B Bus through an I/O Data Buffer and and I/O Buffer, similar to the way in which the CPU interacts with the memory bus through the memory data and address buffers. Serial input is routed to the B Bus; however, serial output is directed from the D Bus directly to the serial interface buffer. Note also, that the control panel has a buffer and is treated as an I/O device; thus setting sense switches can input information directly into the CPU.

DATA INTEGRITY FEATURES

The following paragraphs summarize data integrity features available on Prime systems and the purpose of traps, and interrupts within the central processor.

The Prime 200 and 300 CPU's include several levels of automatic, program-independent data integrity check features:

Memory Parity	Checks parity of every 8-bit byte read from high-speed memory. If machine check mode is in effect, an interrupt through location '67 is taken.
Machine Check Mode	Enabled or disabled by EMCM and LMCM instructions. Enables memory parity interrupts and microverification, if present.
Machine Checks	Parity checks of byte transfers between internal registers and busses. Errors cause entry into microverification routines, if the option is present and machine check is enabled.
Microverification	Optional microcode test routines that test the logic of the entire CPU. (See Section 5.)
Interrupt	Associated with the user level program. An interrupt performs a control transfer to a location specified by the location associated with the type of interrupt found. This amounts to an indirect JST.
Trap	Associated with microcode. A trap transfers micro-control to a specific trap catching microroutine. Some traps also generate interrupts; others do not.
Memory Parity Error	A parity error in a word read from memory.
Machine Check Error	A parity error in any other situation (in a register, over the I/O bus, etc.)

Memory Parity and machine checks are standard on the Prime 200 and 300 and microverification is optional on both processors. These features are not implemented in Prime 200's.

Machine Check Functions

Occurrence of either memory parity error or machine check error in any Prime processor always sets the Machine Check Flag, depends on the type of processor (does it have microverification or not?), its operation

mode (Normal Operating Mode or Machine Check Mode), type of error (memory parity or machine check), and type of failure (solid or transient).

Normal Operating Mode (Enabled by MASTER CLEAR or LMCM Instruction)

Memory Parity Error: Memory Parity Error in any Prime machine operating in Normal Operating Mode always sets the Machine Check Flag. There is no interrupt to the operating program. To check for parity error, the operating program may use the SMCS (Skip on Machine Check Set) or SMCR (Skip on Machine Check Reset) instruction. It is then up to the system programmer to handle this problem. Master Clear or RMC (Reset Machine Check) can be used to reset the flag.

Machine Check Error: The same procedure as for Memory Parity Error applies.

Machine Check Mode (Enabled by EMCM)

Memory Parity Error: In any Prime processor operating in Machine Check Mode, a memory parity error sets the Machine Check Flag. This causes a microcode trap that executes a microroutine to reset the machine check flag and causes a program interrupt through location '67. Response to this interrupt is decided by the system programmer's interrupt service routine.

Machine Check Error: CPU Without Microverification. A machine check error occurring in a Prime Type 211 or Type 215 central processor running in Machine Check MOde, causes a microprogrammed interrupt that resets the Machine Check Flag and causes the processor to halt (indicated by the control panel STOP light). If the operator turns the function selector to STOP/STEP, all address lights will be lit.

CPU with Microverification: A machine check error occurring in a Prime computer with microverification running in Machine Check Mode, initiates execution of the microprogram verification routine to check (verify) proper operation of the processor. The verification routine always resets the Machine Check Flag.

Parity Errors

Several alternative ways of detecting and recovering from parity errors are provded by Prime hardware.

Prime 200 and 300 series computers detect Memory Parity errors by checking byte parity on each memory read operation. Byte parity errors that occur during data transfers between CPU registers, the backplane and the arithmetic unit, are all classified as Machine Check parity errors. Both memory and machine check parity errors set the Machine Check Flag.

In the Normal operating mode, Prime 200 and 300 computers resemble the Prime 100, which has no parity check hardware. The user may employ the SMCS (Skip on Machine Check Set) and SMCR (Skip on Machine Check Reset) instructions to sense parity errors by testing the Machine Check Flag and may provide subroutines to handle parity errors.

Special instructions (EMCM, LMCM) are provided that cause the computer to enter the Machine Check Mode. In the Machine Check Mode, when a Memory Parity error sets the Machine Check Flag; a microcode program resets the flag and causes an interrupt through location '67.

Depending on a program's sensitivity to memory parity errors, a user may choose to provide reentry points and a service routine to repeat the calculation or a user may choose some other solution.

In Prime 300 computers, memory is organized into 512-word sectors or pages; and the Virtual Memory paging technique enables the user to edit out and work around a defective page if interrupts through location '67 occur consistently from a particular area in memory. Also, operating system software checks for bad memory and takes appropriate action to work around that memory (refer to the Disk and Virtual Memory Operating Systems User Guide).

In processors without microverification, machine check parity errors cause the processor to halt, as indicated by the control panel stop light. In this case, turning the function selector to the STOP/STEP position lights all the ADDRESS lights. This action confirms that a CPU parity error has occurred.

In 200 and 300 series machines with the microverify option, a Machine Check error activates the microcode verification program. This program runs a series of tests on individual registers in the processor, arithmetic unit and I/O bus. If the entire microverify routine is cycled without a failure being diagnosed in a particular circuit, the parity error is assumed to have been caused by a transient condition. The microverify routine then clears the Register File and Machine Status Keys and causes interrupt through location '70. The program can then resume execution after the machine state is restored if the user program has been set up to handle this situation.

If the microverification routine encounters a nontransient circuit failure, it continues to cycle as long as the failure persists; and the number of the test is displayed in the ADDRESS lights when the function selector is set at the RUN or LOAD position. The processor leaves the Machine Check mode and reenters the Normal operating mode when it encounters the LMCM (Leave Machine Check Mode) instruction. Thus, there are two operating modes: Normal and Machine Check. In Normal mode, parity errors do not influence program flow unless explicit instructions are inserted into the user's program. In the Machine Check mode, a parity error during memory read causes an interrupt through location '67 that may be acted upon at the user's discretion; otherwise, program execution continues. In processors without microverification, a Machine Check parity error halts the machine because processor parity

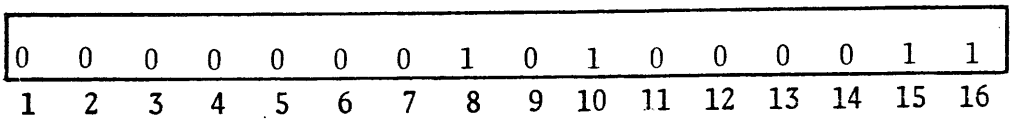
errors are assumed to be more serious than memory parity errors. In Machines with the microverify option, if the CPU passes the tests performed by the microverify routine; the assumption is that the trouble was a transient one and processing resumes.

When power is turned-on (and when the MASTER CLEAR button is pressed), processors with microverification perform a CPU circuit integrity check. Then, the computer operates in Normal mode. The Machine Check Flag signifying a memory/CPU parity error is ignored in this case.

The microverification routine can be executed at any time by means of the VIRY instruction to assure the integrity of processor circuits. Similarly, the user can assure himself that the CPU is functioning properly by turning on the power and pressing MASTER CLEAR to initiate the microverify sequence.

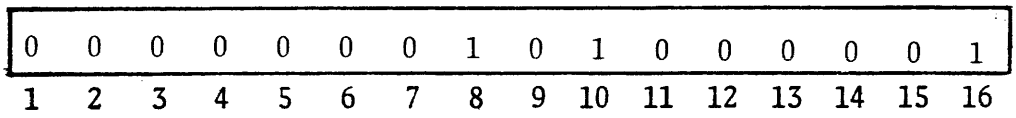
The following paragraphs describe instructions that may be used to enter, leave and reset Machine Check mode and to execute the verification routine.

EMCM Enter Machine Check Mode '000503



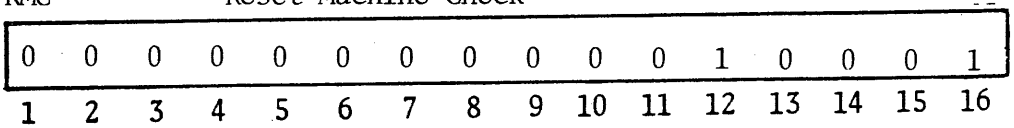
Enter Machine check mode so that the microprogram responds to a parity error.

LMCM Leave Machine Check Mode '000501



Leave machine check mode so that a parity error will simply set the Machine Check flag but will not cause the machine to halt.

RMC Reset Machine Check '000021



Reset the Machine Check flag.

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

SMCS Skip on Machine Check Set '101200

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

If the Machine Check flag 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

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

If the Machine Check flag is reset (indicating no 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 an absolute skip.)

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

MICROVERIFICATION

The microverify feature provides the Prime processor (200 (optional), 300, STANDARD) with a self-test capability. This self-test capability consists of a set of microcode routines that verify the operations that can be successfully performed by the processor. These tests are carefully constructed to verify successively larger portions of the CPU hardware, always building on those portions that are already verified. Table 2-2 lists the verification routines and describes the microcode logic that the test exercises.

One of the fundamental tasks of the microverify feature is to verify that the machine checking hardware (Machine Check) is detecting both good and bad parity correctly; this is done with Tests 4, 5, and 6. Machine Check checks byte parity on internal and external data paths. Parity is generated only when necessary (i.e., on shift and ALU operations), and parity is normally transmitted from one register to another unchanged. Each processor register includes parity checking logic. This parity checking normally detects all single bit parity failures and detects looping multiple faults after a few data patterns have been used. A parity error detected by the machine check hardware traps to the microprocessor and causes the verify microcode to be executed. The verify microcode is a series of microcode tests (refer to Table 2-2). The tests begin by testing as little as possible and continue by testing larger portions of the machine and using more involved data patterns. Finally, the microcodes test large external blocks of logic (e.g., memory and I/O).

The verify microcode exercises the processor's control unit as well as the ALU, the registers, and the various data paths (See Figure 2-1). Because address and data busses of both memory and I/O (Bus D, Memory Bus, Bus B, etc.) are tri-state and because all but the memory address bus are bi-directional, a failed component (board) anywhere in the system can stop all data transfer to and from a bus. Two microverify tests (11 and 12) explicitly check for this condition by verifying that both ones and zeroes can be placed upon the busses.

Microverify status is displayed at the Control Panel Indicator Lights.

The VIRY Instruction

The self-test (microverify) routines described in this section may be initiated by a VIRY instruction. The VIRY instruction is described in Section 6. The VIRY instruction skips on no error and returns with the failed test number on error. The microverify routines are also initiated by a MASTER CLEAR and by a Machine Check error.

Table 2-3 shows how entry and exit of the verify test operates.

Table 2-2. Verification Routines

Test No.	Test Exercises:
0	ALU - 0, Condition Code Jump on not equal
0	RM, RY, EMIT, RA (Register A) all can be set = 0 ALU (subtract) Internal busses - transmit 0
1	BB can be loaded from RY
2	Modals and Traps are tested to verify reset
3	RX can be incremented BD can transmit bits on lower byte RSC words (it counts and loads correctly) RCM emit $\neq$ 0 can be done
4	RY parity detection Control unit 16 way branch BB, BD data transmission Jump Logic
5	RM parity detection Control unit 16 way branch BB, BD data transmission Jump Logic
6	Register files parity detection and all tested in 5
7	ALU = -1 BD shift left, BD parity generate RF parity check RSC increment Jump
7	Carry bit, Load, set and reset ALU = subtract Jump Logic
10	Register file with various patterns of bits RSC increment BB, BD various sources and patterns Jump Logic

Table 2-2. (Cont)

Test No.	Test Exercises:
11	I/O busses, BPA, BPD are able to transmit a one and zero in each bit. Right Shift RM, RY, RF data and parity
12	Memory Busses, BMA, BMD. Memory Location 5 BMD is tested for a 1 and 0 in each bit Memory timing must work
13	Discovers a parity failure in tests 7-12

Table 2-3. Microverify Entry & Exit

ENTRY to microverify routines will be upon:	EXIT from microverify routines will be upon:
System Clear (MASTER CLEAR)	No Error
CPU parity error (Machine Check)	Successful pass or first error detected.
VIRY instruction	Successful pass or first error detected.

SECTION 3

INSTRUCTION FORMATS & ADDRESSING TECHNIQUES

The Prime instruction set is subdivided into classes by function (control, arithmetic, I/O, interrupt, memory reference, I/O, shifts, and generics) which include logical operations, byte manipulation and internal processor control. See Appendix C for a summary of all instructions.

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 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.

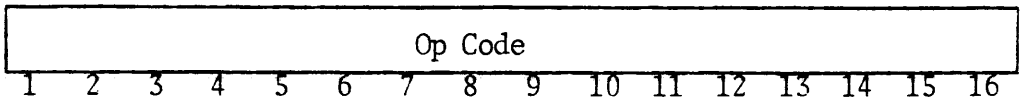
Two common conventions are to regard a number as an integer (binary point at the right) or as a proper fraction (binary point at the left); in these two cases the range of signed numbers represented by a single word is -2^{15} to $2^{15}-1$, or -1 to $1-2^{-15}$. For instructions that operate on double 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, memory-reference, and input/output.

Generic Instructions

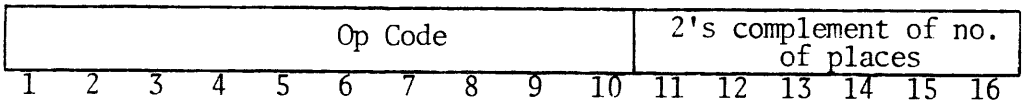
In the generic group, the entire instruction word is treated as an op code, as follows:



Certain instructions for virtual memory and extended control store operations include a target address in the word following the op-code word.

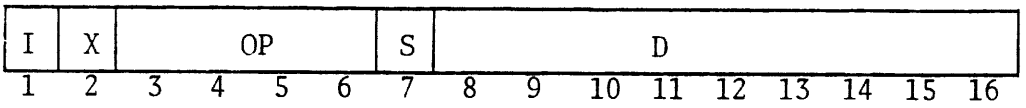
Shift Instructions

Instructions in the shift group consist of an op code plus a field that indicates the number of places to be shifted as follows:



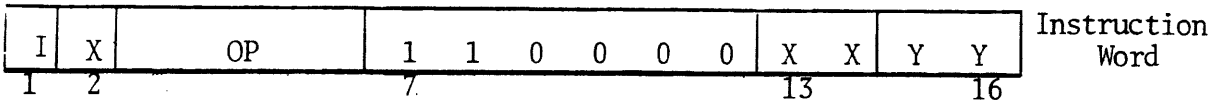
Memory Reference Instructions

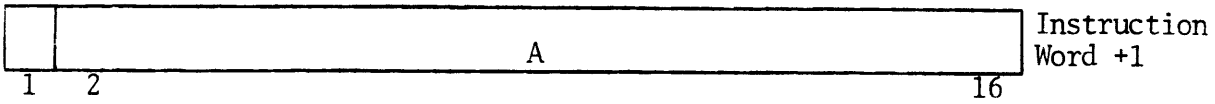
Memory reference instructions take one of three formats, depending on the type of instruction and the addressing mode. A single-word format is used for all standard instructions in the sectored addressing modes, in relative mode when S=0, and for procedure-relative addressing (S=1, D is between -240 and +255):



The op code identifies the type of instruction. The I specifies indirect addressing, X specifies indexing, and D is an address displacement value. These fields determine the effective address of the operand. (For more information on techniques of effective address formation, see the descriptions of the addressing modes.)

A two-word format is used in relative mode for the long reach and stack relative classes of special addressing:

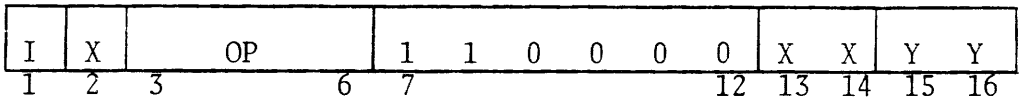




↑
32R: 0
64R: MSB of Address

In this addressing class, the D field in the first word of the instruction contains op code extension bits (XX) used by extended instructions such as DFLD, etc., and a special code (YY) that specifies the special addressing class. (The address value A is in the second word of the instruction.)

For the stack postincrement and predecrement classes of special addressing, only the first word is required:

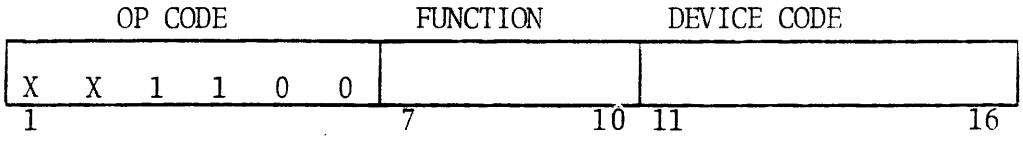


The address displacement value is obtained from the stack register, eliminating the need for a second word.

Instructions of the extended type (DFLD, etc.) always use one of the special addressing formats and thus may be used only in relative addressing mode.

I/O Instructions

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 channel address, and so on.



PROGRAMMING CONVENTIONS

The assembly program recognizes mnemonics and other symbols to construct complete instruction words and organize them into a program. Refer to Appendix C for Op Code Mnemonics. 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.

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 bits 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 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 is written as, LDA 13, the assembler generates '004015, because it interprets all unqualified numbers as decimal. On the other hand, LDA '13 assembles as '004013 and actually accesses location '13.

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

```
LDA* '13
```

assembles as '104013, and produces indirect addressing. Placing ",1" following the memory address causes modification of the address by the contents of the index register. Hence, LDA* '13,1 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 specifies LDA '4000. The assembled form of this instruction depends upon the curent 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.

Other assembler syntax conventions are mentioned in the later description of special addressing techniques. (For example, LDA% '13 generates a two-word instruction with the address value in the second word.)

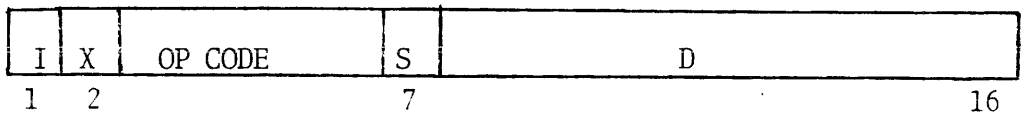
SYMBOLIC ADDRESSING

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 Macro Assembler manual.

MEMORY ADDRESSING TECHNIQUES

This section explains the procedures used to calculate the effective address of all memory reference instructions. The program controls the effective address calculation by the information given in instruction and address words, 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 the first word of every memory reference instruction whether the effective address is used for storage or retrieval of an operand or to alter program flow:



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. (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.)

There are six distinct addressing classes: sectored, relative, and four varieties of extended addressing. The class is determined by the addressing mode, the sector bit, and the value in the D field of the instruction, as shown in Table 3-1. Effective address formation for each class will be described separately.

Sectored Addressing

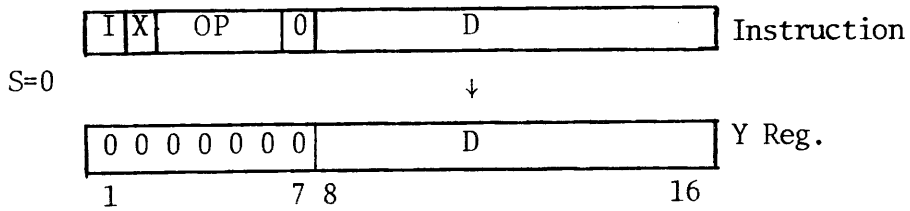
In the sectored addressing modes, and in the relative modes when S=0, memory is organized in sectors of 512 words each. (8K memory consists of sixteen 512-word sectors.) Prime memory reference instructions use the 9-bit displacement field, D, to specify memory location within a sector.

The simplest case is when the indexing and indirect bits are both 0. The effective address of the instruction then depends only on the value in the D field and the sector bit.

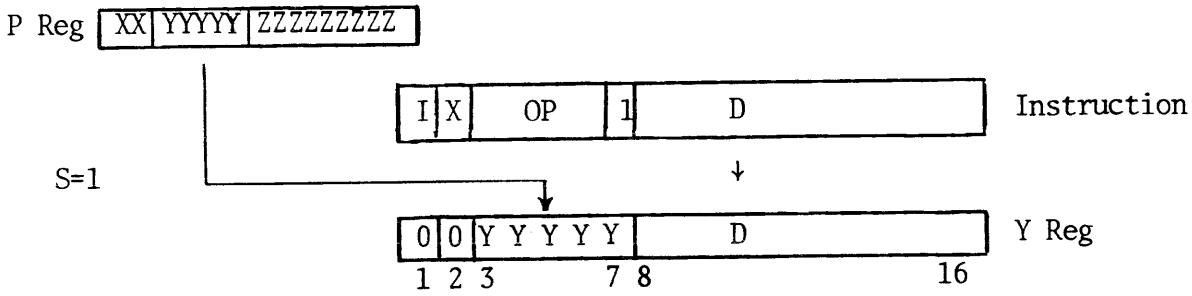
Table 3-1. Addressing Classes

Addressing Mode	S	D	Addressing Class	No. of Words
16S 32S	0, 1	0-777	Sectored	1
32R 64R	0	0-777	Sectored	1
32R 64R	1	-240 to +255	Procedure Relative	1
32R 64R	1	-256 to -241 (op code extension rather than address value)	<u>Extended Addressing</u> Long Reach Stack Relative Stack Postincrement Stack Predecrement	 2 2 1 1

If the sector bit is a 0, the displacement field is used directly to specify one of '777 locations in sector 0. This is accomplished by transferring bits 8-16 of the instruction to corresponding bits of the memory address (Y) register and forcing 0's in bits 1-7 of Y:



When the sector bit is a 1, the D field is concatenated with high-order bits from the program counter (P register), indicated by the notation P|D. For example, in 16S mode a current sector address is formed as follows:



The displacement value from the instruction word itself is transferred to Y 8 through 16; bits 3 through 7 of the P register are transferred to corresponding bits of Y. Bits 1 and 2 of Y are set to zeroes in 16S mode.

Indexing: If the X bit is a 1, the contents of the index register (location 0) are summed with the address being formed, by 2's complement addition. (The X register may be preset by the program to any value between -32768 and +32767.) The result of the addition is then truncated to the number of significant bits permitted by the addressing mode. Higher-order bits of the result are filled with zeros. The X bit of the instruction word usually specifies pre-indexing, that is, the X register is added to the displacement field before the next memory access. In certain special cases, described later, the X bit of the instruction word specifies post-indexing, in which adding of the X register is postponed until an indirect address word is accessed.

Indirect Addressing: 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. In 64R mode, address words do not contain an I bit, so only one level of indirection (specified by the instruction word itself) is possible.

Address Truncation: After effective address formation is complete, the resulting address in the Y register is truncated to the number of address bits appropriate to the addressing mode in effect:

Mode	Address Bits	Size of Memory Addressable
16S	14	16K
32S 32R	15	32K
64R	16	64K

Higher order bits of the Y register are forced to zeroes. Thus, an address cannot be formed that addresses a memory location beyond the range of the current addressing mode. However, it is possible for an executing program to increment the program counter out of the current range (instead of overflowing to zero).

Any addressing mode operates legally with as little as 4K memory. However, in any mode, it is possible to form an address that is beyond the range of the available memory. When this happens, a missing memory module trap occurs.

16K Sectored Addressing

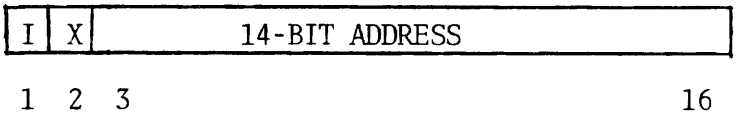
The 16K sectored mode (abbreviated as "16S" mode) is the default mode of operation when the machine is first turned on or the computer is cleared from the control panel. In this mode, indexing may occur both before and after indirect references since an absolute address requires only 14 bits - leaving 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 following table lists the address word configurations and the calculation procedures for all cases in 16S mode. P is the contents of the program counter prior to the instruction fetch, the symbol P|D represents the sectored address formed by concatenation, X is the contents of the index register, A is an absolute address, and I (expression) is the result of the indirect chain beginning with access to the location addressed by (expression). Assembler syntax is shown for each case as an LDA to location Q in sector 0 or location R in the current sector.

I	X	S	EA	Assembler Notation	Type
0	0	0	D	LDA Q	Direct
0	1	0	D+X	LDA Q, 1	Indexed
1	0	0	I(D)	LDA Q, *	Indirect
1	1	0	I(D+X)	LDA Q, 1*	Indirect, preindexed
0	0	1	P|D	LDA R	Direct
0	1	1	P|D + X	LDA R, 1	Indexed
1	0	1	I(P|D)	LDA R, *	Indirect
1	1	1	I(P|D + X)	LDA R, 1*	Indirect, preindexed

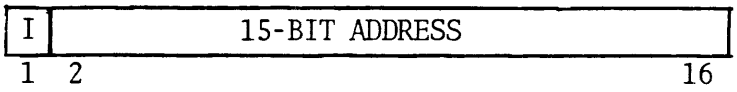
NOTE: The asterisk, meaning indirection, may be appended to the mnemonic, as in LDA* R (indirect) or LDA* R,1 (indirect, preindexed).

All indexing cycles indicated in the table are pre-indexing. Post-indexing occurs if the X bit of an indirect address word is set. Indirect address words in 16S mode are in the following form:



32K Sectored Addressing

The sectored modes for 32K (32S and 32R with S=0) extend the addressing range to 32K by using a 15-bit address field in each address word:



Such address words have no X bit. Therefore, except for a special case, indexing must be done after all levels of indirect addressing have been performed. The special case permits indexing to occur prior to the first indirect access if the sector bit is zero and the displacement is less than '100.

Address word configurations and calculations for all cases of indexing and indirect addressing are:

I	X	S	D	EA	Assembler Notation	Type
0	0	0	0 to '777	D	LDA Q	Direct
0	1	0	"	D+X	LDA Q, 1	Indexed
1	0	0	"	I(D)	LDA Q, *	Indirect
1	1	0	0 to '77	I(D+X)	LDA Q, 1*	Indirect, preindexed
1	1	0	'100 to '777	I(D)+X	LDA Q, *1	Indirect, postindexed
0	0	1	0 to '777	P|D	LDA R	Direct
0	1	1	"	P|D+X	LDA R, 1	Indexed
1	0	1	"	I(P|D)	LDA R, *	Indirect
1	1	1	"	I(P|D)+X	LDA R, *1	Indirect, postindexed

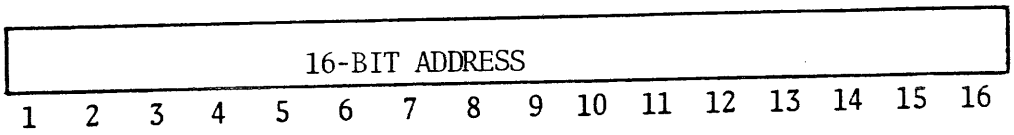
Relative Addressing

In 32R or 64R addressing mode, when the sector bit is a 1 and the displacement field is a two's-complement number in the range from -240 to +255 (decimal), addressing is relative to the current program counter value. The effective address is formed by adding the value of the displacement field to the program counter value plus 1, and then performing indirection and indexing as specified by the I and X bits:

I	X	EA	Assembler Notation	Type
0	0	P+1+D	LDA D	Direct
0	1	P+1+D+X	LDA D, 1	Indexed
1	0	I(P+1+D)	LDA D, *	Indirect
1	1	I(P+1+D)+X	LDA D, *1	Indirect, postindexed

This addressing method cannot be used by two-word instructions (DFLD, etc.), all of which have, by definition, a displacement field out of the -240 to +255 range. (See 'Extended Addressing'.

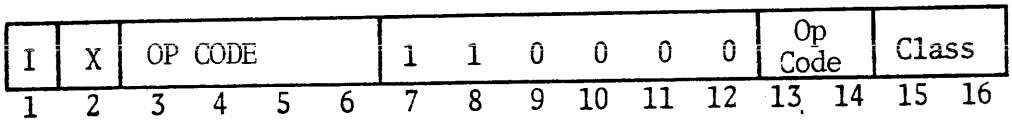
The 32R and 64R modes are identical in this class except for the interpretation of indirect address words. 32R is the same as 32S. 64R address words are interpreted in this way:



Since the entire word is used for addressing, only one level of indirection is possible.

Extended Addressing (32R and 64R Modes)

In 32R and 64R modes, when the sector bit is a 1 and the displacement field is a two's-complement number below -240 decimal, the instruction word format is interpreted as follows:



The I and X and op-code bits perform the same functions as in sectored mode, but Bits 7-12 no longer indicate displacement; they exhibit the fixed pattern 11000. Bits 13 and 14 are op-code extensions. For instructions of the basic set, they are always 00. Other values extend the normal op-code to define the floating point instructions and the Prime 300 extended instruction set. Table 3-2 shows the combinations that are currently assigned. Unassigned combinations cause an unimplemented instruction interrupt.

Bits 15 and 16, in combination with I and X bits, select one of the extended addressing classes:

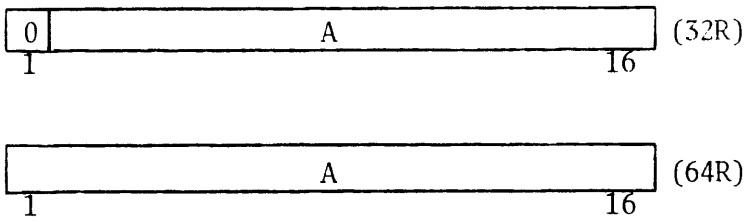
I	X	Class Code (Bits 15, 16)			
		0	1	2	3
0	0	Long Reach	Stack Relative	Stack Postincrement	Stack Predecrement
0	1				
1	0				
1	1	Indirect, Preindexed	Indirect Preindexed	Long Reach, Indirect, Postindexed	Stack Relative, Indirect, Postindexed

Table 3-2. Extended Instruction Op Codes

Op Code Bits 3-6	Extended Op-Code Bits 13-14			
	00	01	10	11
00	Generic	--	--	--
01	JMP	EAA	XEC	ENTR
02	LDA/DLD	FLD	DFLD	JEQ
03	ANA	--	--	JNE
04	STA/DST	FST	DFST	JLE
05	ERA	--	--	JGT
06	ADD/DAD	FAD	DFAD	JLT
07	SUB/DSB	FSB	DFSB	JGE
10	JST	--	CREP	--
11	CAS	FCS	DFCS	--
12	IRS	--	--	--
13	IMA	--	--	--
14	I/O Group	--	--	--
15	STX	FLX	JDX	JIX
35*	LDX	--		JSX
16	MPY	FMP	DFMP	--
17	DIV	FDV	DFDV	--

* Including X Bit

The long reach and stack relative classes force two-word instructions. The word following the instruction must contain an address value, A, in the following format:



Stack postincrement and predecrement classes require only the first word of the extended format since the address value is obtained from the S register.

Long Reach (Two-Word): The value of A in location P+1 is used for effective address calculation in combination with the I and X bits of the instruction word itself:

Class Bits 15, 16	I	X	EA	Assembler Notation	Type
0	0	0	A	LDA% A	Direct
0	0	1	A+X	LDA% A, 1	Indexed
0	1	0	I(A)	LDA% A, *	Indirect
0	1	1	I(A+X)	LDA% A, 1*	Indirect, preindexed
2	1	1	I(A)+X	LDA% A, *1	Indirect, postindexed

In 64R mode, the resulting EA is the final effective address (no further indirection is possible because all 16 bits of EA are significant.) In 32R mode, indirection may continue until an EA is formed which contains a 0 in bit 1.

Stack Relative (Two-Word): This class is identical to two-word long reach except that the contents of the stack pointer are added to A during the initial effective address calculation. Indexing and indirection take place under control of the I and X bits of the instruction word:

Class Bits 15, 16	I	X	EA	Assembler Notation	Type
1	0	0	A+S	LDA @+A	Direct
1	0	1	A+S+X	LDA @+A, 1	Indexed
1	1	0	I(A+S)	LDA @+A, *	Indirect
1	1	1	I(A+S+X)	LDA @+A, 1*	Indirect, preindexed
3	1	1	I(A+S)+X	LDA @+A, *1	Indirect, postindexed

NOTE:

S = Contents of stack register.

Stack Postincrement, Predecrement: These classes use the stack pointer as the address displacement, and perform an auxiliary post-increment or predecrement of the pointer. Instructions using these address methods are always 1-word instructions, even those such as DFLD that are normally two-word instructions. The I and X bits are interpreted as follows:

Class Bits 15, 16	I	X	EA	Auxiliary Action	Assembler Notation	Type
2	0	0	S	S+1 → S	LDA @+	Postincrement
2	0	1	I(S)+X	S+1 → S	LDA @+,1	Postincrement, indirect, post-indexed
2	1	0	I(S)	S+1 → S	LDA @+, *	Postincrement, indirect
3	0	0	S-1	S-1 → S	LDA -@	Predecrement,
3	0	1	I(S-1)+X	S-1 → S	LDA -@,1	Predecrement, indirect, post-indexed
3	1	0	I(S-1)	S-1 → S	LDA -@, *	Predecrement, indirect

In 64R mode, address calculation is complete after the first level of indirection. In 32R mode, further indirect cycles proceed if the indirect word I bit is set.

Push-Pop: Classes 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,... and it is assumed that S always points to the next open location, then class 2 is push and class 3 is pop. On the other hand, if the stack is viewed as locations N, N -1, N -2,... and S always points to the last filled location, then class 3 is push and class 2 is pop.

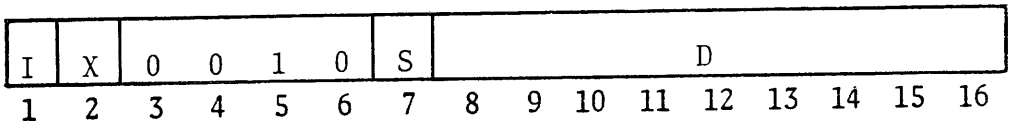
SECTION 4

STANDARD INSTRUCTION SET

This section describes the computer instructions that are implemented in all Prime central processors.

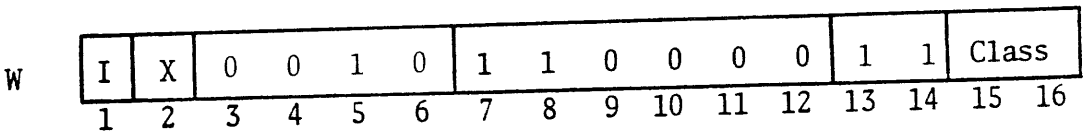
For easy reference, all instruction descriptions are presented in the following format: mnemonic and instruction name at top left and octal op code at top right, over a diagram of the binary word(s) into which the mnemonic is assembled by the Macro Assembler:

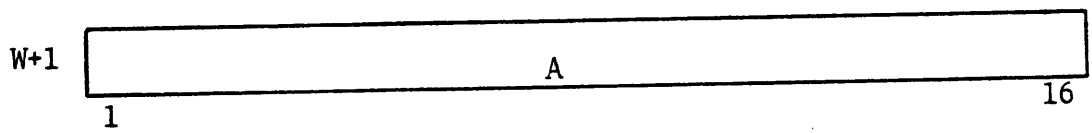
Mnemonic	Description	Octal op code
LDA	Load A	'02



For extended instructions, the octal code of the op code extension in bits 13-14 appears below the octal op code:

		Op-code extension	
ENTR	Enter Recursive Procedure Stack		'01 3





The following abbreviations are used in instruction words to indicate variable information:

I	Indirect Addressing Bit
X	Indexed Addressing Bit
S	Sector Bit
D	Address Displacement (Single-word instructions)
A	Full Word Address (Two-word instructions)
Class	Extended Addressing Class Code

ADDRESS MODE SELECTION

The following instructions alter the current addressing mode. See 'Memory Addressing Techniques' for detailed information on effective address formation, indexing, and indirect addressing in each mode.

E16S Enter 16K Sectored Mode '000011

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

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

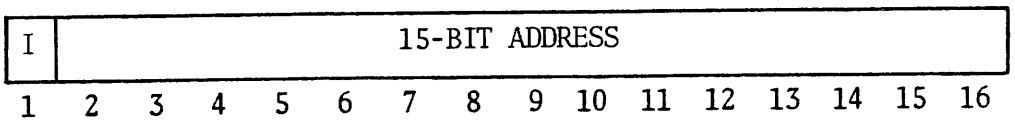
I	X	14-BIT ADDRESS													
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Indexing may be performed before or after indirect references.

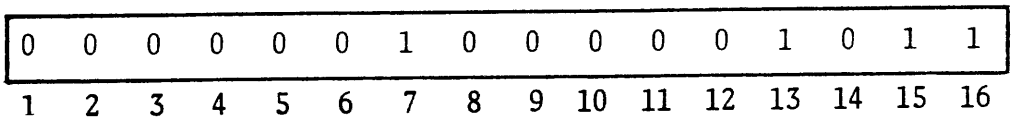
E32S Enter 32K Sectored Mode '000013

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

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



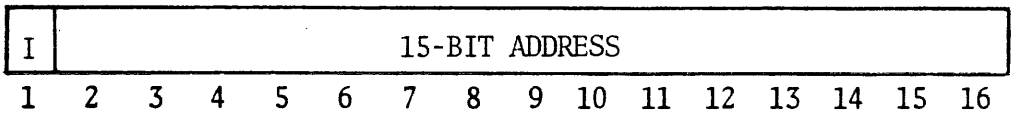
E32R Enter 32K Relative Mode '001013



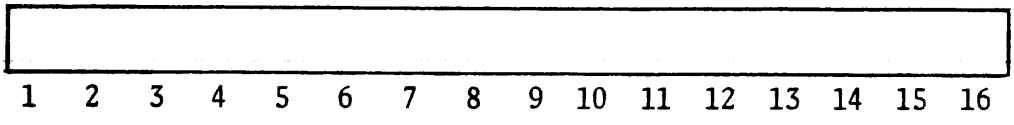
In subsequent effective address calculations if S = 0 proceed as in 32S mode; if S = 1, interpret D field as 2's complement number to determine type of addressing:

D Field	Addressing Type
-240 to +255	Procedure Relative
-256 to -241	Extended Addressing:
	Long Reach (two word)
	Stack Relative (two word)
	Stack Post-Increment
	Stack Pre-Decrement

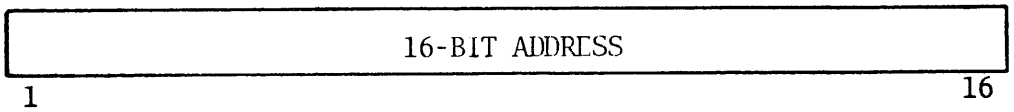
(See 'Memory Addressing Techniques' for details.) Interpret indirect address words this way:



E64R Enter 64K Relative Mode '001011



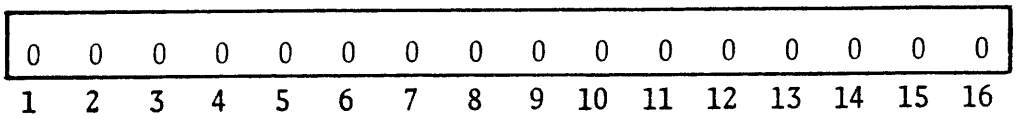
In subsequent effective address calculations, proceed as in E32R mode but interpret indirect address words this way:



CONTROL INSTRUCTIONS

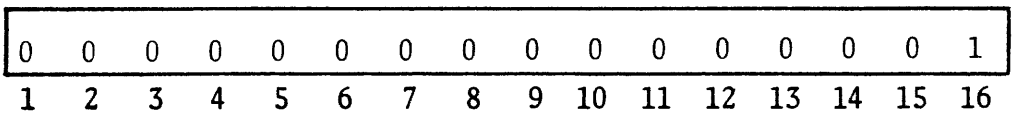
Here are several miscellaneous control instructions and those associated with parity errors and the status keys.

HLT Halt '000000



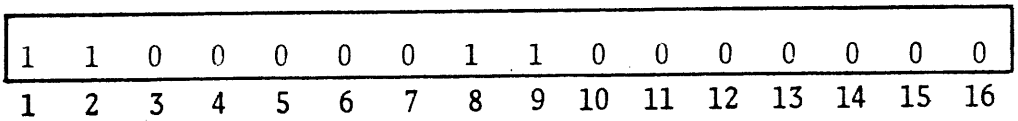
Halt the processor with the STOP indicator lit on the control panel and P pointing to the next instruction in sequence (the instruction that would have been executed had the HLT been replaced by a no-op). The data lights display the next instruction, and the address lights display the instruction OTA'1720.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

NOP No Operation '000001



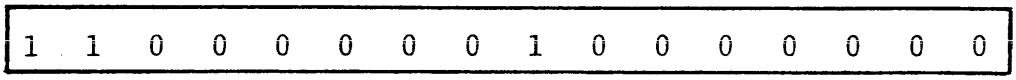
Do nothing but go on to the next instruction.

SCB Set C Bit '140600



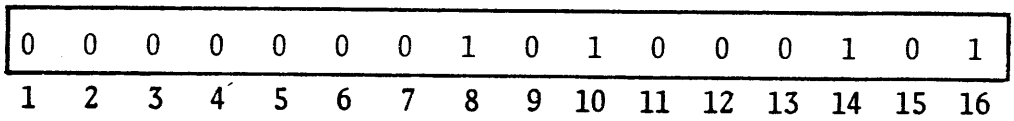
Set C.

RCB Reset C Bit '140200



Reset C.

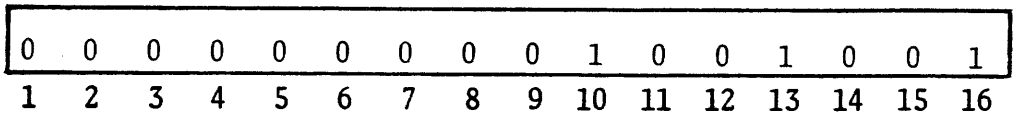
SVC Supervisor Call '000505



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.

CEA Compute Effective Address '000111

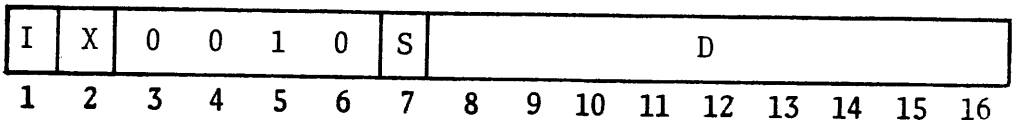


Calculate the effective address indicated by the contents of A interpreted as an address word in the current addressing mode, and place the result in A.

LOAD AND STORE INSTRUCTIONS

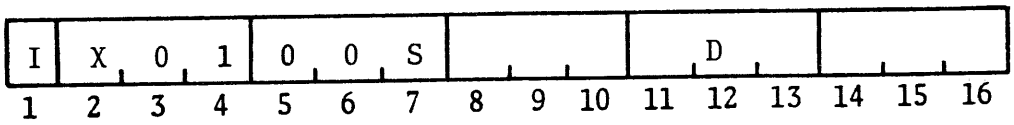
There are five instructions for moving data between memory and the A and the index registers.

LDA Load A '02



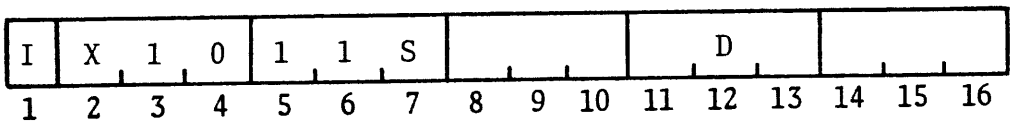
Load the contents of location EA into A. The contents of EA ar unaffected, the previous contents of EA are lost.

STA Store A '04



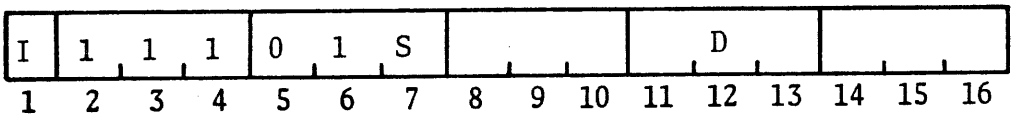
Store the contents of A in location EA. The contents of A are unaffected, the previous contents of EA are lost.

IMA Interchange Memory and A '13



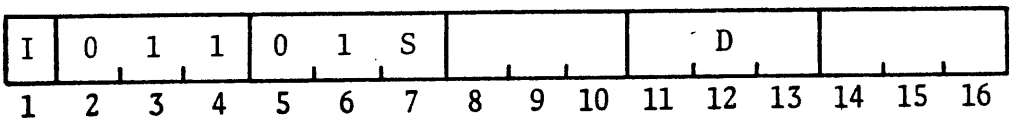
Store the contents of A in location EA and load the original contents of location EA into A.

LDX Load Index Register '35



Load the contents of location EA into the index register. The contents of EA are unaffected, the previous contents of the index register are lost. This instruction cannot itself specify indexing, although an address word retrieved in the effective address calculation may do so.

STX Store Index Register '15

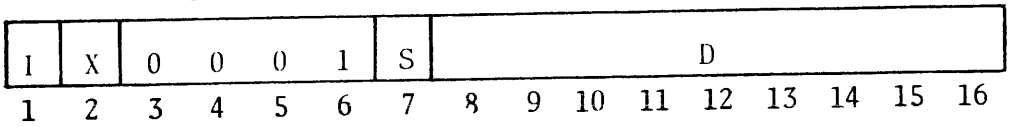


Store the contents of the index register in location EA. The contents of the index register are unaffected and the previous contents of EA are lost. This instruction cannot itself specify indexing, although an address word retrieved in the effective address calculation may do so.

JUMP INSTRUCTIONS

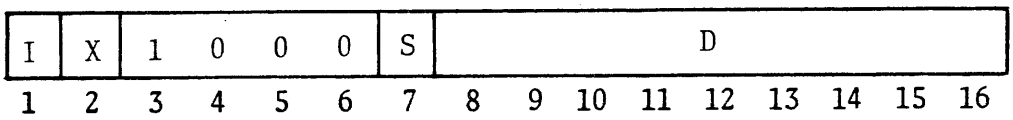
Two instructions allow the programmer to alter the normal program sequence by jumping to a location calculated from the EA.

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 a return address (the next address to be executed) in location EA. If the current instruction is a single-word instruction, store P+1 in location EA; if the current instruction is a double-word instruction, store P+2 in EA. Hence, location EA receives the address of the next executable location following the JST instruction. Load EA+1 (or EA+2) into P. Take the next instruction from location EA+1 (or EA+2) and continue sequential operation from there. Interrupts are inhibited for one instruction time following a JST.

The return address is truncated according to the addressing mode before it is stored, and higher-order bits of the memory location are not altered. It is thus possible to preset the I or X bits of such locations:

Mode	Preset Allowed
16S	I, X
32S, 32R	I
64R	-

The usual procedure for calling a subroutine is to use a JST with an effective address that specifies the subroutine's starting location. Since P+1 (or P+2) is saved at the entry point, a subsequent return can be made to the instruction following the JST by an indirect JMP through the entry point.

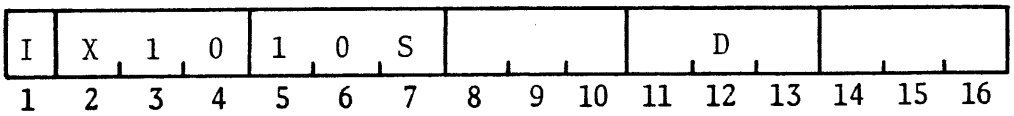
SKIP INSTRUCTIONS

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

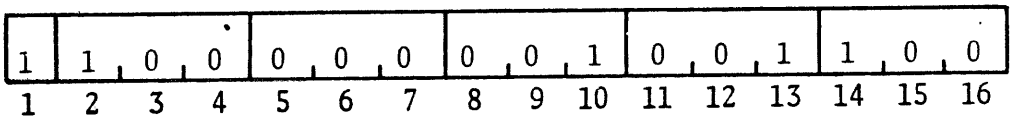
These instructions allow the program to keep a count in a memory location and to count the contents of the index register up or down. The skip test is always for a zero result. The instructions are used to count loop iterations or successively to modify a word for a series of operations.

IRS Increment Memory, Replace, and Skip '12



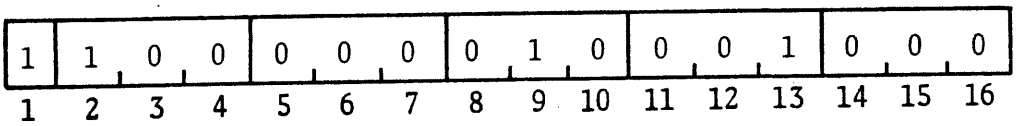
Add 1 to the contents of location EA and place the result back in EA. Skip the next instruction in sequence if the result is zero.

IRX Increment and Replace Index '140114



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.

DRX Decrement and Replace Index '140210

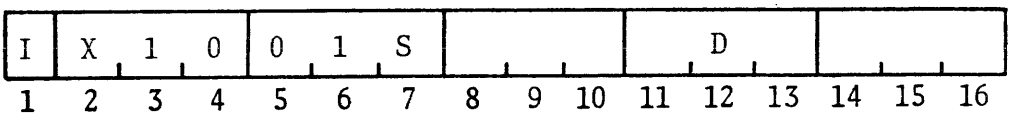


Subtract 1 from the contents of the presently selected index register and place the result back in that register. Skip the next instruction in sequence if the result is zero.

Compare

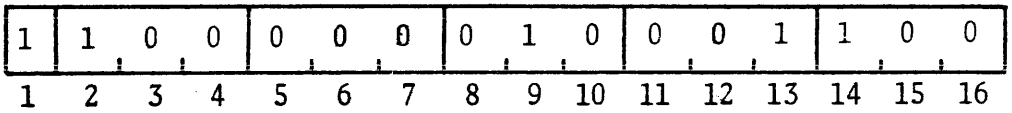
These two instructions do an algebraic comparison of the number in A with zero or a number in memory. They use a three-way test to allow skipping one or two locations as well as not skipping at all.

CAS Compare A and Skip '11



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.

CAZ Compare A with Zero '140214

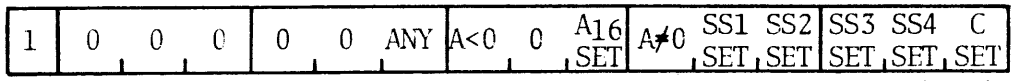


Compare the contents of A (fixed or floating) algebraically with zero and act on the result as follows: if A > 0, execute the next instruction in sequence; if A = 0, skip the next instruction in sequence; if A < 0, skip the next two instructions in sequence.

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

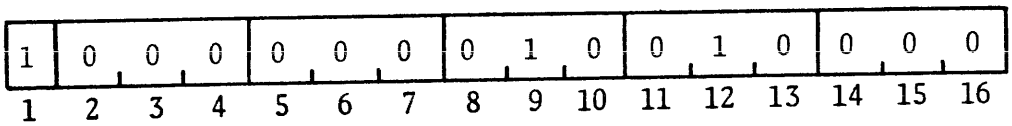


Skip the next instruction in sequence if bit 7 is 1 and any of the conditions specified by 1s in bits 8 and 10-16 is satisfied, or if bit 7 is 0 and none of the conditions specified by 1s in bits 8 and 10-16 is satisfied. (The conditions listed in the format box are those selected by 1s.) The various 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	Non (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	0	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

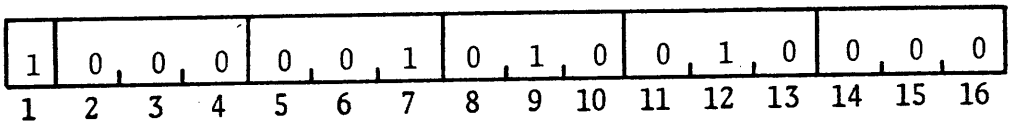
Skip conditions can be combined using SKP and giving the bit 7-16 configuration for the combination in the address field.

SGT Skip if A Greater Than Zero '100220



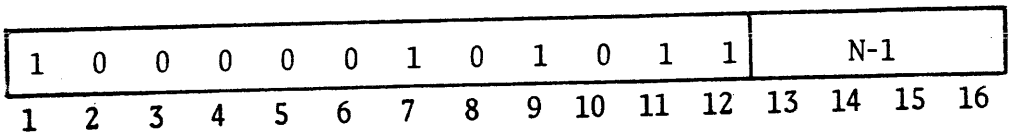
If the number contained in A (fixed or floating) is greater than zero, skip the next instruction in sequence.

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.

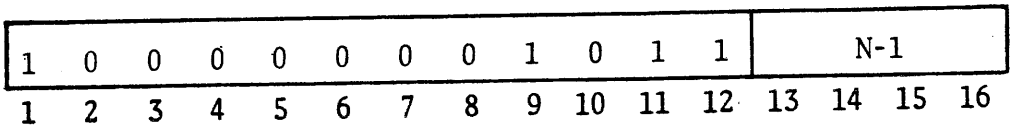
SASn Skip on A Bit Set '10126-
'10127-



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.

SARn 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 octal equivalent of the bit number minus one.

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

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

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

SNRn Skip on Sense Switch Reset '10024-
'10025-

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

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

Summary of Sense Switch and Bit Test Instruction Op Codes

SNS 1 - 101240	SNR 1 - 100240
SNS 2 - 101241	SNR 2 - 100241
SNS 3 - 101242	SNR 3 - 100242
SNS 4 - 101243	SNR 4 - 100243
SNS 5 - 101244	SNR 5 - 100244
SNS 6 - 101245	SNR 6 - 100245
SNS 7 - 101246	SNR 7 - 100246
SNS 8 - 101247	SNR 8 - 100247
SNS 9 - 101250	SNR 9 - 100250
SNS10 - 101251	SNR10 - 100251
SNS11 - 101252	SNR11 - 100252
SNS12 - 101253	SNR12 - 100253
SNS13 - 101254	SNR13 - 100254
SNS14 - 101255	SNR14 - 100255
SNS15 - 101256	SNR15 - 100256
SNS16 - 101257	SNR16 - 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

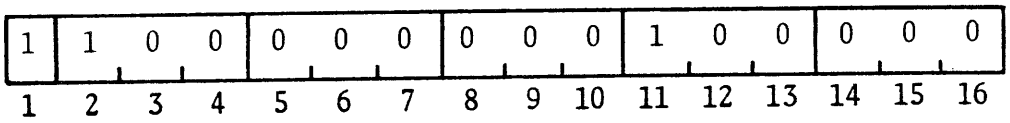
NOTE

The SMCS and SMCR skip instructions are described under "Machine Check".

REGISTER OPERATE

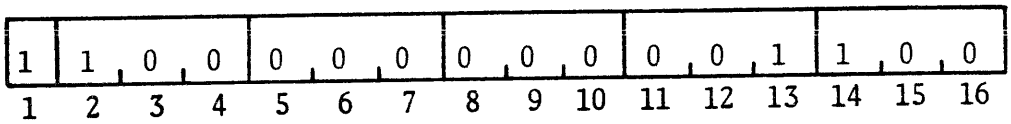
These instructions are simply for clearing the A and B registers and moving data between then.

CRA Clear A '140040



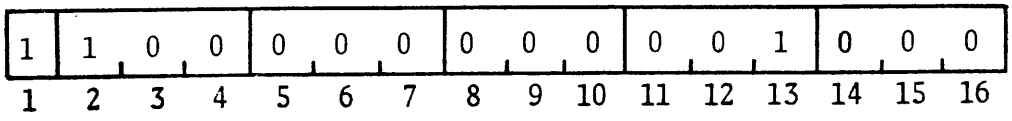
Change the contents of A to all 0s.

CRB Clear B '140014



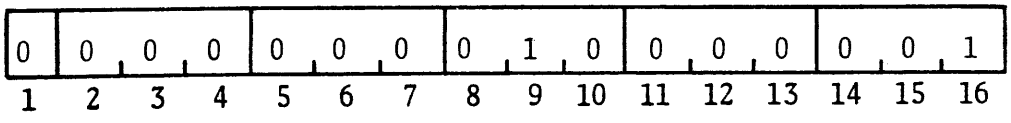
Change the contents of B to all 0s.

CRL Clear Long '140010



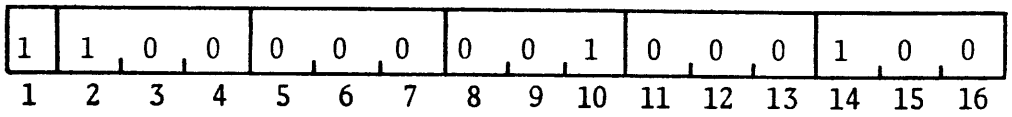
Clear A and B.

IAB Interchange A and B '000201



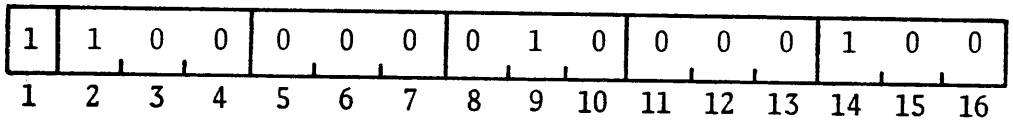
Move the contents of A to B and the contents of B to A.

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

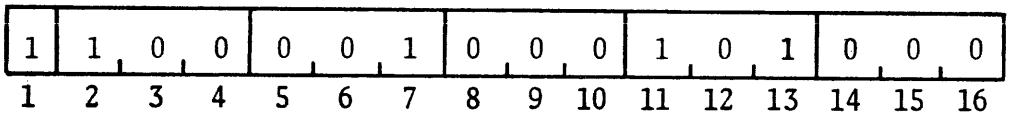


Move the contents of B to A and clear B. The original contents of A are lost.

BYTE MANIPULATION

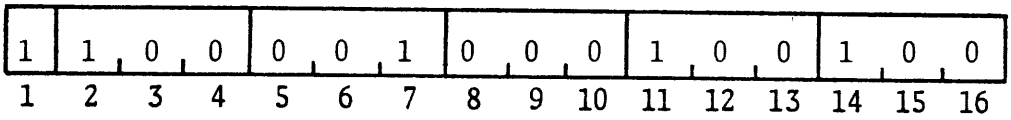
These instructions are for manipulating half words in A. They are useful for handling ASCII characters, 8-bit data bytes packed two to a word in memory, tables where half of each table location is used for the entry and the other half for a label, etc.

CAL Clear A Left '141050



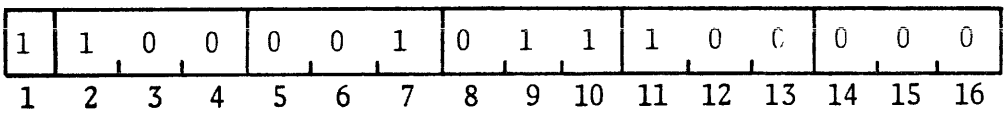
Clear A bits 1-8 without affecting bits 9-16.

CAR Clear A Right '141044



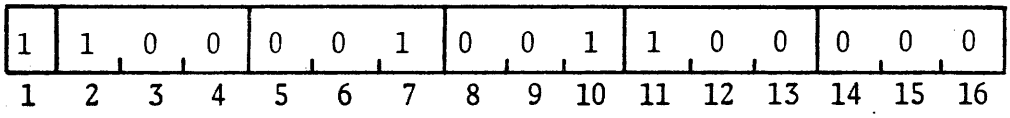
Clear A bits 9-16 without affecting bits 1-8.

ICA Interchange A '141340



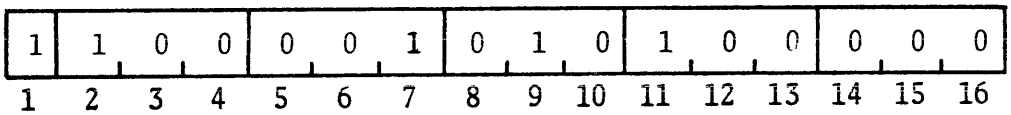
Interchange the two halves of A (move the contents of bits 1-8 to bits 9-16 and the contents of bits 9-16 to bits 1-8).

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.

SHIFT GROUP

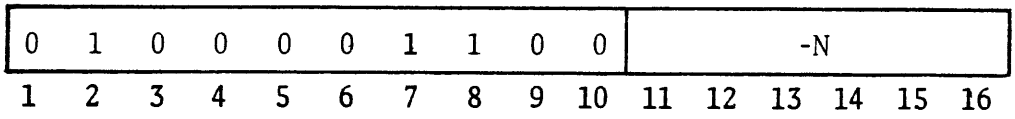
Shifting is the movement of the contents of a register bit-to-bit. The instructions in this group shift or rotate right or left the contents of A or the contents of A and B treated as a single register with A on the left. Although these instructions are similar in format and operation, functionally some 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.

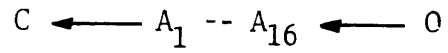
A right arithmetic shift fills the vacated left positions with the contents of the sign bit and does not change the sign. A left arithmetic shift includes the sign (A bit 1 only - B bit 1 is left out), but interprets a sign change as overflow and fills the vacated right positions with 0's. Hence, arithmetic 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 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/LGL A Left Logical '0414

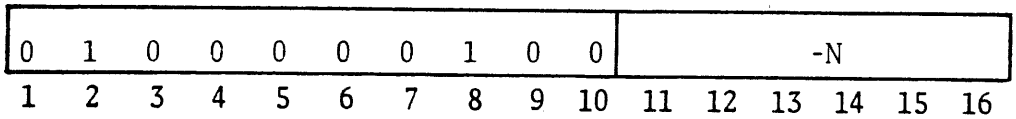


Shift the contents of A left N places, bringing 0s into bit 16; data shifted out of bit 1 is lost, except that the last bit shifted out is saved in C.

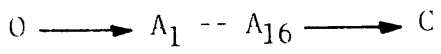


Note: The assembler recognizes ALL and LGL as equivalent.

ARL/LGR A Right Logical '0404

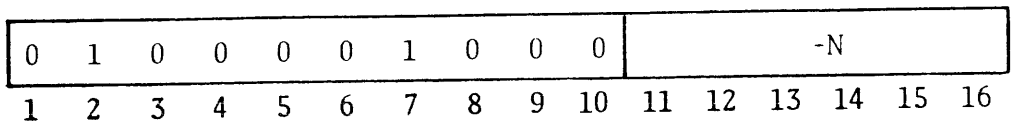


Shift the contents of A right N places, bring 0s into bit 1; data shifted out of bit 16 is lost, except that the last bit shifted out is saved in C.

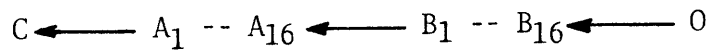


Note: The assembler recognizes ARL and LGR as equivalent.

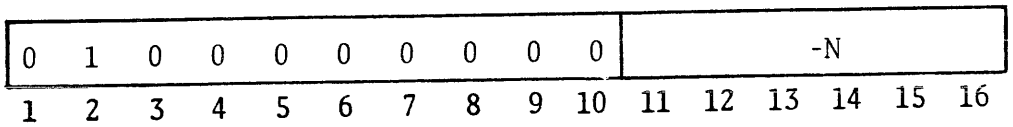
LLL Long Left Logical '0410



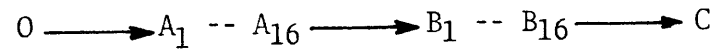
Shift the contents of A and B left N places, bringing 0s into A bit 16; B bit 16; B bit 1 is shifted into A bit 16; data shifted out of A bit 1 is lost, except that the last bit shifted out is saved in C.



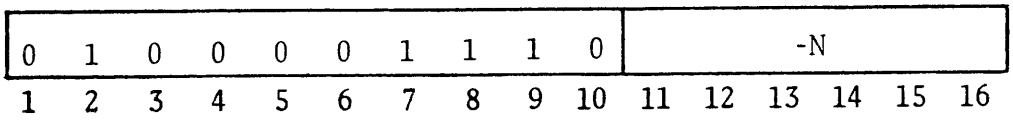
LRL Long Right Logical '0400



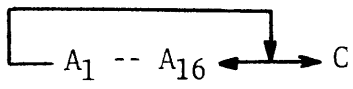
Shift the contents of A and B right N places, bringing 0s into A bit 1; A bit 16 is shifted into B bit 1; data shifted out of B bit 16 is lost, except that the last bit shifted out is saved in C.



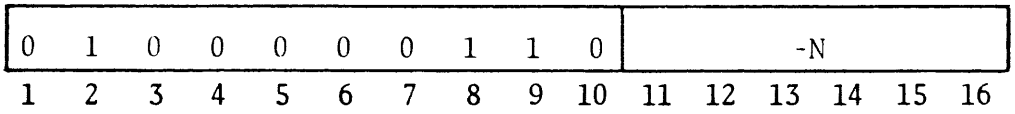
ALR A Left Rotate '0416



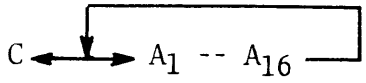
Shift the contents of A left N places, rotating bit 1 into bit 16. The last bit rotated back in at the right is also saved in C.



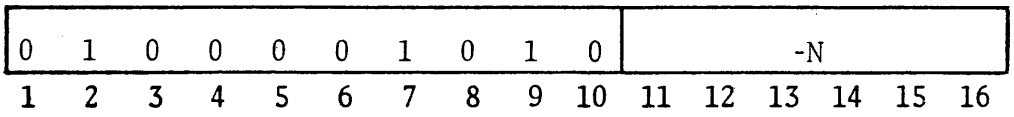
ARR A Right Rotate '0406



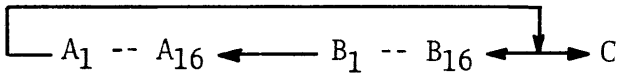
Shift the contents of A right N places, rotating bit 16 into bit 1. The last bit rotated back in at the left is also saved in C.



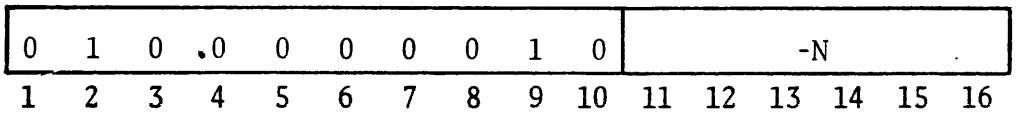
LLR Long Left Rotate '0412



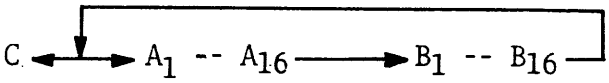
Shift the contents of A and B left N places, rotating A bit 1 into B bit 16; B bit 1 is shifted into A bit 16. The last bit rotated from A back to B is also saved in C.



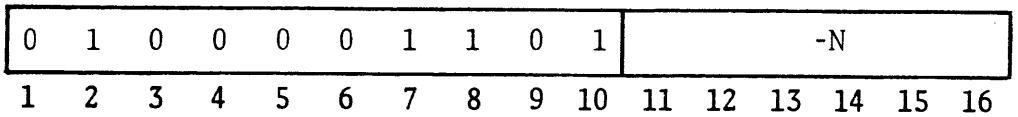
LRR Long Right Rotate '0402



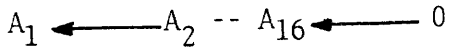
Shift the contents of A and B right N places, rotating B bit 16 into A bit 1; A bit 16 is shifted into B bit 1. The last bit rotated from B back to A is also saved in C.



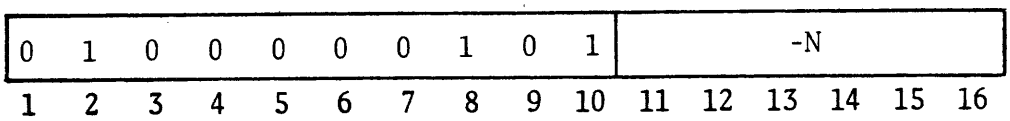
ALS A Left Shift '0415



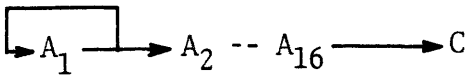
Shift the contents of A left arithmetically N places, bringing 0s into bit 16; data shifted out of bit 1 is lost. If the sign (bit 1) changes state, set C; otherwise reset it. A sign change indicates that a bit of significance (a 1 in a positive number, a 0 in a negative) has been shifted out of the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



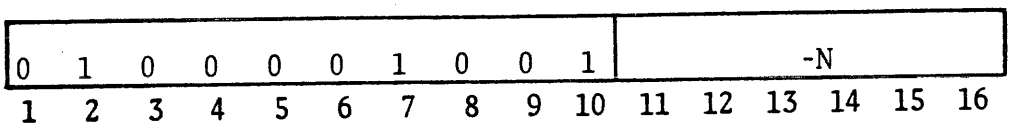
ARS A Right Shift '0405



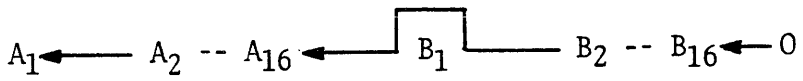
Shift the contents of A right arithmetically N places, leaving the sign (bit 1) unaffected, but shifting it into the magnitude part (0s in a positive number, 1s in a negative); data shifted out of bit 16 is lost, except that the last bit shifted out is saved in C.



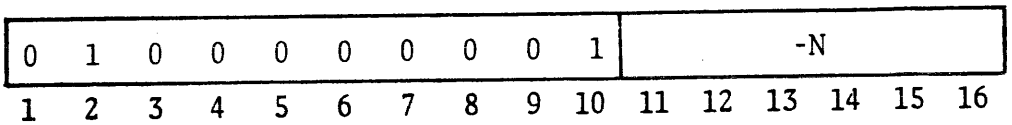
LLS Long Left Shift '0411



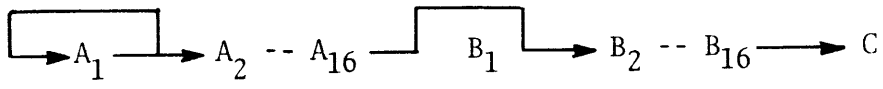
Shift the contents of A and B left arithmetically N places, bringing 0s into B bit 16 and bypassing B bit 1; B bit 2 is shifted in A bit 16; data shifted out of A bit 1 is lost. If the sign (A bit 1) changes state, set C; otherwise reset it. A sign change indicates that a bit of significance (a 1 in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



LRS Long Right Shift '0401



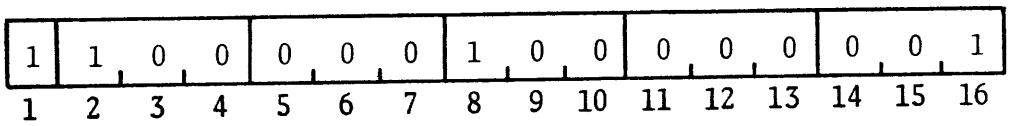
Shift the contents of A and B right arithmetically N places, leaving A bit 1 unaffected and bypassing B bit 1, but shifting the sign (A bit 1) into the magnitude part (0s in a positive number, 1s in a negative); A bit 16 is shifted into B bit 2; data shifted out of B bit 16 is lost, except that the last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



LOGIC

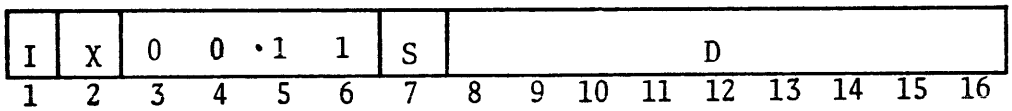
Besides the logical shift and rotate instructions described in the preceding section, the Prime 200 repertoire includes instructions for performing the complement, AND, and exclusive OR functions (the latter two being memory reference), and a group of instructions that "logicize" numbers. A number is 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 th contents of A in A (replace all 1s in A with 0s, all 0s with 1s).

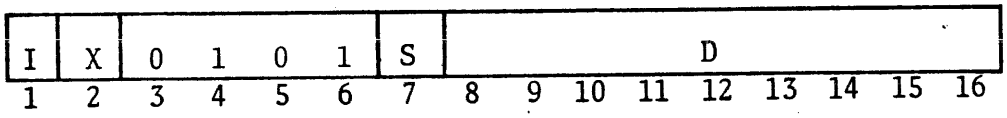
ANA And to A '03



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 to A '05

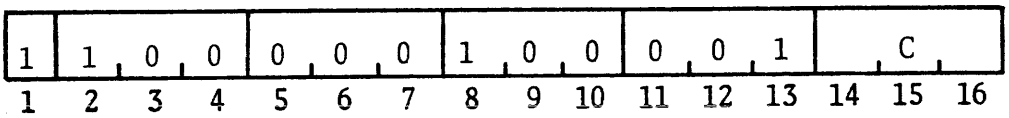


Form the exclusive OR function of the contents of location EA with the contents of A and place the result in A. A given bit of the result is 1 if the corresponding bits of the operands differ; otherwise the resulting bit is 0.

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

Logicize Group

Logicize '14041-



If the number in A satisfies the condition C, 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	≤0	'140411
LNE	2	≠0	'140412
LEQ	3	=0	'140413
LGE	4	≥0	'140414
LGT	5	>0	'140415
LF	6	False*	'140416
LT	7	True*	'140417

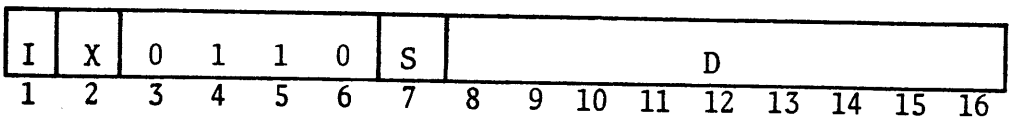
*These two instructions do not specify any condition: LF and LT simply set A to the truth values 0 and 1 respectively.

FIXED POINT 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.

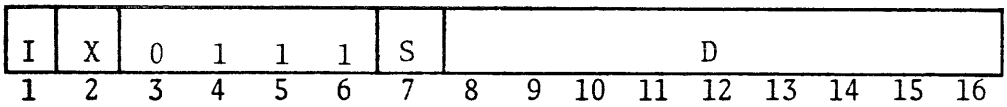
Besides the instructions for the basic operations and the arithmetic shift instructions, there are also instructions for manipulating signs, incrementing and decrementing numbers, negating, manipulating operands to allow multiplication and division using single length integers exclusively, and even instructions 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



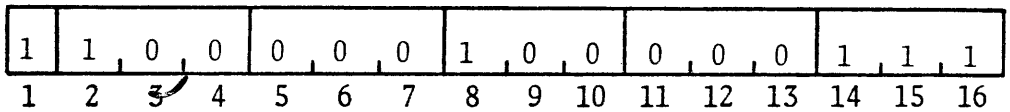
Add the contents of memory location EA to the contents of A and place the result in A. If the sum is $>2^{15}$ or $<-2^{15}$, set C; otherwise reset it. In the first overflow case, the result has a minus sign but a magnitude in positive form equal to the sum less 2^{15}; in the second the result has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

SUB Subtract '07



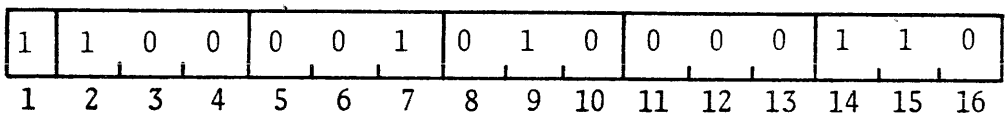
Subtract the contents of location EA from the contents of A and place the result in A. If the difference is $>2^{15}$ or $<-2^{15}$, set C; otherwise reset it. In the first overflow case, the result has a minus sign but a magnitude in positive form equal to the difference less 2^{15}; in the second the 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 produced by negation, negated is $0-2^{15}$, set C and give a result of -2^{15}; otherwise reset C.

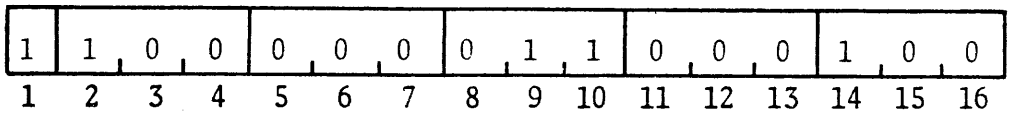
AOA/A1A Add One to A '141206



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.

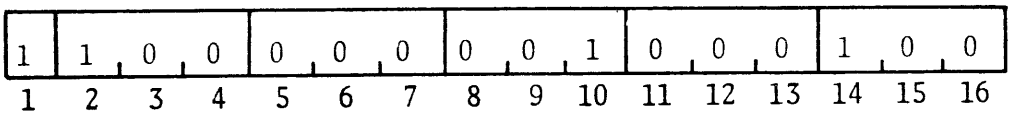
Note: The assembler recognizes the mnemonic A1A as equivalent to AOA.

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.

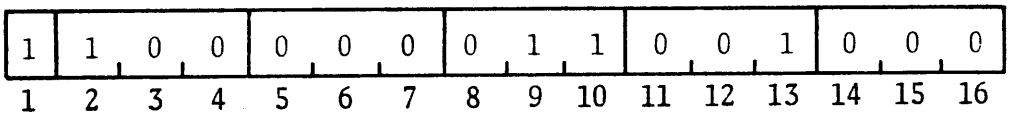
S0A/S1A Subtract One from A '140110



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.

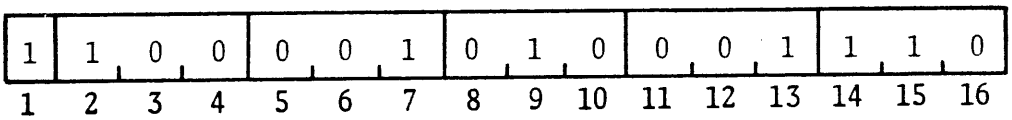
Note: The assembler recognizes the mnemonic S1A as equivalent to S0A.

S2A Subtract 2 from A '140310



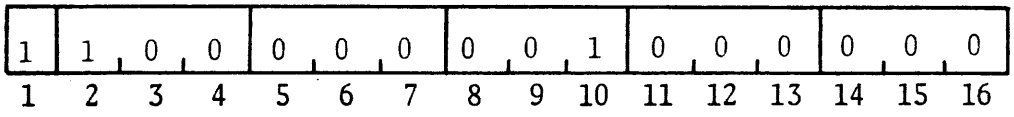
Subtract 2 from the contents of A in A. If the number decremented is $-(2^{15}-1)$ or -2^{15}, set C and give a result of $2^{15}-1$ or $2^{15}-2$; otherwise reset C.

ACA Add C to A '141216



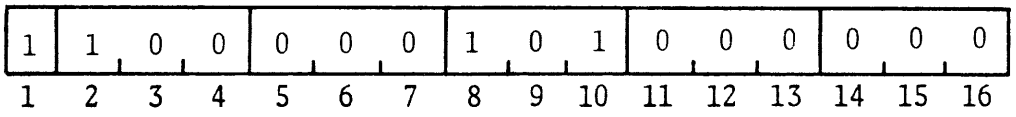
Add the contents of C to the contents of A in A (C is taken as being of the same order of magnitude as A bit 16). If the number originally in A is $2^{15}-1$, set C and give a result of -2^{15}; otherwise clear C.

SSP Set Sign Plus '140100



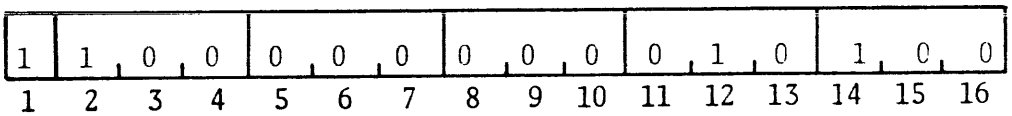
Reset A bit 1 to zero without affecting the rest of the register.

SSM Set Sign Minus '140500



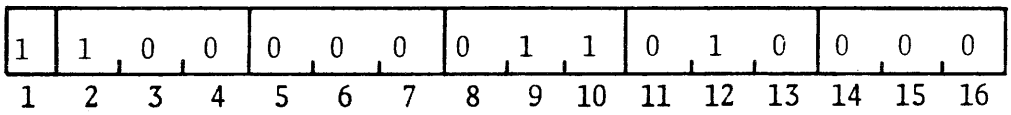
Set A bit 1 to one without affecting the rest of the register.

CHS Change Sign '140024



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

CSA Copy Sign of A '140320



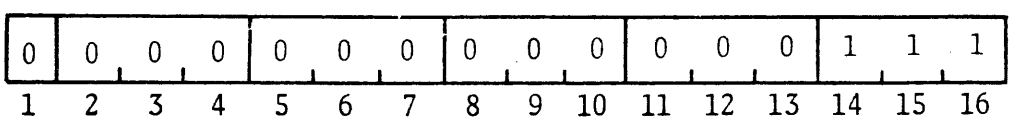
Make C equal to A bit 1, and reset A bit 1 (plus) without affecting the rest of the register.

Double Precision Mode

Single-precision mode is the normal CPU state after a power-on or master clear. The ADD, SUB, LDA and STA instructions operate as described above. Also available is an optional double-precision mode in which these operations act on double-word operands, with the B register serving as an extension of the A register. Double precision arithmetic, along with high-speed hardware multiply divide, is described in Section 5.

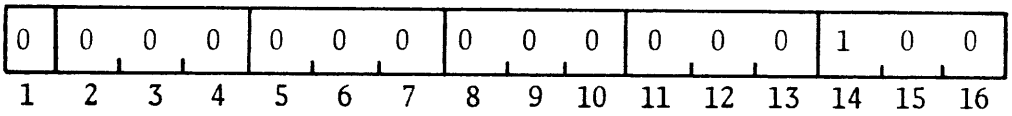
If double precision mode is selected in a CPU without the option, a UII interrupt results whenever an ADD, SUB, LDA or STA instruction is encountered. The MPY, DIV, PIM and PID instructions cause a UII interrupt in either mode. (See Section 3.) Because of the UII handling features of the assembler and linking loader, the user can use any of these instructions as though the option were present. During loading, the appropriate subroutines from the UII library are added to the program automatically. When such an instruction is encountered, a UII interrupt through location '66 directs the CPU to a UII library subroutine that emulates the unimplemented instruction using instructions of the standard set.

DBL Double Precision '000007



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 in Section).

SGL Single Precision '000005

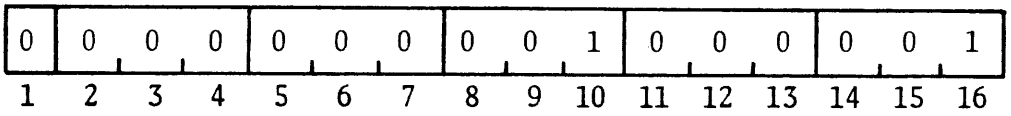


Return to single precision mode so that subsequently any LDA, STA, ADD or SUB instruction handles single precision operands.

Normalize

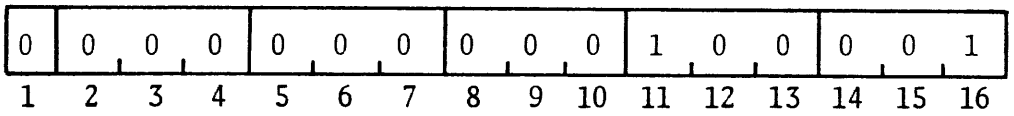
In order to prevent excessive information loss, it is necessary to adopt a consistent procedure for keeping floating point numbers in a normal form. The usual procedure is to make the fractions as large as possible, thus keeping the exponents as small as possible. The processor has these two instructions to facilitate the manipulation of floating point numbers in normal form.

NRM Normalize '000101



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

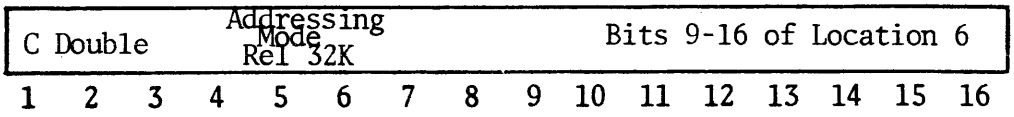


Load the contents of bits 9-16 of location 6 into A bits 9-16 and clear A bits 1-8.

By shifting until bit 2 differs from the sign, normalization produces a fraction in the range 1/2 to (1-LSB) or -(1/2 + 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.

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 Reserved
5-6 The current addressing mode as follows:

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

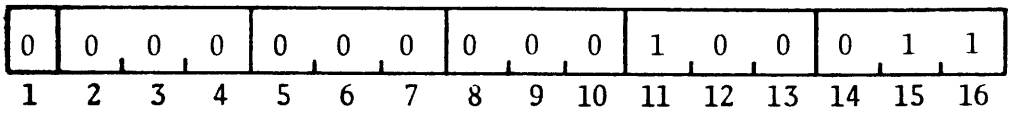
Note that a 1 in bit 5 indicates relative mode, a 1 in bit 6 indicates a 32K addressing space.

7-8 Reserved
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, e.g., giving a key word with a 1 in bit 2 places the arithmetic logic in double-precision mode; giving the word with a 0 in bit 2 limits the basic arithmetic operations to single-precision.

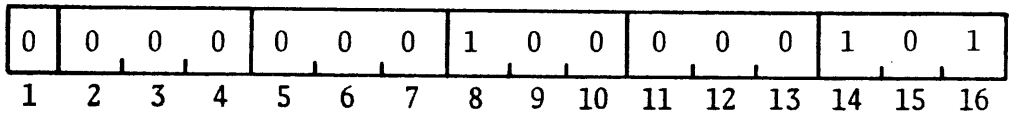
The processor has two instructions for reading and setting up the keys. The principle use of these instructions is for saving and restoring the keys in conjunction with program interrupts. Before doing its own operations, an interrupt service routine should save any parts of the register file it will 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.

INK Input Keys '000043



Read the key word defined above into A.

OTK Output Keys '000405



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.

SECTION 5

STANDARD INPUT/OUTPUT

This section describes the standard input/output functions common to all Prime CPU's: programmed input/output, vectored and compatible interrupts, and Direct Memory Access (DMA). Optional features such as Direct Memory Channels (DMC) and Direct Memory Transfers (DMT) are described in Section 6.

PROGRAMMED 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.

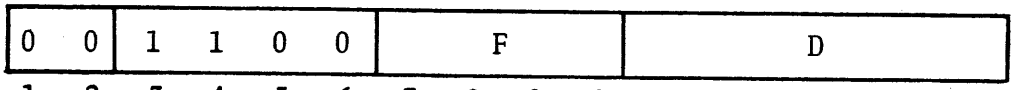
Typically a device interface has a 6-bit device selection network, Ready and Interrupt Enable flags, and logic nets that supply the device code, the device identification, and the number of the slot in which the interface is mounted. The selection network decodes

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.

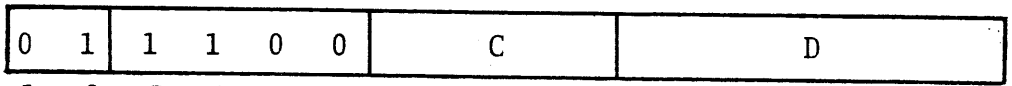
In the discussions of the various I/O devices in Chapter 3 and beyond, all instructions described are special cases of these four I/O instructions types:

OCP Output Control Pulse '14



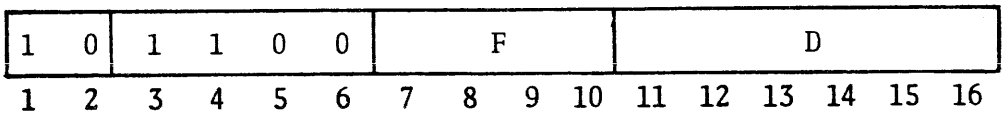
Send a control pulse for the function specified by F to device D.

SKS Skip if Satisfied '34



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

INA Input to A '54

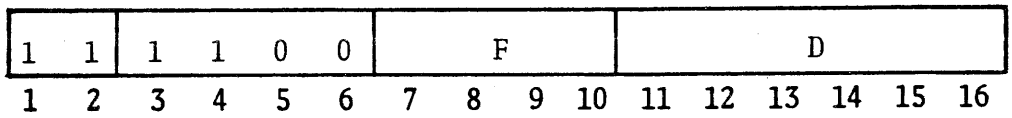


If the function F specifies a transfer for which Ready must be set, then if the Ready flag in device D is reset, do nothing but go on to the next instruction, whereas if Ready is set, perform the function F and skip the next instruction in sequence. To perform the function, the processor reads the information 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
```

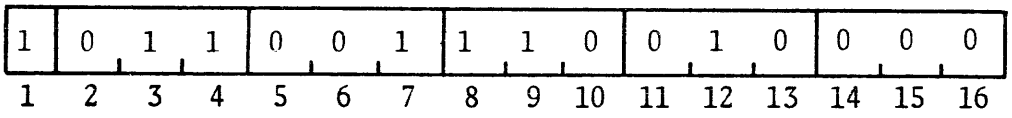
to stop the reader.

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 a 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

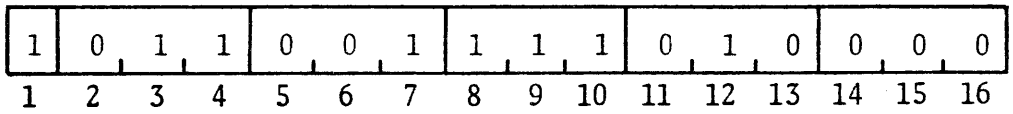
The program can communicate with the operator via the control panel by virtue of the fact that it can address the panel as an I/O device. With the following instructions, the program reads the contents of the switch register as data or as sense switches and loads a data register whose contents can be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

INA '1620 Read Sense Switches '131620



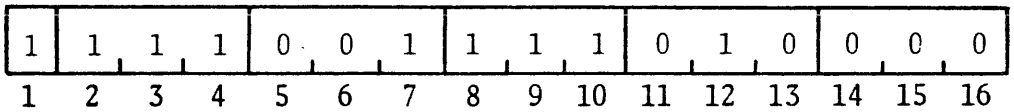
Read the contents of the control panel switch register as sense switch into A, where a switch that is on is read as a 1. A switch is on as a sense switch if it is up. This instruction does not skip.

INA '1720 Read Data Switches '131720



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.

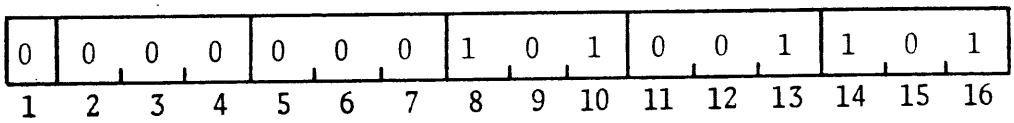
OTA '1720 Load Lights '171720



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.

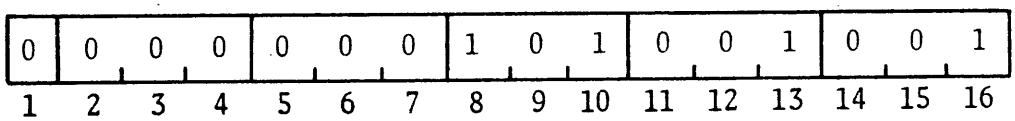
Besides the many peripheral devices connected to the I/O bus and controlled by I/O instructions, there is a basic serial interface that is built right into the processor and is controlled by special instructions. By means of this device, the program can control the transmission of serial data on four 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.

OSI Output Serial Interface '000515



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.

ISI Input Serial Interface '000511



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.

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 facilitiates concurrent operation of the main program and a number of peripheral devices. The interrupt system also allows conditions internal to the processor (traps) to interrupt the program, 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. 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.

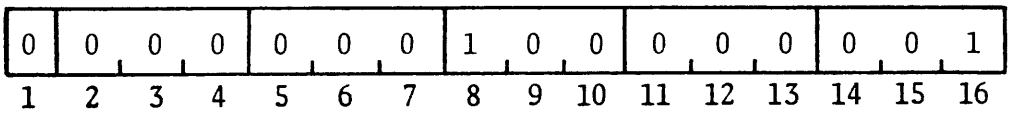
Since the system uses a location unique to each device, there is no need for testing, and the service routine acts only for the interrupting device (it should of course save keys and registers as usual). There is also a built-in priority determined by bus position, so even if the routine allows interrupts, no device higher on the bus can do so (in other words, all devices in higher-numbered slots are of lessor 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 reenabled.

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

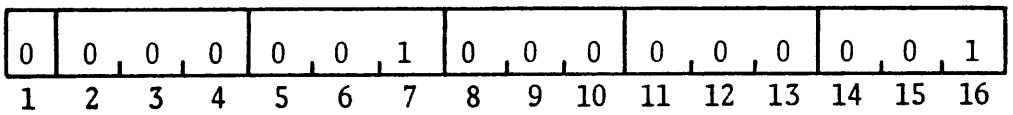
The instructions that control the interrupt system are all of the type with a full word op code, but associated with the system are two I/O instructions that deal with the mask used for setting up the Interrupt Enable flags in certain devices. When power is turned on or the computer is cleared from the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

ENB Enable Interrupt '000401



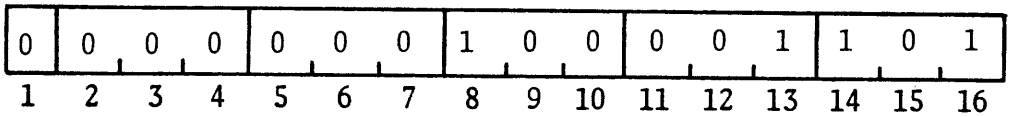
Enable the external interrupt system so the processor will respond to interrupt requests over the I/O bus. This instruction becomes effective following execution of the next sequential instruction.

INH Inhibit Interrupts '001001



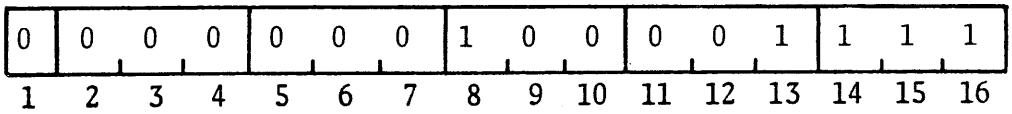
Inhibit the external interrupt system so the processor will not respond to interrupt requests over the I/O bus. This instruction takes effect immediately.

ESIM Enter Standard Interrupt Mode '000415



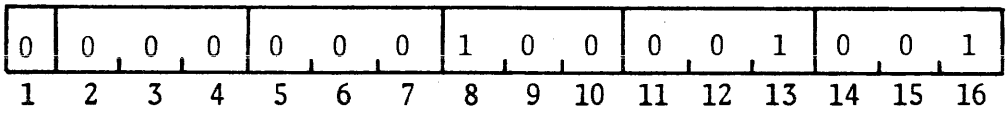
Enter standard interrupt mode so that all interrupts are made through location '63.

EVIM Enter Vectored Interrupt Mode '000417



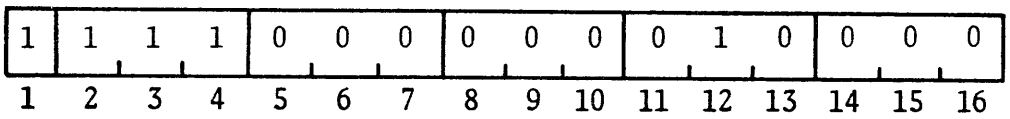
Enter vectored interrupt mode so that for interrupt purposes the priority of a device is determined by its position on the I/O bus (with lower devices having higher priority) and each interrupt is made through the location specified by the sole interrupting device.

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

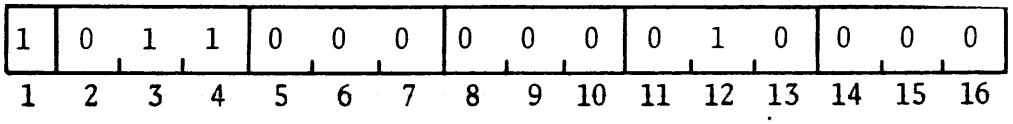


Set up the Interrupt Enable flags in the devices according to the mask in A (a 1 in a mask bit sets the flag in the device corresponding to that bit; a 0 resets it). Note that this instruction is equivalent to OTA'0020; and it never skips.

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:

1. 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.

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

3. The JST and ENB instructions delay external interrupt servicing for one full instruction cycle. So do the ILL and UII internal interrupts.

4. The principal 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.

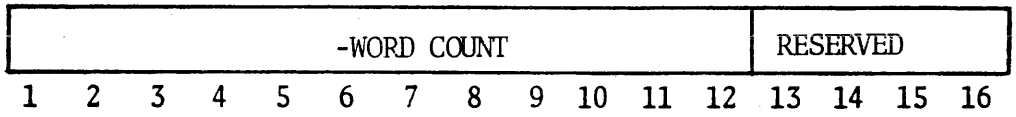
5. Before returning to the interrupted program, the routine should restore the keys and the register file, and in vectored mode it must give a CAI.

DIRECT MEMORY ACCESS

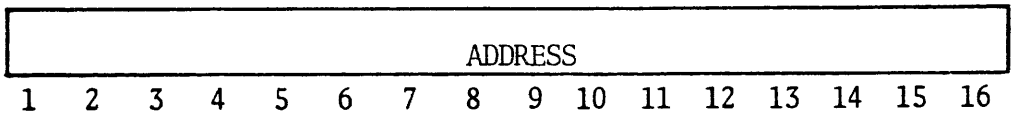
Handling data transfers between external devices and memory under programmed I/O control requires the execution of several instructions for each word transferred. To allow greater transfer rates, the processor contains eight direct memory channels through which devices, at their own request, can gain 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.

FIRST LOCATION



SECOND LOCATION



*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.

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 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 can be handled at the rate of one very 1.2 μs; this allows a maximum of 833,333 words per second, but at this rate all other processing activity is suspended.

SECTION 6

PERFORMANCE OPTIONS

This section describes three features that are standard on the Prime 300 and optional on the Prime 200 and 100:

Double Precision Integer Arithmetic and Multiply/Divide

Microverification

Microverification is not available for the Prime 100, which lacks the machine check function.

These features are implemented by extensions to the standard microcode on the CPU board; no mechanical extensions are required.

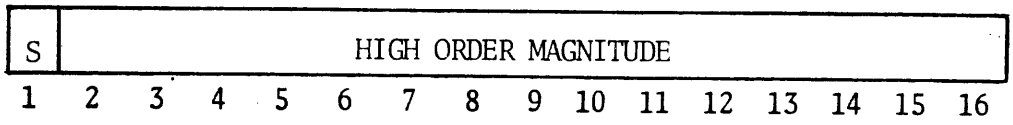
Double Precision Load, Store, Add and Subtract

When the processor is in double precision mode, the instructions that ordinarily load, store, add and subtract single-word numbers, instead operate on double-word numbers. The op codes for these memory reference instructions are the same as their single precision counterparts, but the assembler recognizes unique mnemonics for clarity of documentation.

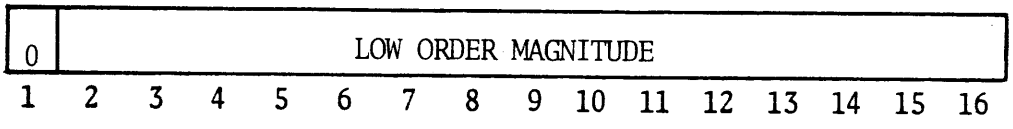
Depending on the operation, the double precision operands and results are held in the A and B registers or memory locations EA and EA+1.

The format is:

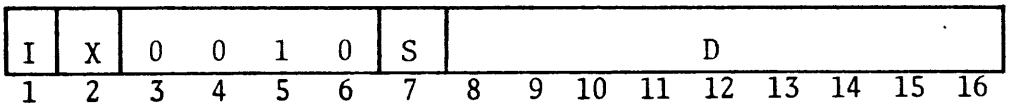
A or (EA)



B or (EA+1)

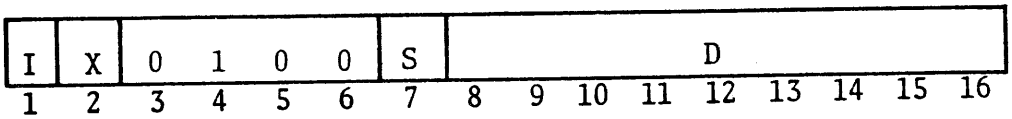


DLD Double Load '02



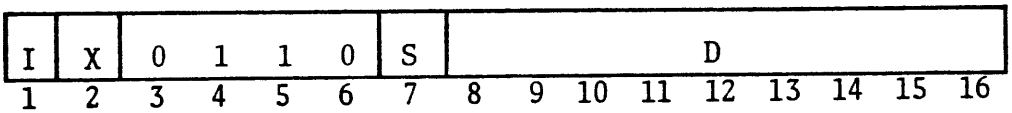
Load the contents of location EA into A and location EA+1 into B. The contents of memory are unaffected, the original contents of A and B are lost.

DST Double Store '04



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.

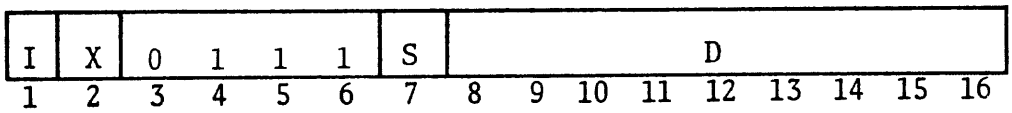
DAD Double Add '06



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 $>2^{30}$ 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



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 $>2^{30}$ 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}.

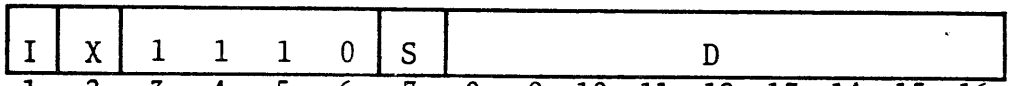
Although bit 1 of the low order part of a double precision number should be 0, this instruction does produce a correct result if the sign bits of the low order parts are both 1. However, the result is invalid if the low order sign bits are not the same.

To negate a double length number, simply subtract it from zero.

Multiply/Divide (Fixed-Point)

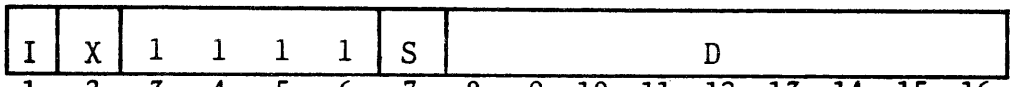
Prime computers have two basic instructions for performing multiplication and division of fixed point numbers. As previously mentioned, multiplication produces a double length product and division uses a double length dividend.

MPY Multiply '16



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

DIV Divide '17



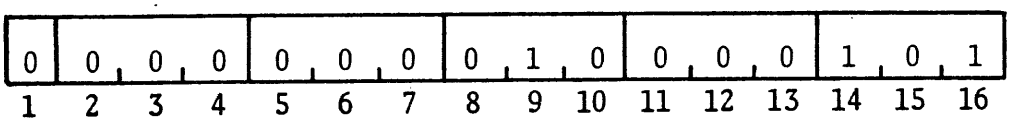
If the absolute value of the number in location EA is less than the absolute value of the number in A (taking both numbers as representing the same order of magnitude), set C; otherwise clear it. Then divide the double length contents of A and B by the contents of location EA, calculating a quotient 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 unchanged.

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.

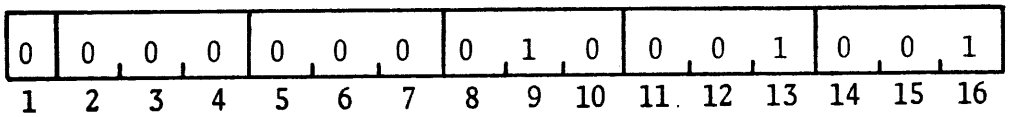
As given above, the instructions are somewhat cumbersome for working entirely with single-length integers. In a multiplication of small integers, the significant bits of the result are all in B, whereas the sign is in A. Similarly, it would be convenient to be able say to divide 15 by 3 and get an answer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

PIM Position following Integer Multiply '000205



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



Move the contents of B bits 2-16 into A bits 2-16. 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 oerder 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.

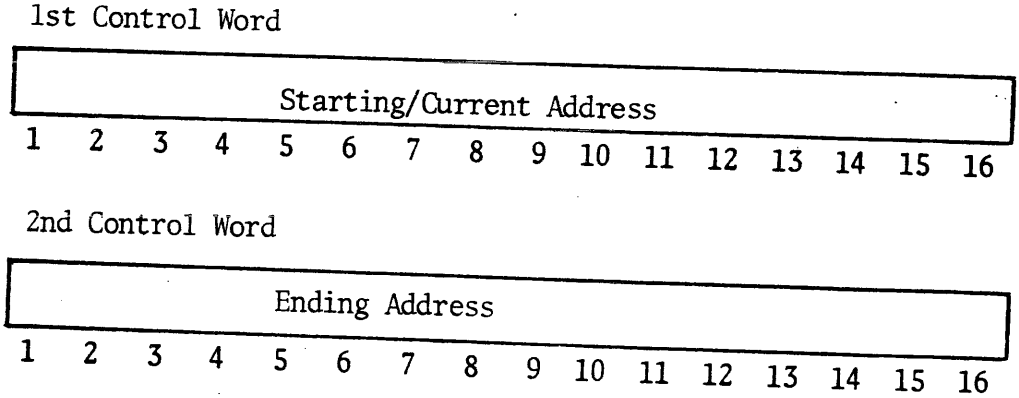
The PID allows the programmer to use a single length dividend and guarantees the division to be meaningful (producing the integral part of the quotient) except in the obvious case of a zero divisor. Effectively, the PID multiplies the given dividend by 2^{-15} so that the divisor is bound (unless it 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.

DIRECT MEMORY CHANNEL, DIRECT MEMORY TRANSFER (DMC, DMT)

The DMC and DMT modes of input/output operation extend the speed and flexibility of the standard DMA system available in all Prime computers. DMC is used to extend the number of direct memory channels (up to 2000) and the maximum block size handled by each channel (up to 64K words). DMT increases the maximum direct memory data rate to one million words per second. In contrast to programmed I/O, which requires the execution of several instructions for each word transferred, direct memory transfers reduce the number of instructions needed for I/O control, allow multiple high speed transfers to be handled concurrently and permit processing to be overlapped with I/O operations.

DMC Operation

DMC transfers are controlled in much the same way as DMA transfers; the program specifies a particular channel for an I/O device via an OTA to the device's controller and sets up a pair of control words to establish at what memory location the transfer will begin and how many words will be transferred. Unlike the DMA system, which uses preassigned pairs of high-speed registers to control the DMA channels, DMC transfers are controlled by pairs of adjacent locations in main memory. This permits up to 2000 DMC channels to be specified using memory locations between 64_8 to 7776_8. The first word of a control word pair contains the starting address for the transfer, and the second word contains the ending address as illustrated below.



Data blocks of up to 64K words can be transferred at input and output rates of up to 271,739 and 268,817 words per second. At the maximum input and output rates, processing is suspended, while at lower rates processing and I/O transfers are overlapped.

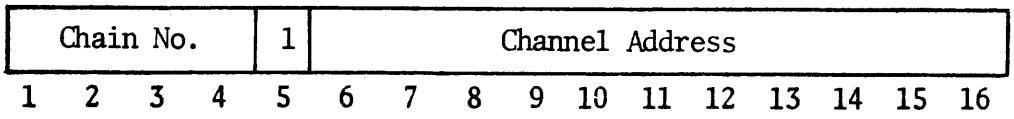
When a device requires DMC service, it requests access to memory via its specified channel. The DMC microcode automatically synchronizes with the instruction currently being executed and causes the processor to pause either at some point within the instruction or upon its completion. If several devices request servicing simultaneously, the order in which the requests are acknowledged is determined by the

priority relationship among the device controllers and the central processor. In general, the controller closest to the processor in the chassis (i.e., mounted in the lowest numbered slot) is given highest priority, while the controller in the highest slot position is assigned the lowest priority.

To service a channel request, the processor accesses the location specified by the first channel control word (starting or current address) and either reads or writes a word as specified by the device controller. The current address is incremented by one after each channel request is serviced until the current address is equal to the ending address, signaling that the block transfer is complete.

Chaining

DMC channels may be chained together to facilitate scatter/gather data transfers. An OTA 14XX loads a device controller with the required DMC set up information from the A register as shown below. A one in bit 5 specifies a DMC transfer (a zero specifies DMA).



The chain number specifies how many DMC channels in addition to one specified by the channel address portion of the word will be used for a data transfer. A chain number of zero causes the transfer to terminate after one end of range. A chain number greater than zero causes the controller to wait for that number of ends of range plus one before terminating the transfer. In this case, the channel address is automatically incremented by two after each end of range, thereby automatically switching control to the next higher DMC channel.

DMT Operation

Certain controllers are capable of providing the necessary memory addresses for direct memory transfers without using external control words stored in the processor or memory as with DMA or DMC transfers. This permits all channel control functions to be completely overlapped with processor and memory functions thereby increasing the computer's maximum input and output rates to 1, 086, 956 and 1, 041, 666 words per second respectively. When operating in the DMT mode, the controller automatically places the memory address of each word to be transferred directly on the I/O bus and terminates the transfer when the end of range has been reached. Because of its high speed and low control overhead, the DMT mode can multiplex data on a word-by-word basis.

Specification Summary*

	DMC	DMT
Maximum Transfer Rate (processing suspended)		
Input (words/sec.)	271,739	1,086,956
Output (words/sec.)	268,817	1,041,666
Interruption To Processing Per Word Transferred at Max.		
Input Rate at Max.	3.68 microsec	920 nanosec
Output Rate	3.72 microsec	960 nanosec
Interleaved Input	4.7 microsec	2 microsec
Interleaved Output	4.7 microsec	2 microsec

* Assumes microsec memory cycle time.

MICROVERIFICATION

Microverification Routines provide a powerful and flexible means of verifying data integrity and preventing the propagation of erroneous data within the system. The Microverification Routines consist of microprogrammed firmware sequences that can test the logic of the entire CPU, verify the reliability of the computer's error detection logic, and test the operation of all data registers, peripheral address and data bus lines, memory address and data bus lines, and the high-speed register file. In addition to these operational tests, the Microverification Routines can also force selected error conditions to occur and then verify that the CPU properly detects those conditions.

Operation

Since the Microverification Routines are implemented in the CPU's microcode, they are always resident within the system yet do not require memory space for storage. A microverification sequence is initiated whenever the system is cleared from the control panel (Master Clear), a machine check flag is set (hardware detection of a CPU error), or a VIRY (Verify) instruction is executed. For greater operating flexibility, initiation of the sequence following a machine check can be enabled or disabled under program control. (See EMCM and LMCM instructions in Section 4.) This is an important feature since if microverification is enabled for machine checks, the detection of a processor error automatically suspends normal processing for as long as the error condition exists. In certain situations, the user may wish to continue processing to predetermined check points and at such points initiate microverification under program control. When microverification is enabled for machine checks and a transient error is detected, the machine will automatically resume normal operation when proper operational status has been verified.

The result of a pass through the microverification routines depends on CPU status and the method of entering the routines. The various alternatives are summarized in Figure 5-1.

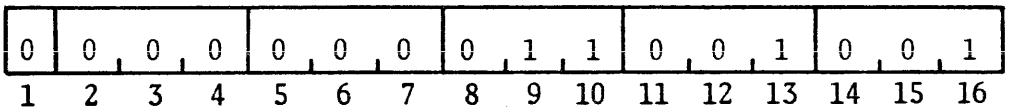
Transient Failure

If the entire routine runs (verification routine did not find an error), the failure may have been a transient one, therefore the micro-routine clears the keys and register file, and issues a programmed interrupt through location '70. This returns control to the system program (which can provide for recovery and continued operation).

Solid Failure

The micro-processor will recycle through the micro-verification routines as long as the failure exists (indefinitely). The number of the failing test is displayed in the address lights. Refer to Figure 8-4.

VIRY Verify '000311



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.

SECTION 7

PRIME 300 FEATURES

These CPU functions are standard in the Prime 300 and are not implemented in the 100 or 200. However, most of the instructions of the extended set can be executed by Unimplemented Instruction Interrupt (UII) software in the smaller processors. (Virtual memory instructions and the XEC instruction are not emulated by UII software.)

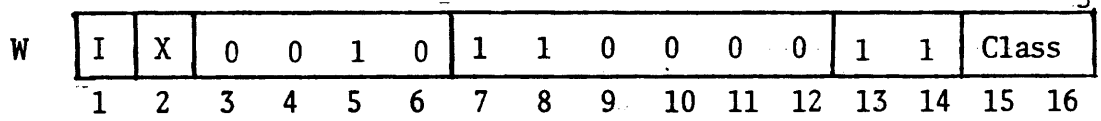
PRIME 300 EXTENDED INSTRUCTIONS

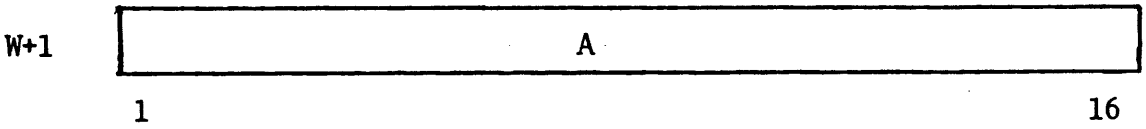
This group of instructions is hardware-implemented only in the Prime 300. The op-codes are obtained by using the extended instruction format. (See Appendix E.) However, all instructions of this group (except XEC) can be implemented on the Prime 100 or 200 through the UII software library. (See Section 2.)

Extended Jump Instructions

There are nine jump instructions in the extended, two-word instruction set. Six of them are conditional on whether the contents of the A register are equal to zero, greater than zero, etc. Other jump instructions combine a jump with incrementing, decrementing and storing the index register.

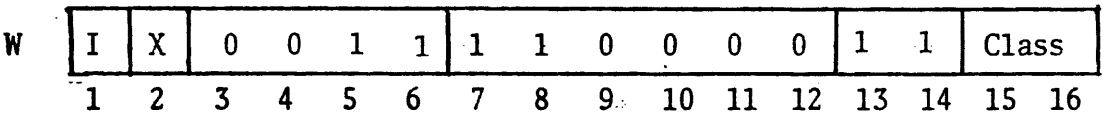
JEQ Jump If Equal to Zero '02 3

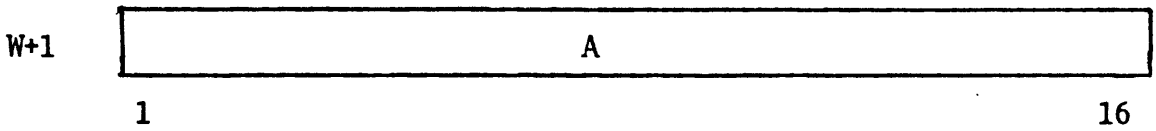




If the contents of the A register are equal to zero, then load EA into P, take the next instruction from location EA and continue sequential operation.

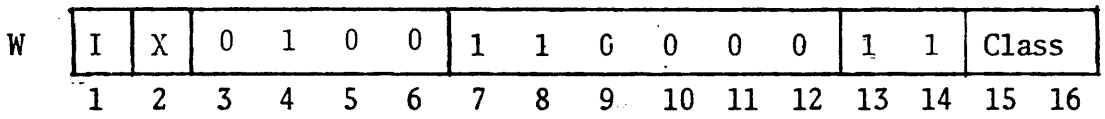
JNE Jump If Not Equal to Zero '03 3

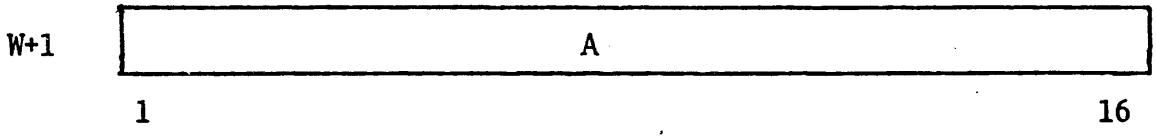




If the contents of the A register are not equal to zero, then load EA into P, take the next instruction from location EA and continue sequential operation.

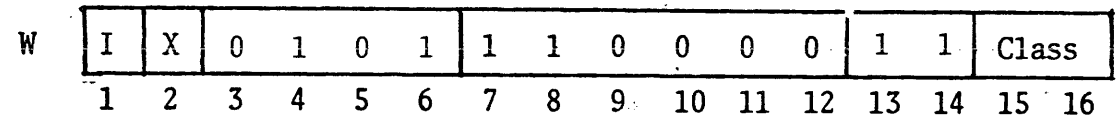
JLE Jump If Less Than or Equal to Zero '04 3

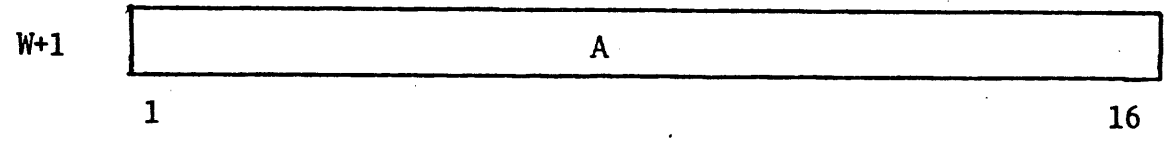




If the contents of the A register are less than or equal to zero, then load EA into P, take the next instruction from location EA and continue sequential operation.

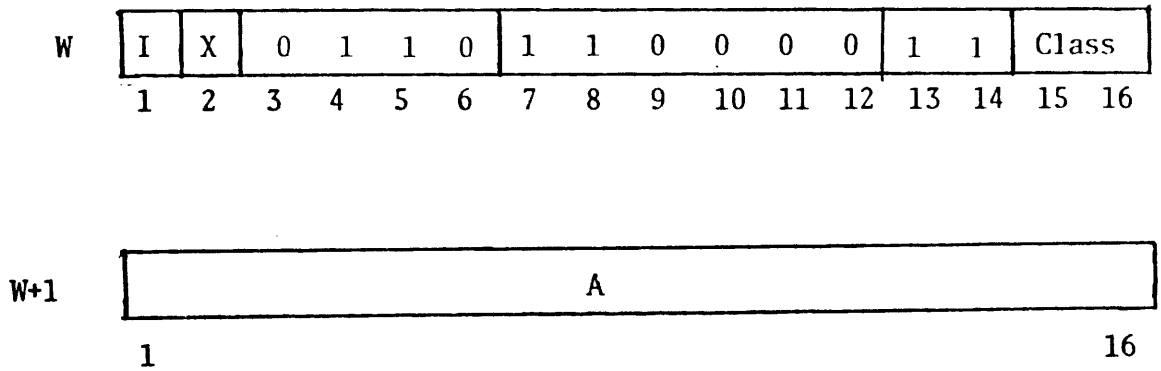
JGT Jump If Greater Than Zero '05 3





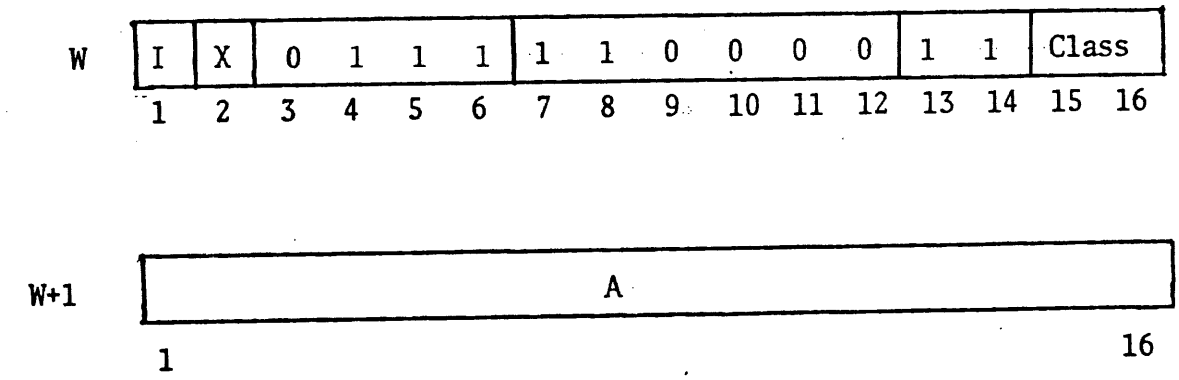
If the contents of the A register are greater than zero, then load EA into P, take the next instruction from location EA and continue sequential operation.

JLT Jump If Less Than Zero '06
3



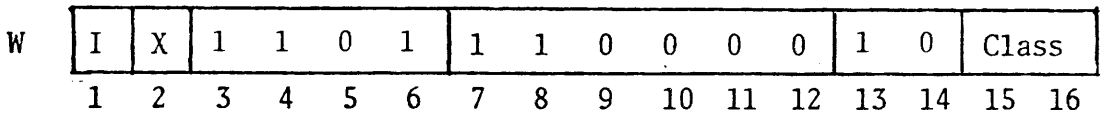
If the contents of the A register are less than zero, then load EA into P, take the next instruction from location EA and continue sequential operation.

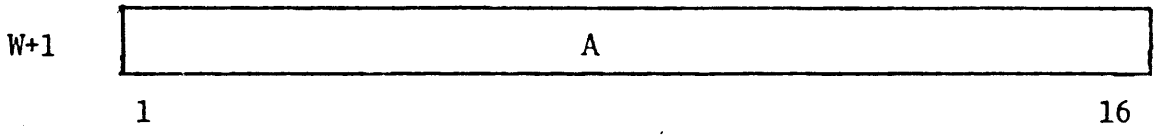
JGE Jump If Greater Than or Equal to Zero '07
3



If the contents of the A register are greater than or equal to zero, then load EA into P, take the next instruction from location EA and continue sequential operation.

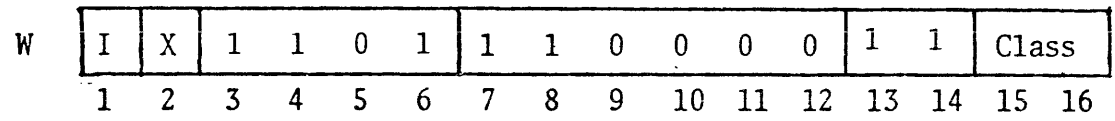
JDX Jump and Decrement Index '15 2

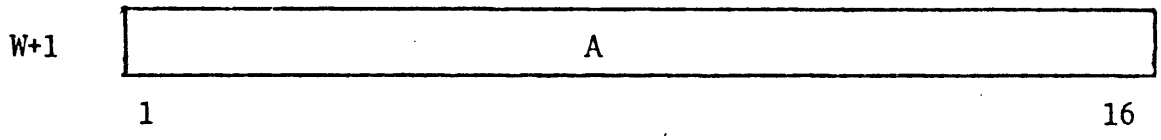




Decrement the contents of the index register by one; then, if the contents of X are not equal to zero, load the contents of EA into the program counter and execute EA as the next instruction. Otherwise, increment the address currently in P and continue.

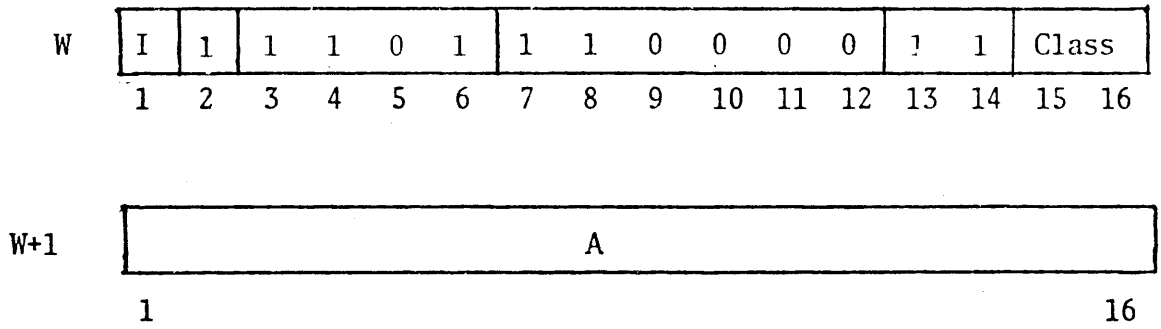
JIX Jump and Increment Index '15 3





Same as JDX but increment index register.

JSX Jump and Store Return in Index '35*
3



Increment the program counter by one and load into the index register. Load EA into P and execute EA as the next instruction in sequence.

* X Bit = 1

Procedure Stack Control

This group of instructions simplifies programming of pure procedures, recursive or reentrant subroutines, and dynamic storage allocation. ENTR alters the stack pointer (S Register) to create an n-word stack frame, and links the new frame with the previous one. CREP saves the program counter in the current stack frame and transfers control to a subroutine. RTN undoes the work of both CREP and ENTR by deleting the current frame and restoring the saved program counter value for the calling program.

ENTR — Enter Recursive Procedure Stack — '01 3

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

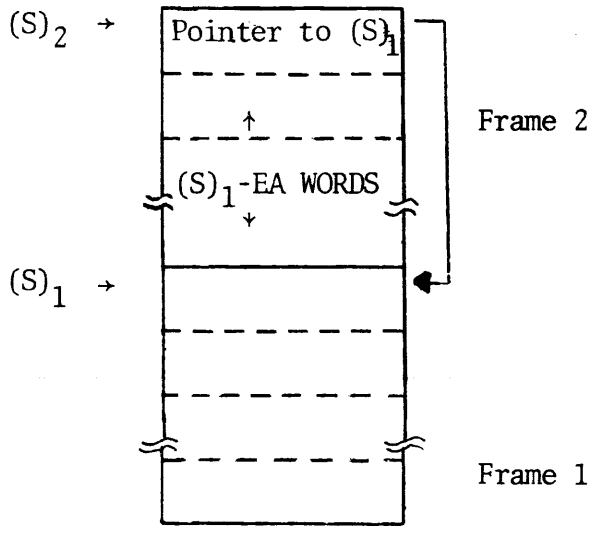
W+1	STACK FRAME SIZE	
	1	16

Alter the stack pointer by subtracting the value of EA and store the previcus value of S in the new location.

$$(S)_1 \rightarrow [\ (S)_1\text{-EA}\ ]$$

$$(S)_1\text{-EA} \rightarrow (S)_2$$

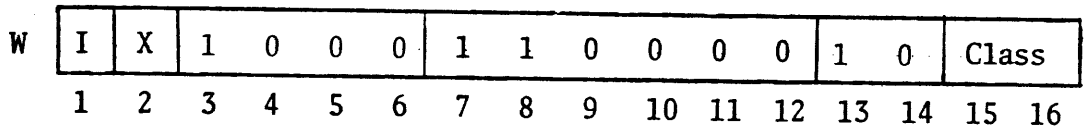
The ENTR instruction allocates a block of memory as a stack frame containing EA locations:

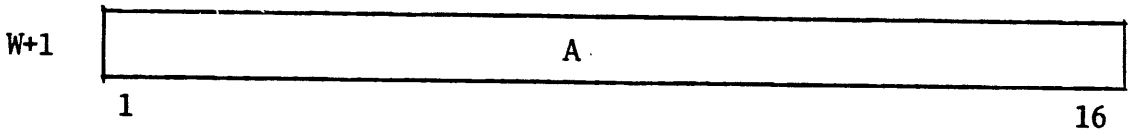


The frame is created by subtracting EA locations from the stack pointer contents, $(S)_1$, to form $(S)_2$, and then storing $(S)_1$ at that address. Thus, the first word of the frame points to the previous frame.

Stack frames created by recursive or reentrant procedures are assumed to contain n+2 words, where n is the number of locations required for variable or parameter storage during a recursion of the subroutine. The other two words are reserved for the frame linkword (inserted by ENTR) and a return address (inserted by CREP).

CREP Call Recursive Entry Procedure '10 2



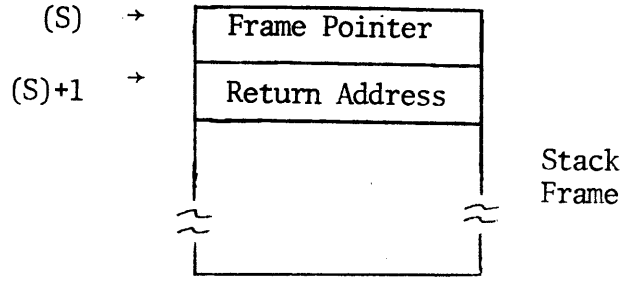


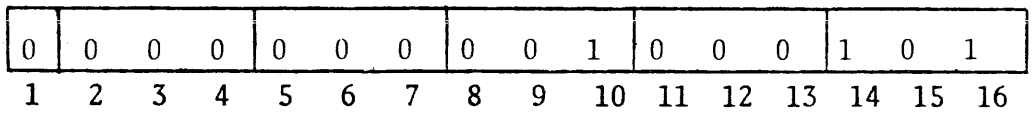
Increment P and load P+1 into the location following the one specified by the current stack pointer. Load EA into P and execute (EA) as the next instruction:

$$(P)+1 \rightarrow [\ (S)\ +1\ ]$$

$$EA \rightarrow (P)$$

The CREP instruction performs subroutine linkage for recursive or reentrant procedures. CREP stores the return address in the second word of a stack frame created by the ENTR instruction, rather than in the destination address as in a JST:





Fetch the return address from word 2 of the current stack frame and load the result in the program counter; then transfer word 1 (the pointer to the preceding stack frame) to the S register.

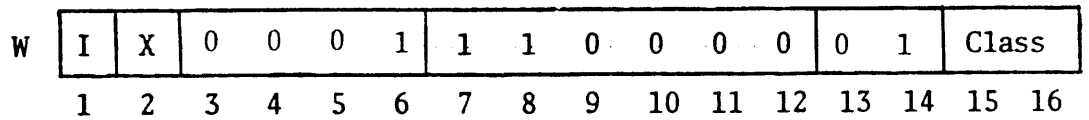
$$[(S)+1] \rightarrow P$$

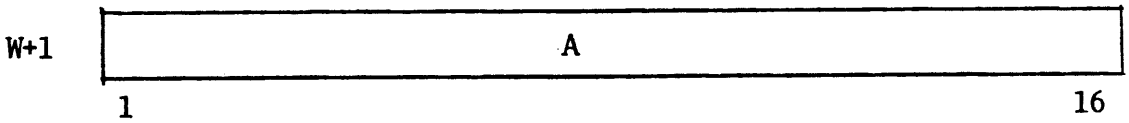
$$[(S)] \rightarrow (S)$$

If the return address is 0, (S) is unchanged and the microcode causes a PSU (Procedure Stack Underflow) interrupt through location '75.

OTHER EXTENDED INSTRUCTIONS

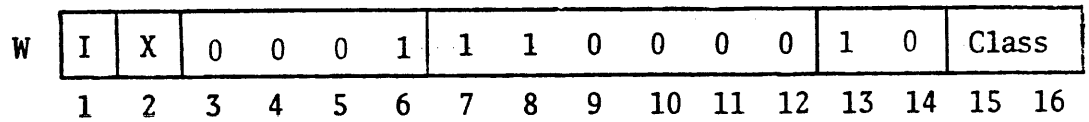
EAA Effective Address to A Register '01 1

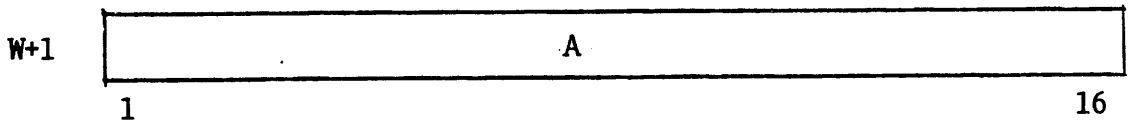




Calculate the effective address and load it into register A. The contents of EA are unaffected and the original contents of A are overwritten and lost.

XEC Execute Effective Address Contents as Next Instruction '01 2





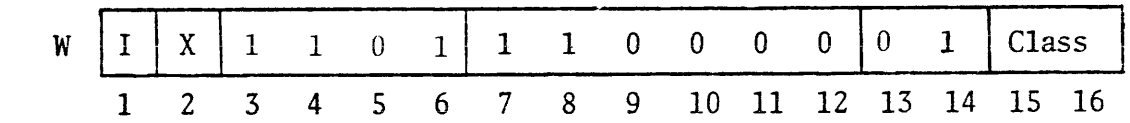
Load the contents of EA into register A and execute (EA) as the next instruction in sequence. This instruction permits portions of program previously coded to be reiterated with one instruction. There are three restrictions:

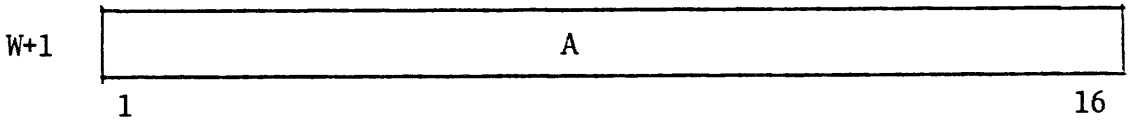
1. Double or multiple word instructions use the location(s) following the XEC for expansion beyond a single word.

2. XEC's using the stack in Push/Pop mode have unpredictable results if the instruction to be executed is unimplemented and a double-word or page fault occurs.

3. Memory reference instructions are "relativized" on the XEC program counter leading to results not anticipated by the assembler or loader.

FLX Load Double Word Index '15 1





Double the contents of the effective address and load the result into the index register X. This instruction facilitates indexing sequences that involve double-word memory reference operations. It works directly for two-word indexing, e.g., double precision integer or floating point.

VIRTUAL MEMORY

See Virtual Memory Data Sheet

The Virtual Memory feature (VM) greatly expands the processing and storage resources of your PRIME small computer.

VM adds the following three basic capabilities:

1. Expansion of memory addressing to 262,144 words of High Speed Memory (HSM) - i.e., real or physical memory.

2. Hardware protection of software integrity.

   *specific areas within a task (user-level) can be protected against being altered by the task itself

   *tasks can be protected against access and alterations from other tasks

   *executive routines (base and supervisory level) can be protected against alteration by (user-level) tasks

3. Automatic swapping of program segments (pages) between HSM and disk.

VM's capabilities facilitate:

1. Multi-user time shared disk operating systems.

2. Multi-tasking real time operating systems (HSM only).

3. Foreground/Background systems with real time multi-user or multi-tasking in the (protected) foreground and batch operations in the background.

4. Execution of single programs larger than 64K with or without a disk.

VM is implemented by the following features:

1. Paging

   Paging is a technique of segmenting memory into a fixed length of 512 words or 'pages', intercepting memory access, and translating the access from a 'virtual' address to a 'real' or physical address. This translation expands the normal 16 bit address field (2^{16} = 65,536 words) to 18 bits (2^{18} = 262,144 words). Each translation references a 'page map' that contains the 'real' address for each of the 'virtual' addresses. When paging operations are enabled, the processor is said to be in the 'Paging Mode'.

2. Page-turning

   A disk based system can effectively expand the 'size' of HSM to be as big as the storage size of the disk. This is done by swapping program segments in and out of HSM or page-turning. This is a practical technique for many applications since a program usually executes one portion at a time. To make page-turning efficient and practical, the executive program is automatically notified when a task tried to access a page not in HSM. The information of which pages are in or out of HSM is stored in the page map. This condition is called a 'page fault'. The executive is also told what page the program tried to access and then brings the required page into HSM. This feature means that user-level programs can be written without concern for paging; i.e., paging is the executive's responsibility and is transparent to the user-level programs.

3. Write Protection

The page map can also specify which pages can be altered and those that cannot. Thus, the map is consulted for each write operation. If a task tries to write into a protected page, the executive is automatically notified of the attempted violation.

4. Restricted Execution

User-level programs operate in the restricted execution mode (RXM). In the RXM mode, the executive is automatically notified when the user-level program attempts to execute any one of a class of I/O, interrupt and control instructions. This insures a delineation of responsibilities between the user-level programs and the executive program that controls all I/O, interrupts, and processor modes. In this way, 'stand alone' user-level programs needn't be re-written to run under a time-sharing executive, and they can run without danger of alteration to the executive or other tasks.

5. Hierarchy of Processing States

Three district processing states can be defined as a direct result of the Paging Mode and the Restricted Execution Mode. These are:

User-level: usually the 'application' software operating in both the Paging and Restricted Execution Mode.

Supervisory-level: A portion of the executive program that needn't be resident in HSM and operates in the Paging and non-Restricted Execution Modes.

Base-level: A fundamental portion of the executive program that must reside in HSM and operates in the non-Paging and non-Restricted Execution Modes.

TECHNICAL DISCUSSION

Paging

A page is a 512 word contiguous address space whose starting address is a multiple of 512. Normally on a PRIME 200 or 300 the maximum address space or segment available to a program is $2^{16} = 65,536$ words because of the inherent 16 bit address field. Consider the format of a 16 bit effective address associated with a program segment.

VIRTUAL ADDRESS

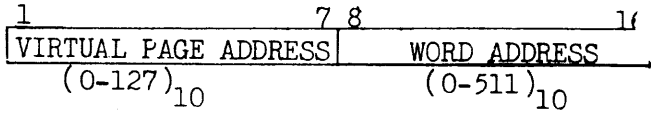


Figure 1

Page Map

In the page mode, the Virtual Page Address (VPA) points to an entry in the page map. That entry contains the real page address and indicates if the page is in HSM and if it is 'write-protected'. The contents of the map are created by the base-level executive software. Each user-level program has a map that normally consists of 128 entries - one for each virtual page address. The format for each map entry is:

to the program. That program can access only those pages in its map.

To reallocate the computers resources to another program, the executive need only change the PMAR to a new map address. Time-sharing and multi-tasking is facilitated in this manner. The reallocation process can be initiated by an external interrupt, specfically the Real Time Clock, or by a call to the executive from a user-level program.

Within a user-level program, pages can be protected against being altered. This is done by the executive setting the write protect bit in each of the protected pages map entries (Figure 2). When a protected page is accessed by an instruction that would alter its contents, a page write violation interrupt is generated.

A third level of protection is provided by restricting selected programs from executing instructions that would alter the processors control state. These instructions include all I/O, interrupt, and mode control. The executive program would normally enable the Restricted Execution Mode (RXM) as part of its transfer to a user-level program. This mode is exited by an external interrupt or when a restricted instruction is attempted. The later case causes a RXM interrupt and, as with an external interrupt, would normally branch to the base-level executive program.

Hierarchy of Processing States

The combination of page mode and restricted execution mode results in three fundamental processing states.

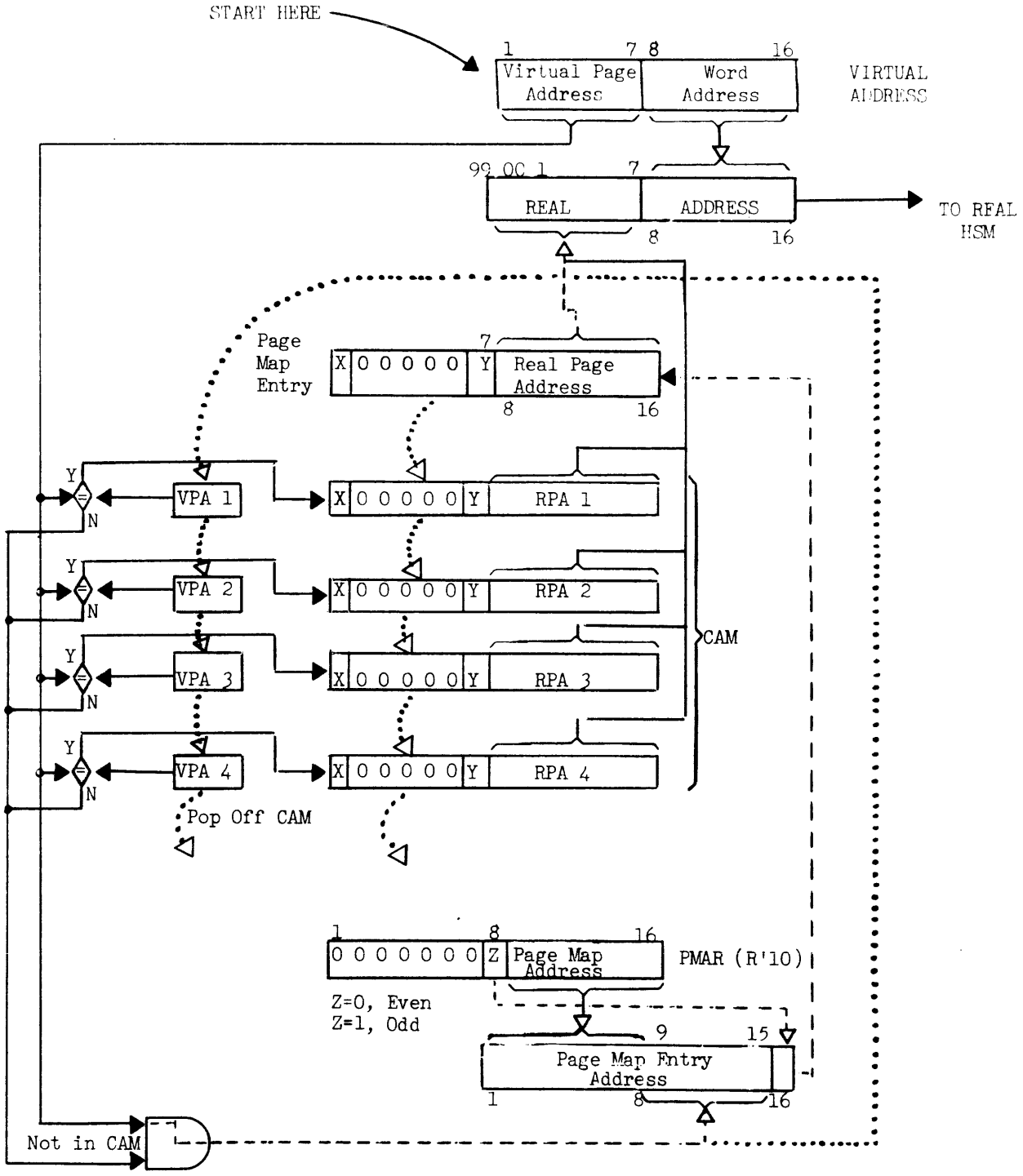
Processing State	Paging Mode	Restricted Execution Mode
User-level	ON	ON
Supervisor-level (executive)	ON	OFF
Base-level (executive)	OFF	OFF

The base-level program is always resident in HSM and is responsible for handling

1. interrupts
    a. I/O devices
    b. Real Time Clock
    c. page-fault
    d. restricted execution fault
    e. disk transfers
2. bookkeeping
    a. operating state
    b. HSM allocation

The supervisor level is part of the operating systems executive and is a continuation of the base-level executive. The supervisory level can be page-turned and therefore inherently slower to respond to stimuli than the base. It can be used for such functions as file management and internal operating system commands such as:

- Attach a user file space to a terminal
- Read batch commands from a specified file and execute
- Load a memory image file into HSM
- Save HSM on a user file



→ Points to
→ Transferred to

X = Page is in HSM; 1=Yes, 0=No

Y = Page is write protected; 1=Yes, 0=No

Phase 1 ——— First check CAM*
Phase 2 - - - If not in CAM
Phase 3 ..... Update CAM*

*If reference was in CAM, that entry is pushed to the top of the CAM

CAM = Content Associative Memory registers
VPA = Virtual Page Address
RPA = Real Page Address
HSM = High Speed Memory
PMAR = Page Map Address Register

Figure 7-4

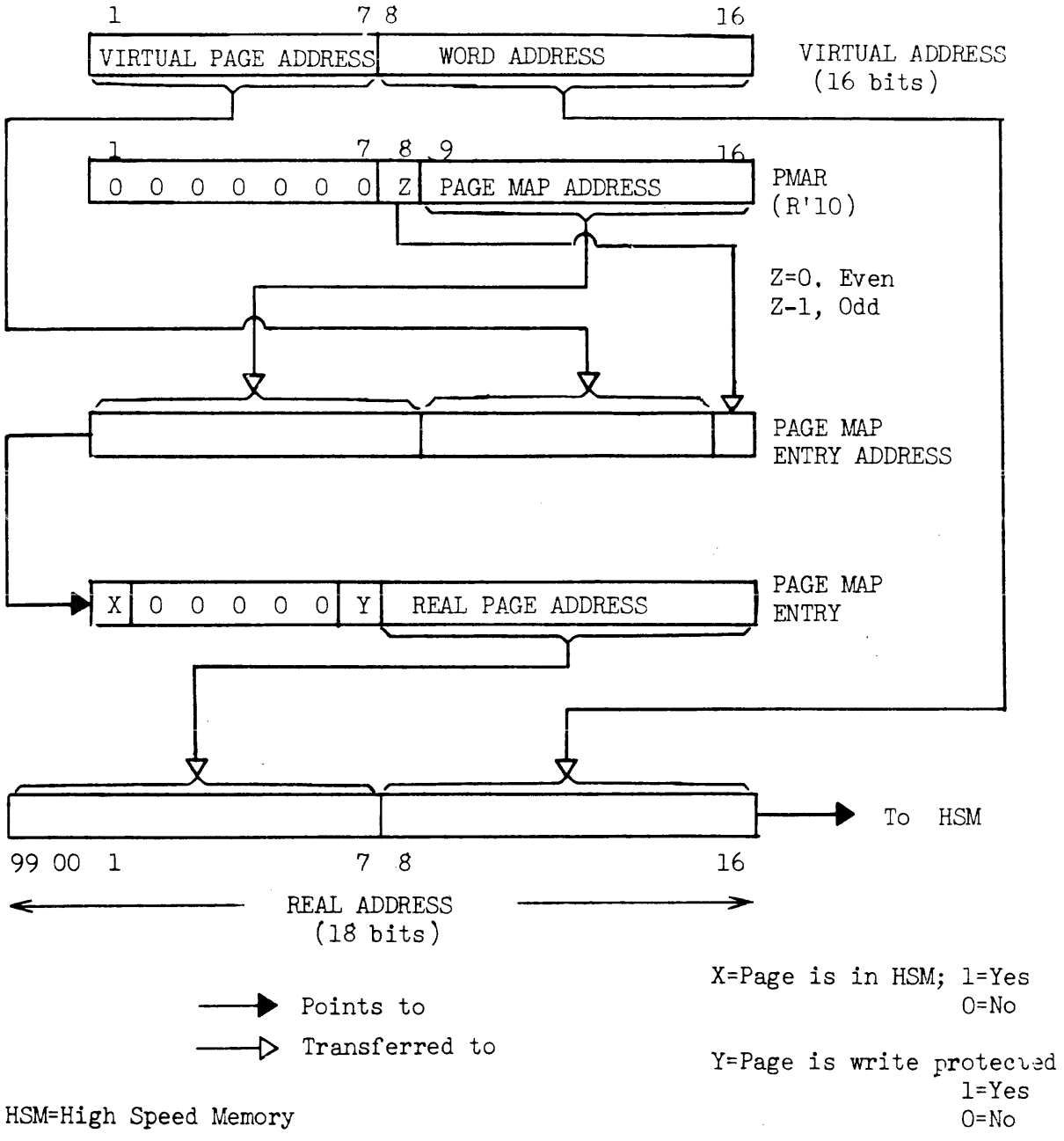
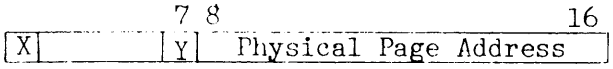

1
7 8
16
VIRTUAL PAGE ADDRESS
WORD ADDRESS
VIRTUAL ADDRESS
(16 bits)
1
7 8 9
16
0 0 0 0 0 0 0
Z
PAGE MAP ADDRESS
PMAR
(R'10)
Z=0, Even
Z-1, Odd
PAGE MAP
ENTRY ADDRESS
X
0 0 0 0 0
Y
REAL PAGE ADDRESS
PAGE MAP
ENTRY
To HSM
99 00 1
7 8
16
REAL ADDRESS
(18 bits)
Points to
Transferred to
X=Page is in HSM; 1=Yes
0=No
Y=Page is write protected
1=Yes
0=No
HSM=High Speed Memory

Figure 7-3

1st word



2nd word

X: page in HSM; 1 = yes, 0 = no
Y: page is write protected; 1 = yes, 0 = no
2nd word is usually used by the supervisor for the pages disk address. It can also be used for a 2nd interleaved map.

Figure 2

A page map consists of 128 entries (2 words per entry). When a map is in HSM, it starts at a multiple of 256 or 256 n +1. To activate a user-level program in the page mode, its map must be in HSM and the map's starting address loaded into Register '10 called the Page Map Address Register (PMAR).

As illustrated in Figure 3, when the user-level program initiates a fetch to an effective address (Figure 1), the following sequence of events occurs:

The virtual page address is doubled and 'added' to the PMAR to create an address that points to the appropriate entry in the map. The physical page address (Figure 2) becomes the high order 9 bits of the physical HSM address; the word address (Figure 1) becomes the low order 9 bits. This is the full 18 bit physical HSM address.

CONTENT ASSOCIATIVE MEMORY REGISTERS (CAM)

The process described above requires an extra memory cycle for each 'virtual' memory reference. The VM feature has four Content Associative Memory (CAM) Registers that reduce the overhead to 80 ns per memory reference. The CAM registers contain a copy of the four last referenced page map entries (see Figure 4). The contents of these registers are inspected before the HSM map. If the CAM registers contain the required map entry, the overhead is 80 ns; if not, the HSM map is accessed and copied into the particular CAM register that has gone the longest time since it was last accessed. Most programs spend most of the time within a page and would usually find the map entry in CAM. PRIME has measured the performance of typical programs operating under its VM operating systems and found that only 3% to 4% of map references are not found in CAM.

Page-fault/Page-turning

The page map entries have a bit that indicates if the page is in HSM or on the disk. When a program accesses a new page and its map entry indicates it is not in HSM, a page-fault interrupt occurs and the virtual address causing the page-fault is loaded into register '12. The base-level executive program must respond to the interrupt, decide what physical space to use for the new page, load the page off the disk into that space, update the page map entry, and return control over to the user-level program. This procedure is referred to as page-turning. One technique used in page-turning is to use the second word of a page map entry to store the page's disk address.

Hardware Memory Protection

VM provides three levels of protection for maintaining program integrity. The first is inherent in the page mode operation. Each user-level program is associated with a page map. To activate that program, the executive must load the PMAR with the maps physical address and turn control over

VM Interrupts

Octal Vector Location	Description
'65	SuperVisor Call. Generated by the execution of the SVC instruction. The contents of the Program Counter is copied into the location addressed in '65. The Program Counter points to the location following the SVC instruction.
'64	Page-fault. This interrupts occurs when the page map (see Figure 2) indicates the page required by the executing instruction is not in memory. The Program Counter pointing to the faulted instruction is copied into the location addressed in '64. The address of the requested page is copied to location $'12_8$. The interrupt can occur only when the paging mode is enabled. When a page fault interrupt occurs, paging mode is disabled.
'73	Page Write Violation. This interrupt occurs when the page map (see Figure 2) indicates the page about to be written into is write protected. The Program Counter pointing to the violating instruction is copied into the location addressed in '73. The interrupt can occur only when paging mode is enabled. When a page write violation interrupt occurs, the paging mode is disabled.
'62	Restricted Execution Violation. This interrupt occurs when the following types of instructions try to execute:

I/O	INTERRUPTS	CONTROL
OCP	ENB	HLT
SKS	INH	EMCM
INA	ESIM	LMCM
OTA	EVIM	RMC/RMP
ISI	CAI	VIRT
OSI	SMK	EPMJ
	IMK	LPMJ
		EVMJ
		ERMJ

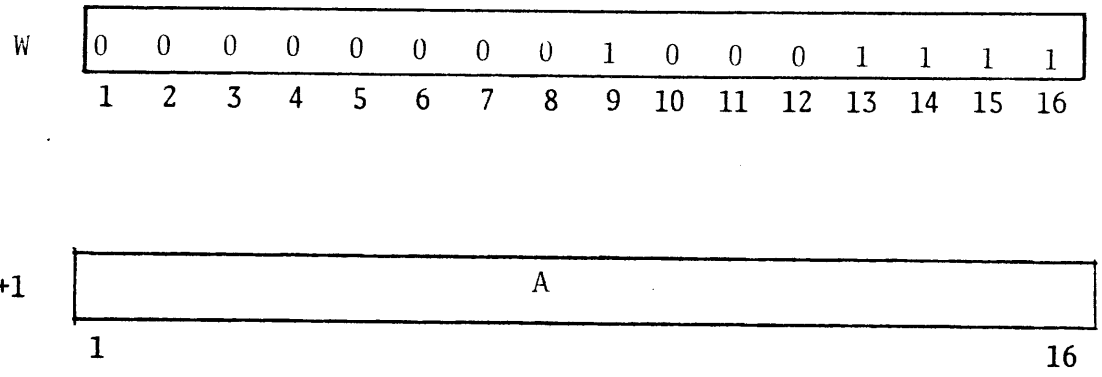
The Program Counter pointing to the violating instruction is copied into the location addressed in '62. The interrupt can occur only when restricted execution mode is enabled. When a restricted execution violation interrupt occurs, the restricted execution mode is disabled.

This page is intentionally left blank!

Virtual Memory Instructions

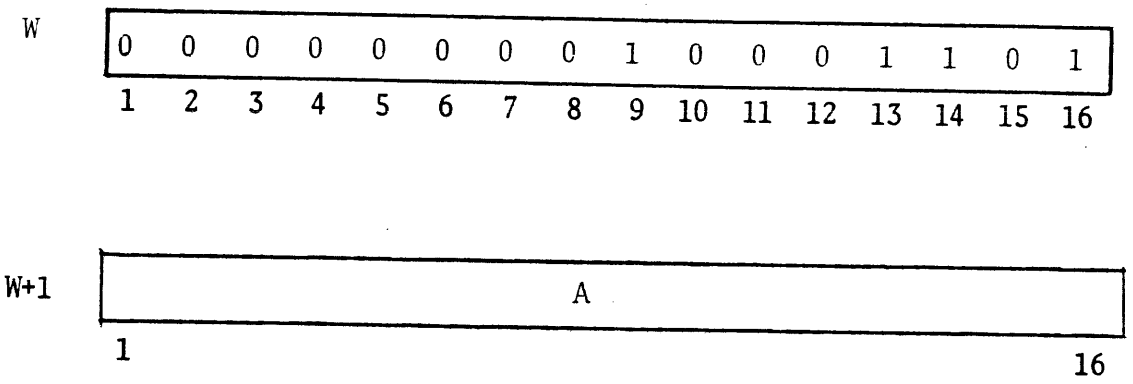
The following instructions control page, virtual, and restricted execution modes. They are not emulated by UII software on the Prime 100 or 200, which lack the virtual memory hardware, and so cause an illegal instruction trap.

EPMJ Enter Paging Mode and Jump '000217



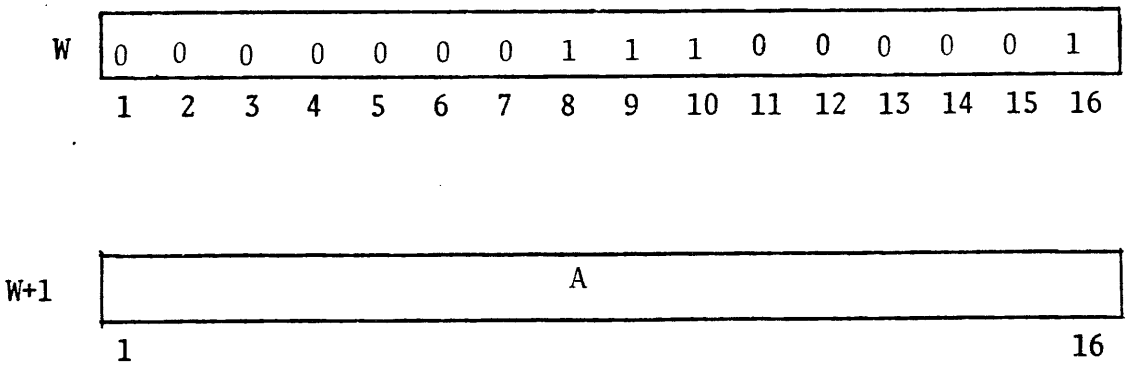
EPMJ is a two-word instruction. The first word is the OP Code '000217; the second word contains a 16-bit address pointing to the final effective address which is transferred to the Program Counter; the CAM registers are cleared, and the page mode is enabled.

LPMJ Leave Paging Mode and Jump '000215



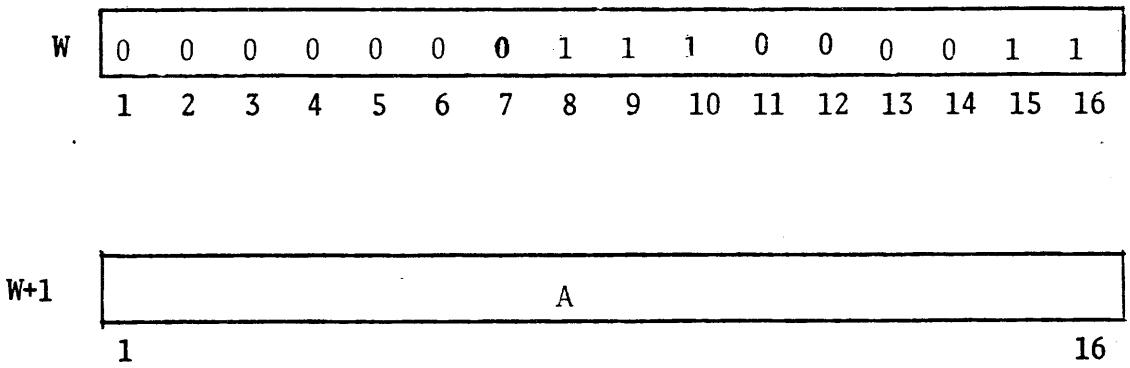
LPMJ is a two-word instruction. The first word is the OP Code '000215; the second word contains a 16 bit address pointing to the final effective address which is transferred to the Program Counter, and the paging mode is disabled.

ERMJ Enter Restricted Execution Mode and Jump '000701



ERMJ is a two-word instruction. The first is the OP Code '000701; the second word contains a 16 bit address pointing to the final effective address which is transferred to the Program Counter; restricted execution mode is enabled, and interrupts are enabled,

EVMJ Enter Virtual Mode and Jump '000703



EVMJ is a two-word instruction that has the effect of an EPMJ and ERMJ combined. The first word contains the OP Code '000703. The second word contains a 16 bit address pointing to the final effective address which is transferred to the Program Counter; the CAM registers cleared, paging mode is enabled, restricted execution mode is enabled, and interrupts are enabled.

SECTION 8

EXTENDED CONTROL STORE OPTIONS

This group of options is implemented by a mechanical extension to the microprocessor control store on the CPU board of a Prime 200 or 300. Floating point arithmetic is optional for the Prime 200 or 300, and can be emulated by Unimplemented Instruction Interrupt (UII) software on any Prime CPU. Writable control store is optional only on the 300 and is not emulated by UII.

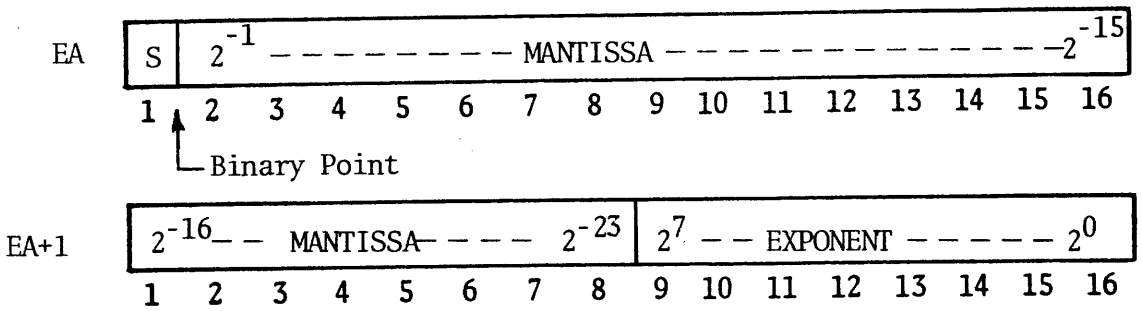
SINGLE AND DOUBLE PRECISION FLOATING POINT ARITHMETIC

Single and double precision floating point arithmetic operations are programmable by means of an optional 26-instruction repertroire on Prime 200 and 300 machines. There are 18 single-precision and 8 double-precision instructions, described below and summarized in Figure 8-1. Floating point instructions use extended addressing formats and so operate only in the relative addressing modes (E32R, E64R).

Floating point hardware is available as an option where execution time is a critical consideration. Alternatively, floating point arithmetic can be performed by the VIP software routines provided in the FORTRAN/ Math Library that accompanies all Prime computers. The formats for single and double precision floating point numbers are shown in Figure 8-1. Mantissa and exponent ranges are compared in Table 8-1.

Single Precision

Single-precision floating point numbers consist of a sign, a 23-bit mantissa and an eight-bit exponent, packed into two consecutive memory locations as follows:



The sign bit is 0 for positive, 1 for negative. The mantissa is a two's complement binary fraction with the binary point between the sign and the most significant mantissa bit. It provides about 6-1/2 decimal digits of resolution (±8,388,607). The eight-bit exponent uses excess 128 notation to represent a power of 2 from -128 to +127. (Approximately $10^{\pm 38}$).

Power of 2	Excess 128 Notation
2^{+127}	11 111 111
2^{0}	10 000 000
2^{-128}	00 000 000

INSTRUCTIONS

Single	Double	Description
FLD	DFLD	Load
FST	DFST	Store
FAD	DFAD	Add
FSB	DFSB	Subtract
FMP	DFMP	Multiply
FDV	DFDV	Divide
FCS	DFCS	Compare
FLX	FLX	Load Floating Index
FLOT	FLOT + CRB	Float A|B
INT	INT	Integer part of floating AC => A|B
FRAC	FRAC	Fractional part of floating AC => A|B
FCM	DFCM	Floating complement
FRN		Round up
FSZE	FSZE	
FSNZ	FSNZ	
FSMI	FSMI	Floating skips: work equally well
FSPC	FSPC	for both single and double precision.
FSLE	FSLE	
FSGT	FSGT	

FORMATS

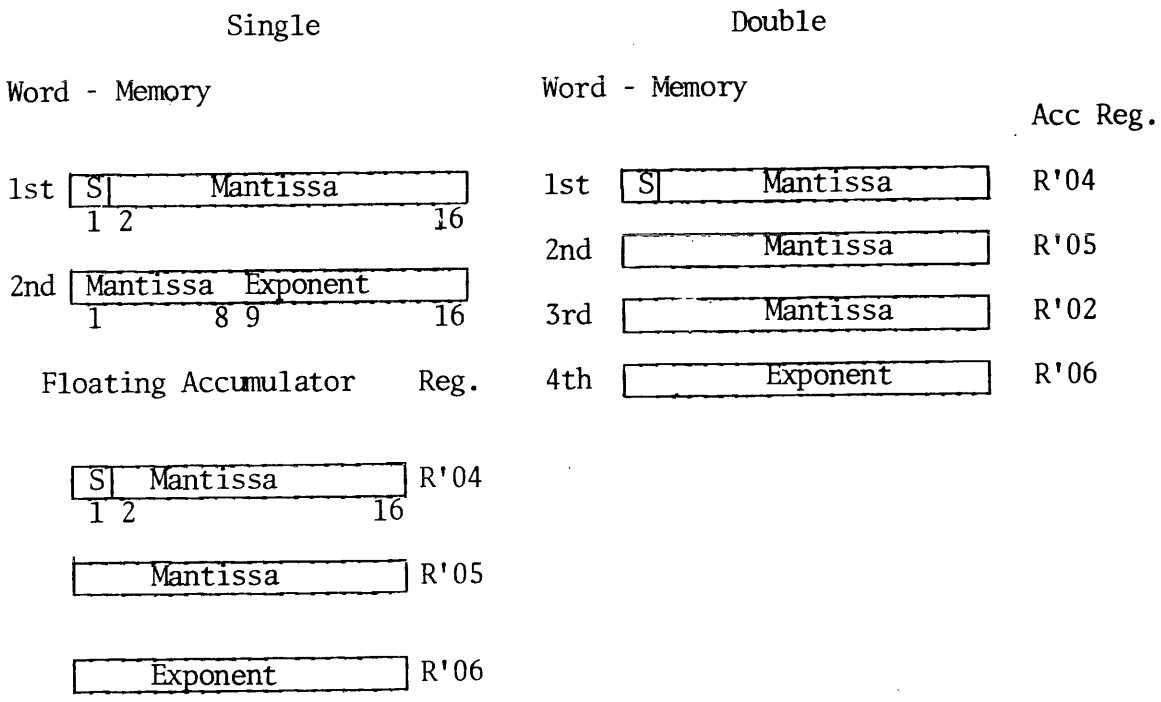


Figure 8-1. Floating Point Summary

Table 8-1. Floating Point Mantissa and Exponent Ranges

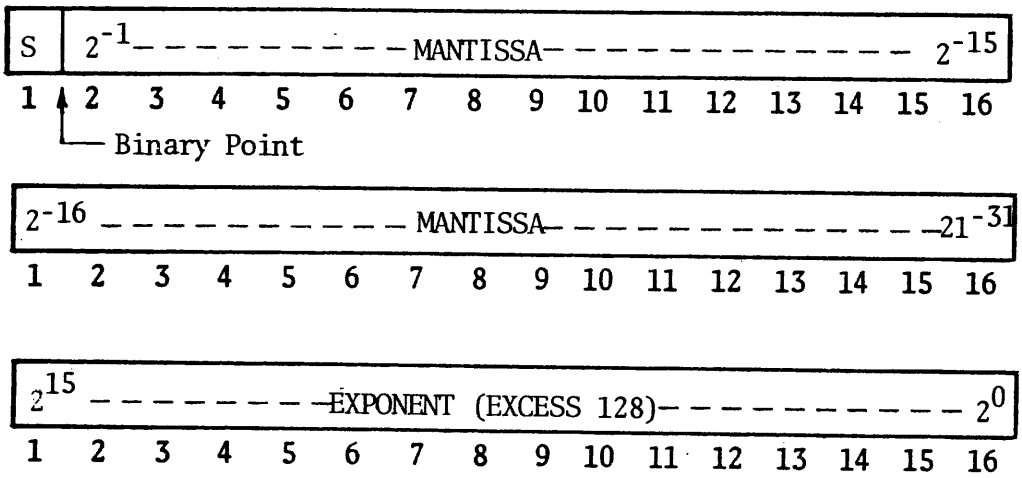
Field	SP (MEM)	SP (FAC)	DP (MEM and DFAC)
Mantissa			
Bits	23 + Sign	31 + Sign	47 + Sign
Precision	$\pm$ 8,388,607	$\pm$ 2,147,483,647	$\pm$ 140,737,488,355,327
Exponent			
Bits	8	16	16
Range	-128 to +127 ($10^{\pm 38}$)	-32896 to +32639 ($10^{+9823, -9902}$)	-32896 to +32639 ($10^{+9823, -9902}$)

Zero is represented by two words of all zeroes. Below are samples of single-precision values as they would be generated by Assembler DATA statements. (The E specifies a decimal exponent.)

```
                        (0001) *     FLOATING POINT EXAMPLES
                        (0002) *
                        (0003) *     SINGLE PRECISION
                        (0004) *
000000:  000000         (0005)       DATA  0E0
000001:  000000
000002:  040000         (0006)       DATA  .5E0
000003:  000200
000004:  040000         (0007)       DATA  1E0
000005:  000201
000006:  100000         (0008)       DATA  -.5E0
000007:  000177
000010:  100000         (0009)       DATA  -1E0
000011:  000200
000012:  040000         (0010)       DATA  128E0
000013:  000210
000014:  100000         (0011)       DATA  -128E0
000015:  000227
000016:  077777         (0012)       DATA  8388607E0
000017:  177627
000020:  100000         (0013)       DATA  -8388607E0
000021:  000627
000022:  077744         (0014)       DATA  1.7E38
000023:  147777
000024:  056216         (0015)       DATA  1.7E-38
000025:  105003
```

Single Precision Floating Accumulators (FAC)

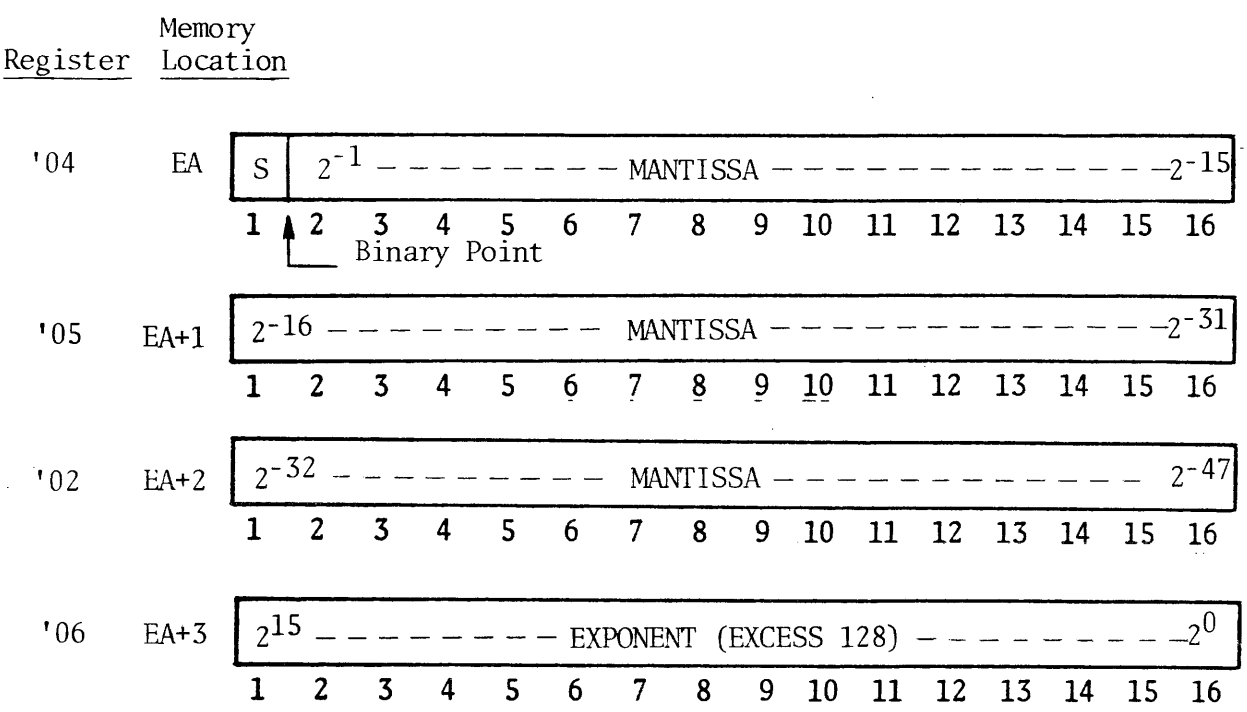
The operands for the main arithmetic functions (ADD, FLD, FST, FAD, FSB, FMP, FDV, FCS) are assumed to be in two consecutive memory words starting at the effective address (EA). However, the results of floating point operations are returned in a three-word format in registers 4, 5, and 6 - called the Single Precision Floating Accumulator (FAC). Floating loads (FLD) convert two-word memory data to the three-word format in FAC, and the FAC is assumed to contain the multiplicand or dividend prior to the FMP or FDV operations. In the FAC, the mantissa is expanded to 31 bits of precision ($\pm$2,147,483,647 decimal) and the exponent is allowed a full 16-bit word (10^{-9902} to 10^{+9825}).



During a series of floating operations, the FAC accumulates results with the increased accuracy of the three-word format. When the results are packed into the two-word memory format by an FST operation, the mantissa is truncated to 23 bits and only the eight least significant bits of the exponent are retained. An attempt to store an FAC exponent larger than the eight-bit field is defined as one of the floating exceptions (described later).

Double Precision

Double precision arithmetic employs a four-word format both in memory and in the double-precision floating accumulators (DFAC). The format is:



The format is identical to the single-precision floating accumulators (FAC) except that an additional 16 bits of mantissa are provided in register '02 (or the third memory word). The additional word extends the decimal precision to 14-1/2 decimal digits, or ± 140,737,488,355,327. The full 16-bit exponent permits a decimal exponent range from 10^{-9902} to 10^{+9825} in memory as well as in the DFAC.

Because the FAC overlaps the DFAC (except for the 16 least significant magnitude bits), numbers placed in the DFAC by double precision instructions can be processed by single precision instructions.

Following are examples of double-precision values as generated by Assembler DATA statements. (The D specifies double precision.)

```
                        (0016)  *
                        (0017)  *       DOUBLE PRECISION
                        (0018)  *
000026:  000000         (0019)          DATA    .0D
000027:  000000
000030:  000000
000031:  000000
000032:  040000         (0020)          DATA    .5D0
000033:  000000
000034:  000000
000035:  000200
000036:  040000         (0021)          DATA    1D0
000037:  000000
000040:  000000
000041:  000201
000042:  100000         (0022)          DATA    -.5D0
000043:  000000
000044:  000000
000045:  000177
000046:  100000         (0023)          DATA    -1D0
000047:  000000
000050:  000000
000051:  000200
000052:  040000         (0024)          DATA    128D0
000053:  000000
000054:  000000
000055:  000210
000056:  100000         (0025)          DATA    -128D0
000057:  000000
000060:  000000
000061:  000207
000062:  077777         (0026)          DATA    1.4073748355327D14
000063:  177777
000064:  177777
000065:  000257
000066:  100000         (0027)          DATA    -1.4073748355328D14
000067:  000000
000070:  000000
000071:  000257
000072:  044732         (0028)          DATA    1.2D9825
000073:  020777
000074:  136552
000075:  077777
000076:  056172         (0029)          DATA    1.2D-9902
000077:  050172
000100:  153152
000101:  100003
```

Normalization

The result of every floating point calculation is normalized in the FAC (or DFAC). In normal form, the most significant digit of the mantissa follows the binary point. If an operation produces a mantissa that is smaller than normal, the mantissa is shifted left until the most significant bit differs from the sign bit, and the exponent is decreased by one for each shift. Bits vacated at the right are filled by zeroes. If the result of an operation overflows the mantissa, it is shifted right one place, the overflow bit is made the most significant bit, and the exponent is increased by 1.

Floating Point Exceptions

Error conditions that arise during floating point operations may be detected and handled by floating exception (FLEX) interrupts. These are enabled when a user program makes location '74 non-zero (i.e., inserts a pointer to a routine that identifies and processes floating point error conditions). When any one of several types of error occurs, the CPU interrupts through location '74 to the error handling routine. A standard error handler, F$FLEX, is supplied as part of the Prime software library.

If location '74 is zero, FLEX interrupts do not occur; instead, the C bit is set. The user can test the C bit after possible error situations and take action as appropriate.

All FLEX interrupts vector through location '74 and locations '11 and '12 are set in certain cases to indicate the type of error condition. Table 8-2 shows the codes currently assigned.

In the basic arithmetic operations, increasing the exponent in the FAC (or DFAC) beyond 32639 is an overflow; decreasing it below -32896 is an underflow. Note that the exception is detected during an overflow or underflow of the full 16-bit exponent in the FAC or DFAC.

An attempt to store a single-precision number with an exponent greater than 127 or less than -128 in the two-word memory format results in a different type of exception. The number in FAC is not altered by the FST operation and so can be recovered if necessary.

Other detected exceptions are an attempt to divide by zero or to form an integer exceeding the capacity of the concatenated A and B registers ($\pm$ 30 bits or about $\pm$ 1 billion decimal).

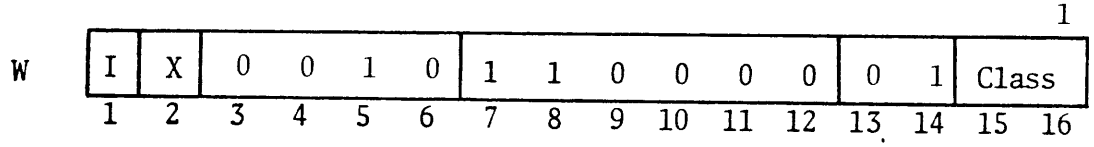
Table 8-2. Floating Exception Codes

Register 11		Register 12	Type of Exception
Single Pre.	Double Pre.		
$100	$200	--	Overflow/Underflow (Exponent exceeds approx. $10^{\pm 9800}$)
$101	$201	--	Division by zero
$102	--	(EA)	Attempt to store single precision exponent exceeding 8-bit memory format (>127, <-128)
$103	--	--	Attempt to form INT exceeding capacity of concatenated A and B registers (approx. $\pm$ 1 billion).

Note: $ indicates hexadecimal codes

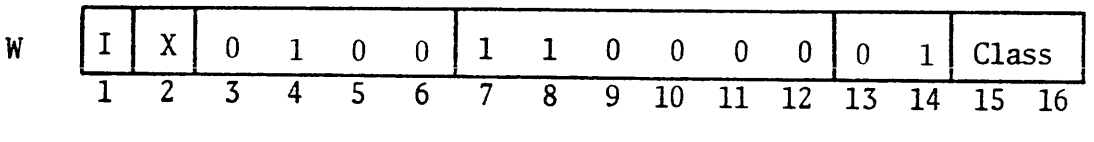
Single Precision Floating Point Instructions

FLD — Floating Load — '02



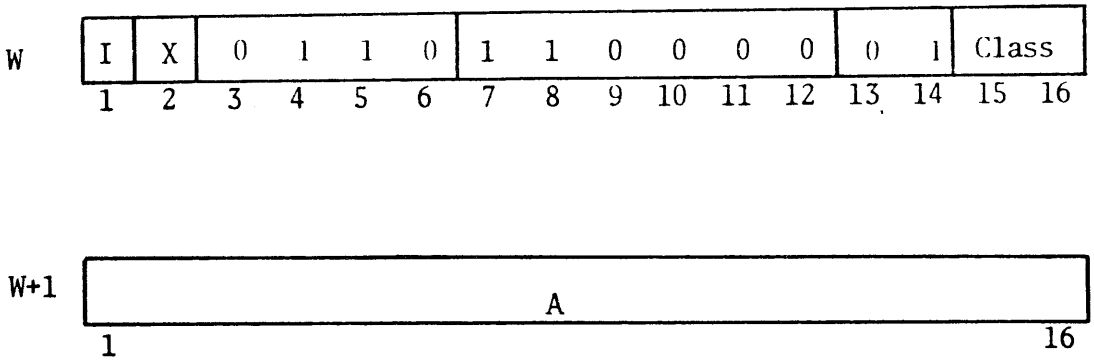
Load the double precision number contained in the two successive words at EA into the floating point accumulators (registers '04, '05, and '06).

FST — Floating Store — '04



Store the single precision floating point number contained in the FAC in two memory words starting at the effective address. Bits 24-31 of the 31-bit mantissa are truncated when written into the 23-bit capacity memory storage. However, the programmer can precede the FST with a RND instruction which adds 1 to bit 23 if bit 24 is 1. If the FAC contains an exponent outside the 8-bit range ($-128 \leq E \leq +127$), set C bit or initiate a floating exception.

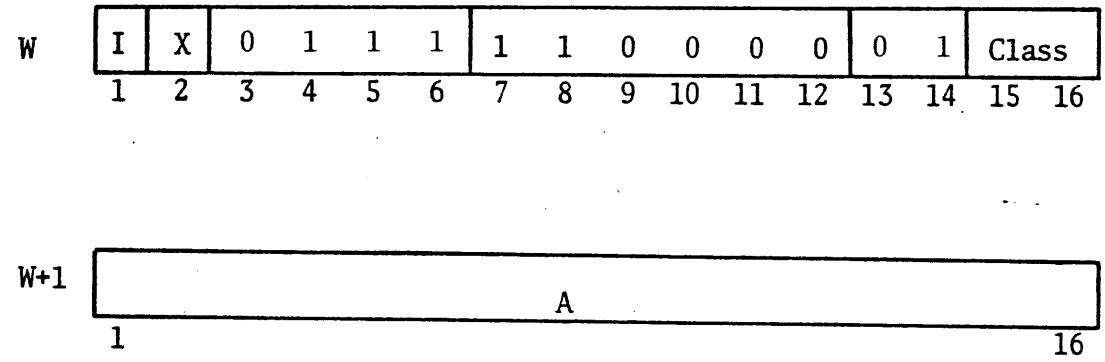
FAD Floating Add '06
1



Add the contents of the floating point number in the FAC to the floating point number at the effective address, and leave the resulting floating point number in the FAC registers. Addition of floating point numbers requires that their exponents be the same power of two. This is accomplished by incrementing the smaller exponent while right shifting its mantissa until the exponents match. With the exponents thus aligned, the mantissas are added.

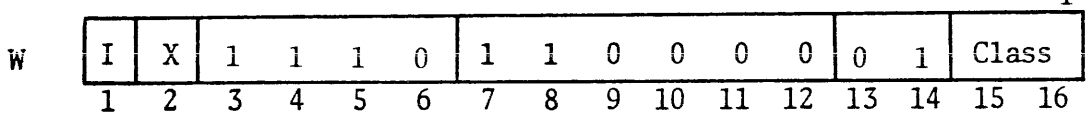
If there is an overflow from the most significant bit (not the sign), the sum mantissa is shifted right one place, the exponent is incremented by one and the overflow bit becomes the high-order bit in the normalized mantissa. If the result is otherwise not in normal form (as when numbers with unlike signs are added), the result is normalized. If there is an exponent under/overflow (<-32896, >+32639) the C bit is set or a floating exception is initiated.

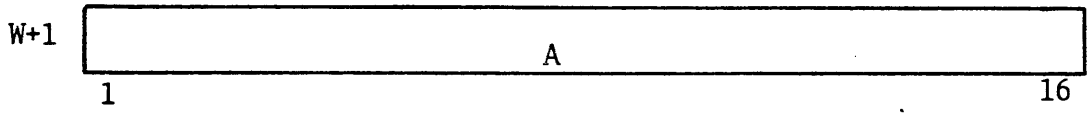
FSB Floating Subtract '07
1



Subtract the contents of EA from FAC by aligning exponents, forming the two's complement of the mantissa of (EA) and proceeding as in FAD.

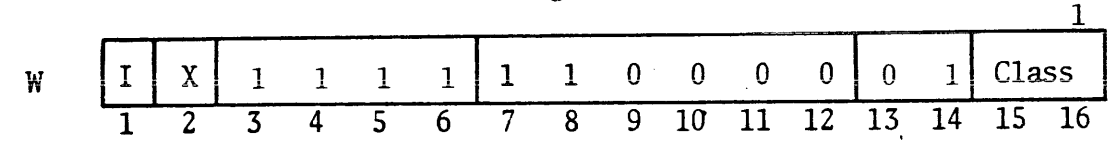
FMP Floating Multiply '16

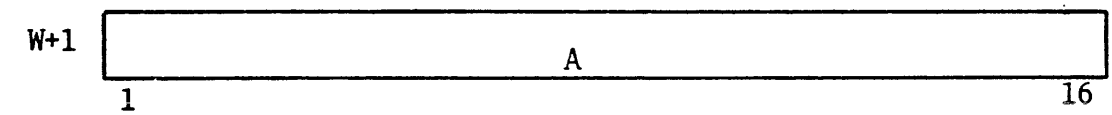




Multiply the contents of FAC by (EA) and place the results in FAC. The mantissa of the quotient is a normalized binary fraction and its exponent is the sum of the multiplier and multiplicand exponents plus the number of powers of two needed to compensate the shifts required to normalize the mantissa. If there is an exponent under/overflow, the C bit is set or floating exception is initiated.

FDV Floating Divide '17

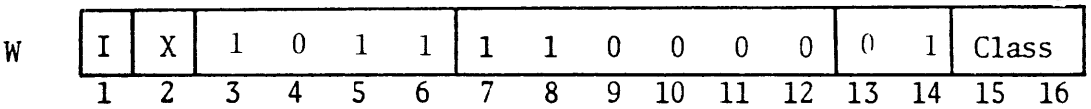


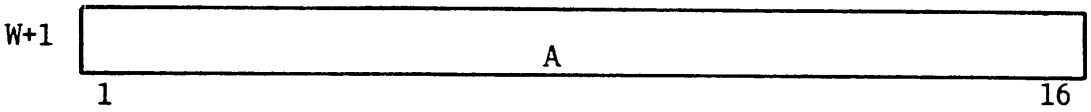


Divide the contents of FAC by the number in EA and leave the quotient in FAC with the mantissa normalized.

If there is an exponent under/overflow, the C bit is set or a floating exception is initiated.

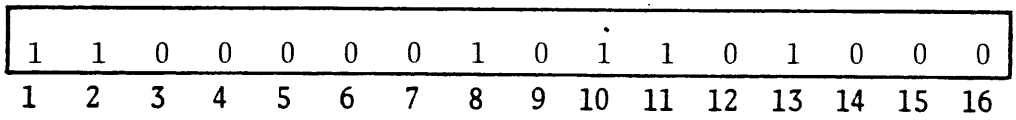
FCS Compare and Skip '11 1





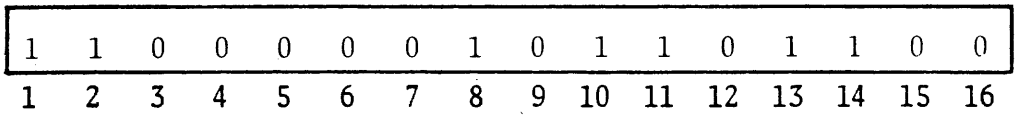
If the contents of FAC are greater than the contents of EA, execute the next instruction. If FAC contents equal those of EA, skip the instruction following location FCS and execute FCS+2. If FAC contents are less than EA, skip 2 locations and execute the FCS+3 instruciton.

FLOT Float '140550



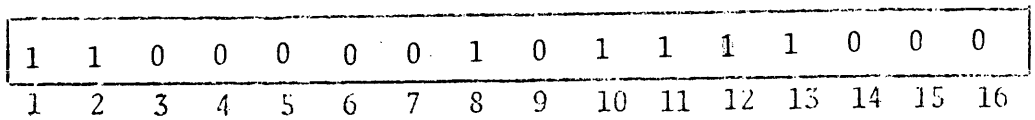
Take the double precision integer in the concatenated A and B registers and convert it into a normalized floating point number in the FAC registers.

INT Fix As Integer '140554



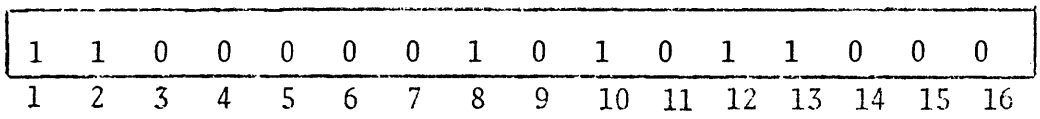
Convert the integer part of floating point number in the FAC to a double precision two's-complement integer in the concatenated A and B registers with the binary point following bit 31. If the FAC contains a number too large to be represented in the double precision integer format, the C Bit is set or a floating exception is initiated.

FRAC Fix As Fraction '140570



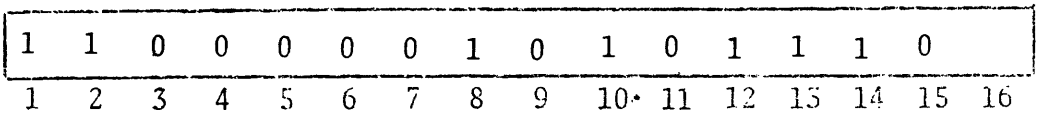
Convert the fractional part of the floating point number in FAC to a binary fraction in the concatenated A and B registers with the binary point between A_1 and A_2.

FCM Complement '140530



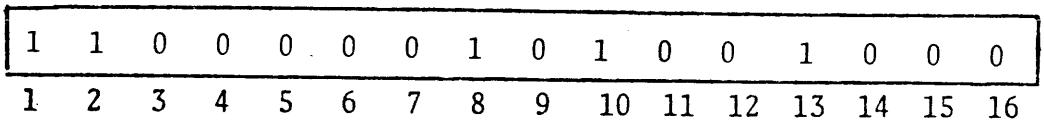
Two's complement the mantissa of FAC and normalize if necessary. (Overflow is possible.)

FRN Round Up '140534



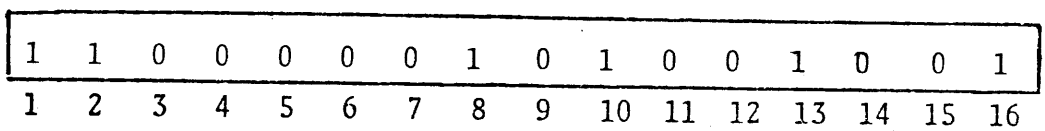
If bit 24 of the mantissa in FAC is 1, add 1 to bit 23. (Overflow is possible.)

FSZE Floating Skip if Zero '140510



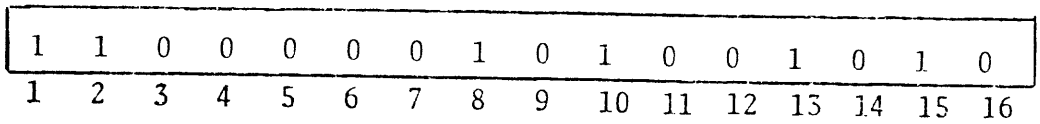
If register '04 =0, skip next location.

FSNZ Floating Skip if Not Zero '140511



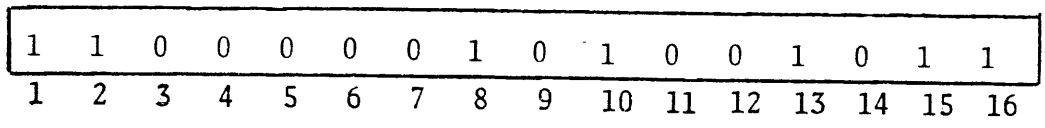
If register '04 is not equal to zero, skip next location.

FSMI Floating Skip If Minus '140512



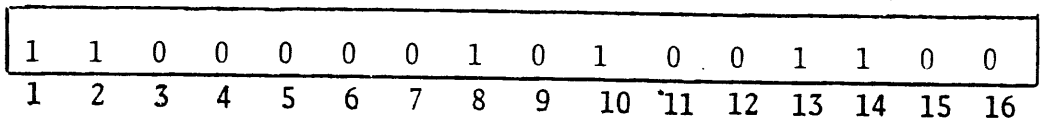
If bit 1 of register '04 is 1, skip next location.

FSPL Floating Skip If Plus '140513



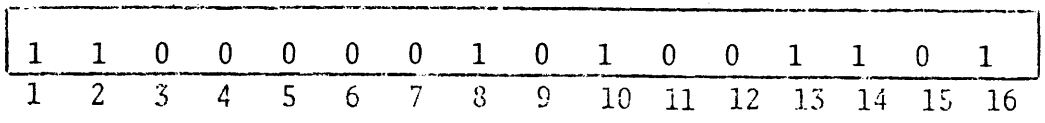
If bit 1 of register '04 is 0, skip next location.

FSLE Floating Skip If Less or Equal Than Zero '140514



If register '04 is less than or equal to zero, skip next location.

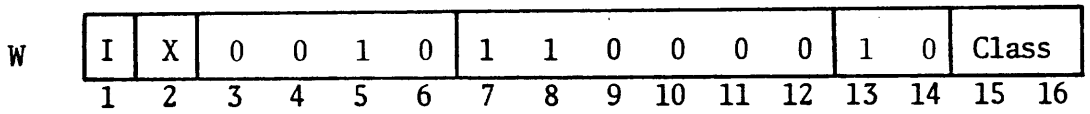
FSGT Floating Skip If Greater Than Zero '140515

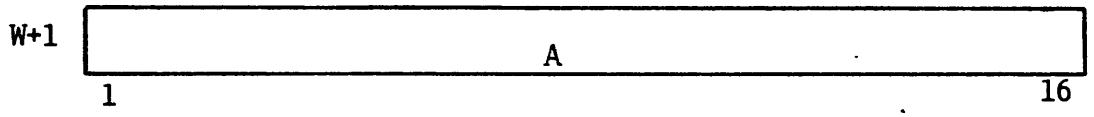


If register '04 is greater than zero, skip next location.

Double Precision Floating Point Instructions

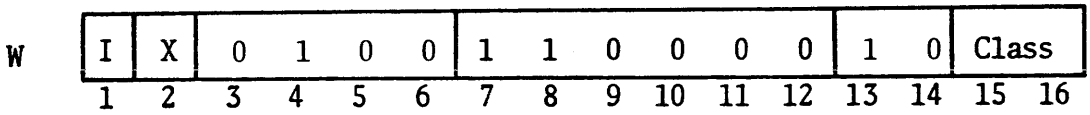
DFLD — Double Precision Floating Load — '02 (2)

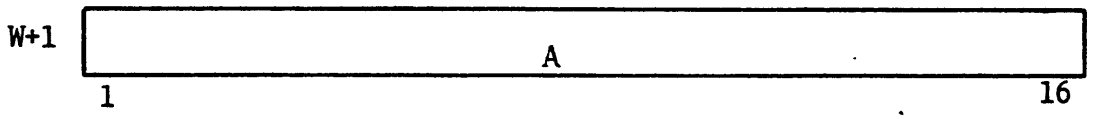




Load the double precision number contained in the four memory words at EA into the DFAC registers.

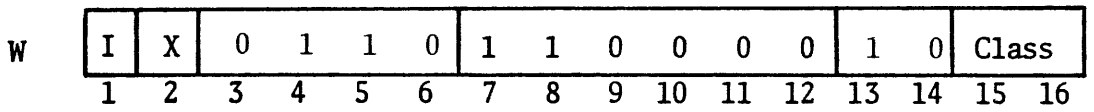
DFST — Double Precision Floating Store — '04 (2)

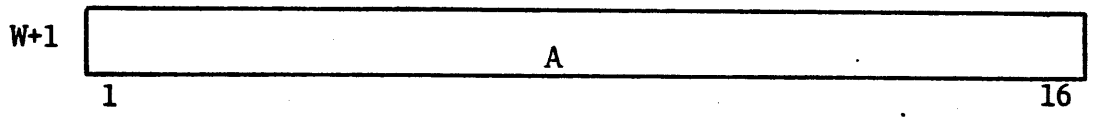




Store the double precision number contained in four double precision FACs into the location specified by EA. Exponent and mantissa bit capacities are the same so that no floating point exceptions are possible.

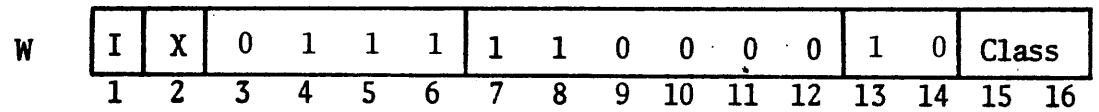
DFAD Double Precision Floating Add '06 2

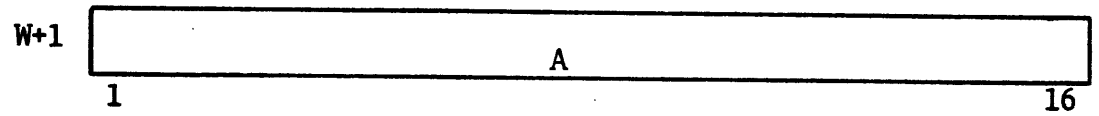




Add the double precision number in DFAC to the double precision number starting at EA and leave the result in DFAC. (Same procedure as FAD except a 47-bit mantissa is produced.)

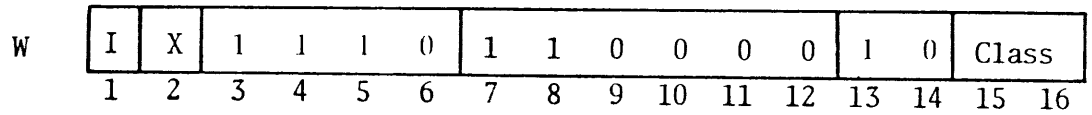
DFSB Double Precision Floating Subtract '07 2

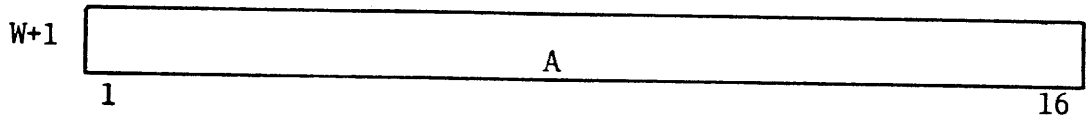




Subtract the double precision floating point number starting at EA from the double precision floating point number in DFAC. (Same procedure as FSB except a 47-bit mantissa is produced.)

DFMP Double Precision Floating Multiply '16
2

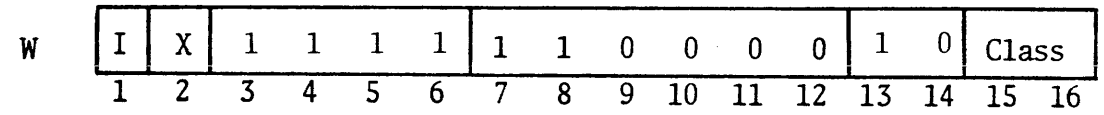


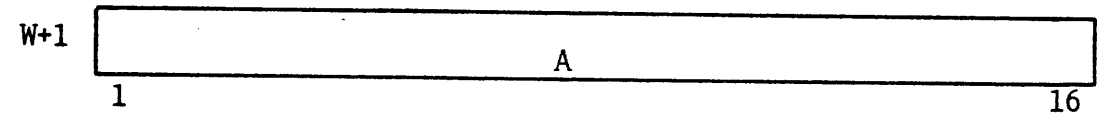


Multiply the double precision floating point number in DFAC by the double precision floating number starting at EA and leave the result in DFAC. Exponents are added and, after mantissas are multiplied, the product is normalized.

An exponent under/overflow sets the C bit or initiates a floating exception interrupt.

DFDV Double Precision Floating Divide '17
2

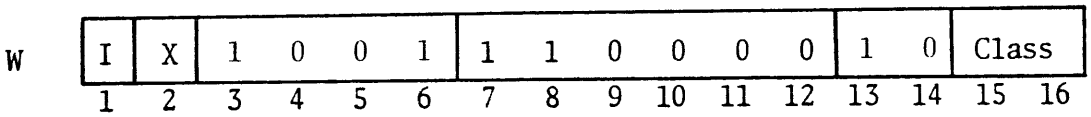


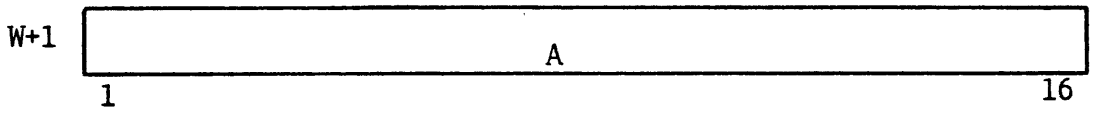


Divide the double precision floating point number in DFAC by the double precision floating point number starting at EA and leave the result in DFAC. Exponents are subtracted, and after the divisor mantissa is divided into the dividend mantissa, the quotient is normalized.

An under/overflow or an attempt to divide by zero sets the C bit or initiates a floating exception interrupt.

DFCS Double Floating Point Compare and Skip '11 2



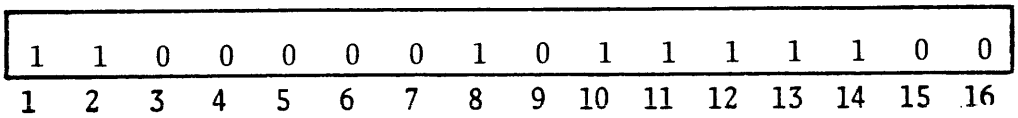


If DFAC is greater than (EA), execute the next instruction.

If DFAC equals (EA), skip the instruction at the next location and execute the instruction at second location following.

If DFAC is less than (EA), skip next two locations in instruction sequence and execute the instruction at third instruction location.

DFCM Double Precision Floating Complement '140574



Two's complement the double precision mantissa in DFAC and normalize if necessary. Overflow is possible.

WRITABLE CONTROL STORE

See Writable Control Store Data Sheet

SECTION 9

AUXILIARY CPU FUNCTIONS

These functions are available as options for any Prime CPU. (Automatic Program Load for ASR and HSR is standard on the 300.)

PRIME's family of computers are microprogrammed using a 64-bit wide microprocessor as the control unit. This microprocessor or "inner processor" has a control store containing microinstructions arranged as microprograms. These execute the computer's or "outer processor's" machine instructions, I/O and control panel functions. WSC extends the PRIME 300's basic control store (512 words of PROM, Programmable Read Only Memory) with 256 words of RAM (Random Access Memory). Microprograms for WCS are written using PRIME's Micro Assembler and are loaded from main memory using a PRIME-supplied loader. Control is turned over to WCS microprograms by executing a set of special 'Jump to WCS' instructions.

WCS CAPABILITIES

The capabilities of WCS are a function of both the inner and outer processors. The outer processor has 32 general purpose registers, an arithmetic and logic unit, main memory interface, I/O bus interface and an 8-bit auxiliary counter. This general architecture is enhanced by hardware assists to speed the execution of skips, instruction fetches, and decode operations.

The inner processor contains a control unit and control store. It features a three-deep push/pop stack and condition testing to assist in subroutining, and microprogram trapping. The microprocessor fetches and executes microinstructions from control store. The 64-bit wide microinstruction format permits multiple functions to be performed by a single instruction. A narrower word may require several instructions to accomplish an equivalent task. Also, the generalized structure of the outer processor broadens the spectrum of tasks that can be implemented with the powerful inner processor.

Microprogramming can be used to gain a speed advantage and minimize main memory storage compared to in-line programming with the standard instruction set. A microprogrammed subroutine eliminates individual instruction fetches from main memory. A typical microinstruction executes in 280 ns - much faster than main memory cycle time. Thus, when main memory references are minimized, execution speed increases. Furthermore, since a single in-line instruction transfers control to a WCS subroutine, there can also be a more efficient use of main memory.

XCS BOARD

The PRIME microprocessor can address up to 4096 words of control store. To extend the control store past the 512 words located on the central processor (cp) board, a second board is used. The extended control store (XCS) board connects to the cp board in adjacent chassis slots with three short ribbon cables attached to the boards' rear edge connectors.

Both WCS (RAM) and floating point arithmetic (PROM) can populate the XCS board depending upon which type processor and option has been ordered.

PROGRAMMING

Loading WCS

The XCS board also interfaces to the I/O bus to facilitate the loading of WCS from main memory. This is accomplished by first loading the A register with the

starting address of WCS and transferring this to the XCS board with an OTA instruction. Data is transferred to WCS by a sequence of LDA & OTA instructions. The WCS control logic automatically packs four sequential 16-bit words into one 64-bit microinstruction, and advances to the next WCS location.

"Jump to WCS" Instructions

Four "Jump to WCS" instructions have been added to the PRIME 331, 332, 333 and 334 processors:

EPMX - Enter page mode and jump to XCS.

EVMX Enter virtual mode and jump to XCS.

(Virtual mode = page mode & restricted execution mode)

ERMX - Enter restricted execution mode and jump to XCS.

LPMX - Leave page mode and jump to XCS.

All of the above are 2 word instructions. The first word is the op code; the second is a pointer to a main memory location containing the address of the microinstruction. This format permits pure procedures to be separated from variables (the control store address may be a variable) and facilitates recursive, re-entrant programming. To transfer control to WCS takes approximately 2.5 microseconds. These are restricted instructions and cause a trap when executed with restricted execution mode enabled. This feature gives the executive software the ability to decide how to handle user-level requests for WCS access.

Use of PMA

RIME's Macro Assembler (PMA) can be used to create special mnemonic representations for microprograms. These are actually one instruction macro's a "jump to WCS" with a unique pointer and WCS address. Once the macro has been defined, the user can refer to his microprogram as a mnemonic, the way he would to a standard PRIME instruction.

SOFTWARE

Micro Assembler

PRIME's macro assembler allows the user to create microprograms with full symbolic assembly capabilities. Symbolic source code is assembled and object code created in a format to be loaded into WCS and/or printed in hexidecimal format. The macro assembler runs under DOS and requires a DOS system with at least 32K words of main memory. The DOS Editor (ED) is used to enter and modify source statements.

Loader

This enables a DOS user to load WCS from main memory.

Test and Verification

Test and verification routines are also provided.

Microprogramming Course

A one week microprogramming course must be attended as a prerequisite to installing and using the WSC feature. The course is designed to give the student hands-on experience writing, debugging, and executing microprograms. For the student to properly benefit from the course, he should at least familiarize himself with the reading material that will be provided him prior to the course. Preferably he will be experienced in machine level programming, logic design, and computer architecture.

The power monitor and related features combine to provide automatic restart from memory after a power failure has been corrected and AC power is restored. Four distinct and interrelated functions are provided by this option: sensing of line voltage not within the PRIME 200's operating specifications, storing of processor status information when power fails, battery refreshing of MOS memory, and automatic restart when AC power is restored.

OPERATION

When the computer is running, AC power is constantly monitored to assure that it satisfies the computer's voltage requirement. Should voltage drop below the specified limit (95 VAC) an automatic power failure interrupt is executed through location '60. This gives the program approximately one millisecond to prepare for loss of AC power. At the end of this interval, a system clear is generated to prevent random logic transitions from altering memory as power is going down. The back-up battery is used to refresh the contents of the MOS memory and provide power to essential processor logic. Use of the battery is indicated by the flashing of the STOP light on the computer's control panel. When power returns to proper operating specifications, the processor automatically restarts at location '1000 and the battery begins recharging.

EQUIPMENT CONFIGURATION

The equipment consists of two separate components: an assembly with panel which mounts on the computer's power supply, and the battery which may be mounted at either the front or back of the equipment rack. The assembly which mounts on the power supply contains the battery charging circuitry and DC to DC conversion circuitry to supply

16.2 volts + 0.5 volts, 2.5 amps
3.5 volts $\mp$ 0.5 volts, 2.0 amps
5.0 volts $\mp$ 0.25 volts, 7.0 amps

with bendback and overvoltage protections. The panel associated with this assembly contains a full-charge indicator, meter terminals, and battery terminals.

The battery is a sealed, gel-electrolyte unit which can be stored or charged in any position. Two 20 Amp hour cells can be housed in the battery mounting, which requires 7" of panel space and a depth of 8".

SPECIFICATION SUMMARY

PRIME 200 input voltage requirement:
95 to 125 VAC, 47 to 63Hz.

Allowable Temporary Voltage Drops:

% Drop From 120 VAC	Maxumum Allowable Duration
100%	12.0 msec.
40%	20.8 msec.
24%	480.0 msec.

Battery:
20 Amp-Hour

Operating Temperature:
0°to 50°C

Battery Back-up Time:

Memory	1 Battery	2 Batteries
4K	6.7 Hrs.	13.4 Hrs.
8K	6	12
16K	4.3	8.7
24K	3.1	6.2
32K	2.4	4.9

HOW TO ORDER

Specify Type Number and Description

Type Number	Description
240	Power Monitor, Power Failure Interrupt and Automatic Restart Protection.

Specifications subject to change without notice.

The automatic program load options enable the operator to load programs from devices such as fixed and moving head disks, and paper tape simply by initiating a hardware bootstrap from the control panel. They may also be used to reload programs when power is restored following a power failure.

These features save considerable time and effort by eliminating the tedious and error-prone procedure of manually keying in a bootstrap loader one word at a time.

There are three basic types of automatic loaders, one for the fixed-and moving-head disks, one for magnetic tape, and one for the ASR and high speed paper tape readers.

All versions are implemented as part of the central processor and the operator uses sense switches to specify the input device.

The disk version reads the contents of 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 magnetic tape version reads the first record from magnetic tape unit 0 into memory beginning at location '770. After reading the data, the processor begins normal program execution at location '1000. Like the disk, the data read from tape is entirely at the discretion of the programmer.

The paper tape version reads any PRIME self-loading tape. Tapes of the assembler, linking loader, text editor and other basic programs are available 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.

SPECIFICATION SUMMARY

Operating Characteristics

SSW 14	15	16	Function Selected
0	0	0	= Start @ '1000
0	0	1	= APL from TTY
0	1	0	= APL from HSR
0	1	1	= APL from FHD
1	0	0	= APL from MHD
1	0	1	= APL from Magnetic Tape

Data Rate - Input Device Dependent

HOW TO ORDER

Specify Type Number and Description

Type Number	Description
142	Automatic Program Load from Teletype and Paper Tape Reader for Series 100 Processors.
143	Automatic Program Load from Magnetic Disk for Series 100 processors.
144	Automatic Program Load from Magnetic Tape for Series 100 processors.
242	Automatic Program Load from Teletype and Paper Tape Reader for Series 200 Processors.
243	Automatic Program Load from Magnetic Disk for Series 200 processors.
244	Automatic Program Load from Magnetic Tape for Series 200 processors.

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 TIMING

Mnemonic	Function	100 CP	200 CP	300 CP (600 n sec.)	300 CP (440 n sec.)
A1A	Add One To A	1.76	1.36	1.28	1.12 μs.
A2A	Add Two To A	1.76	1.36	1.28	1.12
ACA	Add C To A	1.76	1.36	1.28	1.12
ADD	Add To A	2.44	1.96	1.88	1.56
ALL	A Left Logical	1.4+0.36N	1.08+0.24N	1.08+0.2N	0.92+0.2N
ALR	A Left Rotate	1.4+0.36N	1.08+0.24N	1.08+0.2N	0.92+0.2N
ALS	A Left Shift	1.76+0.36N	1.28+0.24N	1.28+0.2N	1.12+0.2N
ANA	And With A	2.44	1.96	1.88	1.56
AOA	Add One To A	1.76	1.36	1.28	1.12
ARL	A Right Logical	1.4+0.36N	1.08+0.24N	1.08+0.2N	0.92+0.2N
ARR	A Right Rotate	1.4+0.36N	1.08+0.24N	1.08+0.2N	0.92+0.2N
ARS	A Right Shift	1.4+0.36N	1.08+0.24N	1.08+0.2N	0.92+0.2N
CAI	Clear Active Interrupt	1.76	1.28	1.28	1.12
CAL	Clear A Left Byte	1.76	1.36	1.28	1.12
CAR	Clear A Right Byte	1.76	1.36	1.28	1.12
CAS	Compare And Skip	3.52	2.16	2.26	1.94
		3.83	2.44	2.48	2.16
		4.24	2.72	2.60	2.36
CAZ	Compare A With 0 And Skip	2.12	1.52	1.48	1.32
		2.48	1.80	1.78	1.52
		2.84	2.08	1.88	1.72
CEA	Compute Effective Address	3.24+1.04I	2.84+0.88I	2.16+0.84I	2.04+0.68I
CHS	Change Sign	1.76	1.36	1.28	1.12
CMA	Complement A	1.76	1.36	1.28	1.12
CRA	Clear A	1.76	1.28	1.28	1.12
CRB	Clear B	1.76	1.28	1.28	1.12
CRLP	Call Recursive Procedure			3.48	3.08
CRL	Clear Long	2.12	1.48	1.48	1.32
CSA	Copy Sign Of A	1.76	1.36	1.28	1.12
DAD	Double Precision Add	4.56	3.56	3.28	2.80
DBL	Enter Dbl Prec Mode	1.76	1.28	1.28	1.12
DFAD	Dbl Precision Floating Add		11.51+ .72A+1.N	11.0+ .64A+.92N	10.0+ .64A+.92N
DFCM	Dbl Prec Flting Complement		5.96	6.52	6.20
DFCS	Dbl Pr Flting Comp & Skip		5.18	4.725	4.245
DFDV	Dbl Pr Floating Divide		73.3	67.52	66.56
DFLD	Dbl Pr Floating Load		5.84	5.56	4.60
DFMP	Dbl Pr Floating Multiply		56.73	52.62	51.66
DFSB	Dbl Pr Floating Subtract		11.51+ .72A+1.N	11.0+ .64A+.92N	10.04+ .64A+.92N
DFST	Dbl Prec Floating Store		6.56	5.48	5.00
DIV	Divide	13.78	13.24	11.2725	10.9525
DLD	Double Load	3.72	2.96	2.72	2.24
DRX	Decrement Replace Index & Skip	2.48 2.12	1.84 1.56	1.78–1.48	1.52 1.32
DSB	Double Precision Subtract	3.6	3.32	3.22	2.74
DST	Double Store	3.72	3.04	2.64	2.32
E16S	Enter 16K Sectored Mode	1.76	1.28	1.28	1.12
E32R	Enter 32K Relative Mode	1.76	1.28	1.28	1.12
E32S	Enter 32K Sectored Mode	1.76	1.28	1.28	1.12
E64R	Enter 64K Relative Mode	1.76	1.28	1.28	1.12
EAA	Effective Address To A			2.52	2.20
EMCM	Enter Machine Check Mode		1.28	1.28	1.12
ENB	Enable Interrupt	1.76	1.28	1.28	1.12
ENTR	Enter Recursive Procedure			3.24	2.84
EPMJ	Enter Page Mode & Jump		3.72	3.28	2.80
ERA	Exclusive Or To A	2.44	1.96	1.88	1.56
ERMJ	Enter Restricted Mode & Jump		3.72	3.28	2.80
ESIM	Enter Std Interrupt Mode	1.76	1.28	1.28	1.12
EVIM	Enter Vectored Interrupt Mode	1.76	1.28	1.28	1.12
EVMJ	Enter Virtual Mode & Jump		3.72	3.28	2.80
FAD	Floating Add		9.35+ .48A+.8N	8.75+ .48A+.72N	8.11+ .48A+.72N
FCM	Floating Complement		3.96	3.45	3.32
FCS	Floating Compare & Skip		4.71	4.62	3.98
FDV	Floating Divide		39.46	37.92	37.28

Mnemonic	Function	100 CP	200 CP	300 CP (600 n sec.)	300 CP (440 n sec.)
FLD	Floating Load		4.6	4.36	3.72
FLOT	Float (A & B) As An Integer Normalized		5.8+.8N	5.2+.72N	4.88+.72N
FLX	Load Floating Index		3.36	3.32	2.84
FMP	Floating Multiply		27.82	25.20	24.56
FRAC	Fix As Fraction		4.96+.48N	4.62+.44N	4.3+.44N
FRN	Round-Up		3.68	3.32	3.16
FSB	Floating Subtract		9.35+ .48A+.8N	8.75+ .48A+.72N	4.88+ .48A+.72N
FSGT	Floating Skip If >0		3.12 2.84	2.88 2.68	2.72 2.52
FSLE	Floating Skip If >=0		3.12 2.84	2.88 2.68	2.72 2.52
FSMI	Floating Skip If Minus		3.12 2.84	2.88 2.68	2.72 2.52
FSNZ	Floating Skip If Not 0		3.12 2.84	2.88 2.68	2.72 2.52
FSPI	Floating Skip If Plus		3.12 2.84	2.88 2.68	2.72 2.52
FST	Floating Store		5.24	4.80	4.40
FSZF	Floating Skip If 0		3.12 2.84	2.88 2.68	2.72 2.52
HLT	Halt	N/A	N/A	N/A	N/A
IAB	Interchange A & B	2.84	1.88	1.84	1.68
ICA	Interchange Bytes Of A	1.76	1.36	1.28	1.12
ICL	Interchange Bytes Of A & Clear Left Byte	1.76	1.36	1.28	1.12 μS
ICR	Interchange Bytes Of A & Clear Right Byte	1.76	1.36	1.28	1.12
IMA	Interchange Memory & A	3.72	2.88	2.72	2.32
INA	Input To A From I/O	3.92 3.2	2.72 2.06	2.64 2.04	2.28 1.88
INH	Inhibit Interrupt	1.76	1.28	1.28	1.12
INK	Input Keys To A	3.56	2.44	2.28	2.12
INT	Fix As Interger		6.20+0.6N	5.70+0.62N	5.50+0.62N
IRS	Increment Memory Replace & Skip	3.16 3.04	2.80 2.64	2.72 2.56	2.33 2.16
IRX	Increment Replace Index & Skip	2.48 2.12	1.84 1.56	1.78 1.48	1.52 1.32
ISI	Input Serial Interface	1.76	1.56	1.48	1.32
JDX	Jump & Decrement X			2.72	2.4
JEQ	Jump If Equal To 0			2.72	2.4
JGE	Jump If >=0			2.72	2.4
JGT	Jump If >0			2.72	2.4
JIX	Jump & Increment X			2.72	2.4
JLE	Jump If <=0			2.72	2.4
JLT	Jump If <0			2.72	2.4
JMP	Unconditional Jump	1.76	1.28	1.28	1.12
JNE	Jump If Not Equal To 0			2.72	2.4
JST	Jump To Subroutine	3.36	2.64	2.56	2.16
JSX	Jump & Store Return In X			2.88	2.56
LDA	Load A	2.44	1.88	1.88	1.56
LDX	Load Index (Not Indexed)	2.44	1.88	1.88	1.56
LEQ	Convert A=0 To True	2.48 2.12	1.84 1.64	1.76 1.56	1.60 1.40
LF	Convert A To False	2.48	1.84	1.56	1.40
LGE	Convert A >= 0 To True	2.48 2.12	1.84 1.64	1.76 1.56	1.60 1.40
LGL	A Left Logical	1.4+0.36N	1.08+0.24N	1.08+0.2N	0.92+0.2N
LGR	A Right Logical	1.4+0.36N	1.08+0.24N	1.08+0.2N	0.92+0.2N
LGT	Convert A >0 To True	2.48 2.12	1.84 1.64	1.76 1.56	1.60 1.40
LLE	Convert A H<= 0 To True	2.48 2.12	1.84 1.64	1.76 1.56	1.60 1.40
LLL	Long Left Logical	1.4+0.72N	1.08+0.48N	1.08+0.4N	0.92+0.4N
LLR	Long Left Rotate	1.4+1.08N	1.08+0.68N	1.08+0.6N	0.92+0.6N
LLS	Long Left Shift	1.76+0.72N	1.28+0.48N	1.28+0.4N	1.12+0.4N
LLT	Convert A <0 To True	2.48 2.12	1.84 1.64	1.76 1.56	1.60 1.40
LMCM	Leave Machine Check Mode		1.28	1.28	1.12
LNE	Convert A Not = 0 To True	2.48 2.12	1.84 1.64	1.76 1.56	1.60 1.40
LPMJ	Leave Page Mode & Jump			3.28	2.80
LRL	Long Right Logical	1.4+.72N	1.08+.08+.48N	1.08+.4N	0.92+.4N
LRR	Long Right Rotate	1.4+1.08N	1.08+.68N	1.08+.6N	0.92+.6N
LRS	Long Right Shift	1.4+.72N	1.08+.68N	1.08+.4N	0.92+.4N
LT	Convert A To True	2.12	1.64	1.56	1.40
MPY	Multiply	11.26	10.48	9.04	8.72
NOP	No Operation	2.48	1.68	1.68	1.52

Mnemonic	Function	100 CP	200 CP	300 CP (600 n sec.)	300 CP (440 n sec.)
NRM	Normalize	3.42+1.08N	2.96+.68N	2.96+.6N	2.24+.6N
OCP	Output Control Pulse	3.2	2.64	2.64	2.48
OSI	Output Serial Interface	2.12	1.48	1.48	1.32
OTA	Output A To I/O	4.28 2.84	2.84 1.88	2.64 2.04	2.28 1.88
OTK	Output A To Status Keys	2.84	2.12	2.04	1.88
PID	Positive For Interger Divide	2.84	2.08	1.88	1.72
PIM	Pos. For Interger Multiply	2.48	1.84	1.68	1.52
RCB	Reset C Bit	1.76	1.36	1.28	1.12
RMC	Reset Machine Check Flag		1.28	1.28	1.12
RIN	Return From Recursive Proc			4.36	3.88
S1A	Subtract One From A	1.76	1.36	1.28	1.12
S2A	Subtract Two From A	1.76	1.36	1.32	1.16
SCB	Set C Bit	1.76	1.36	1.28	1.12 μs
SGL	Enter Single Precision Mode	1.76	1.28	1.28	1.12
SKP	Skip Unconditionally	3.2	2.32	2.12	1.96
All Skips	Skip On Conditions	3.2 2.84	2.32 2.04	2.12 1.92	1.96 1.76
SOA	Subtract One From A	1.76	1.36	1.28	1.12
SSM	Set Sign Minus	1.76	1.36	1.28	1.12
SSP	Set Sign Plus	1.76	1.36	1.28	1.12
STA	Store A	2.32	1.96	1.76	1.52
STX	Store Index (Not Indexed)	2.32	1.96	1.76	1.52
SUB	Subtract From A	2.44	1.96	1.88	1.56
SVC	Supervisor Call	4.24	3.24	3.20	2.80
TCA	Two's Complement A	2.12	1.64	1.48	1.32
VIRY	Exec Verification Routine		128	128	128
XCA	Transfer A To B & Clear A	2.48	1.68	1.68	1.52
XCB	Transfer B To A & Clear B	2.48	1.68	1.68	1.52
XEC	Execute Effective Address Content As Next Instruction			3.24+	2.92+

Timing Notes:

1. Values For N =
   number of steps in shifts,
   number of steps in normalize,
   number of steps in normalize routine of floating point operations.
2. A = number of adjust steps in floating point.
3. I = level of indirection.

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	"	
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	
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	o	
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., 145 Pennsylvania Avenue, Framingham, Massachusetts 01701