LOGICON 2+2

SYSTEM REFERENCE

MANUAL

LOGICON INC.
1075 CAMINO DEL RIO, SOUTH
SAN DIEGO, CALIFORNIA

15 December 1970

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

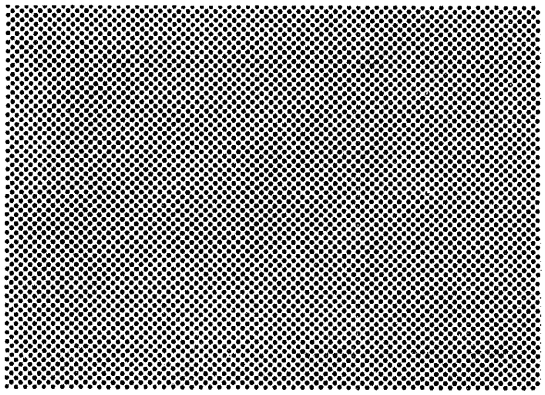
TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES



Preface

This reference manual defines the LOGICON 2+2 System hardware configuration, the characteristics of the major components, the organization and instruction repertoire of the processors, and the characteristics of the software interface with the I/O subsystem and peripheral devices.

As we expect to improve this manual in future revisions, all readers are earnestly requested to send corrections and comments to:

LOGICON 2+2 System Documentation
LOGICON, Inc., San Diego, Calif.

I...

Introduction

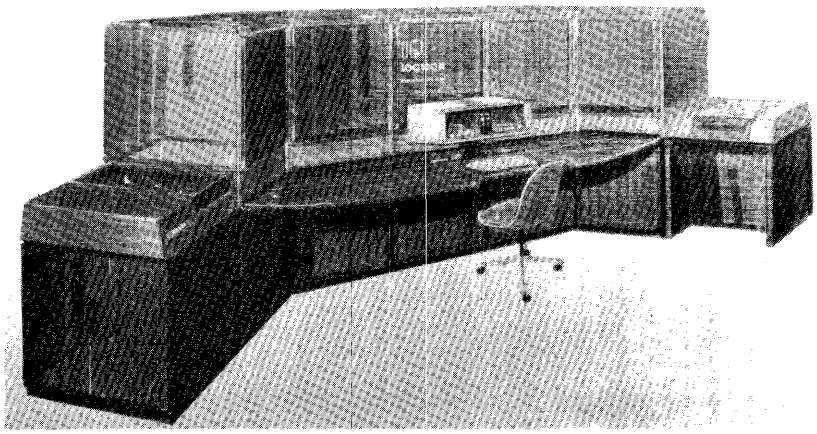
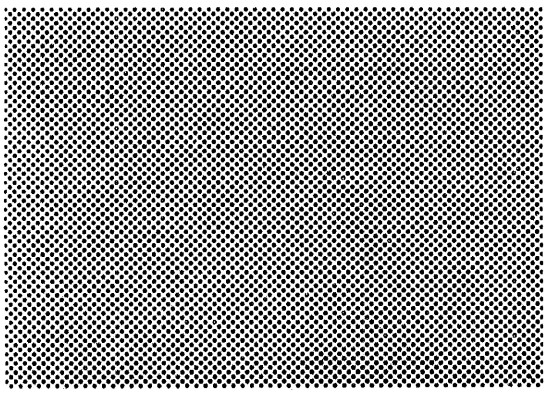


Figure 1-1. Typical LOGICON 2+2 Installation

The 2+2 System is LOGICON's answer to the demand for a medium-sized, modular, highly responsive, inter-active shared-access system. The 2+2 exemplifies the versatility required for today's computing tasks. With its 215-instruction repertoire, micro-programmed hardware, "virtual memory" capability, and flexible peripheral interface it provides a tremendous capacity and excellent response for a multitude of applications.

With the wide range of system hardware and software, the 2+2 user can operate with a minimum of system components to serve his present needs, yet easily expand to meet future system requirements.

The LOGICON 2+2 (Figure 1-1) offers the user a new, more efficient approach to time-sharing.



II ...

System Description

The LOGICON 2+2 is a multiprocessor system, the heart of which consists basically of two processors organized as computers and two processors organized as peripheral controllers. The following paragraphs discuss this system both in terms of its functional and its equipment organization.

FUNCTIONAL ORGANIZATION

A functional block diagram of the LOGICON 2+2 System is given in Figure 2-1. In this figure, solid lines represent information lines — dotted lines are control lines. The various blocks have the following general functions:

- The Application Processor (AP) is a microprogrammed computer that executes all user programs. It communicates with both AP and CP memory, and with the CP via interrupt lines.

- The Control Processor (CP) is a microprogrammed computer that performs system scheduling and I/O control functions. It communicates with AP and CP memory, with the AP via interrupt lines, and with all storage and communication devices.

- Memory accesses by the AP to its own memory are made through a mapping and protection unit called the Virtual Address Translator (VAT), which is controlled by the AP.

- AP memory may also be accessed by the swapping storage system, the file storage system, the backup storage system, and the CP. AP memory contains the active portion of the program currently being run for a user, some resident system code, and possibly parts of other users programs waiting to be swapped out or run.

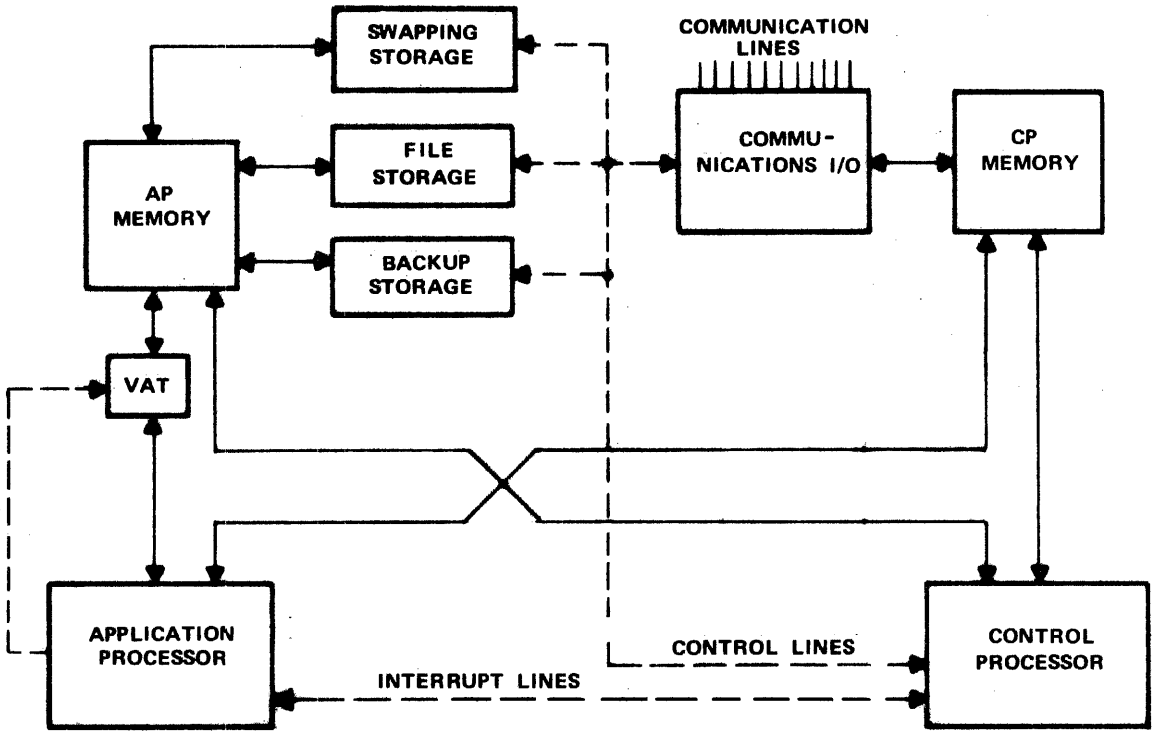


Figure 2-1. LOGICON 2+2 System, Functional Block Diagram

- CP memory is accessed by the CP, AP, and the Communication I/O subsystem. It holds the bulk of the Monitor program and all communications I/O buffers.

- The swapping storage subsystem is mechanized by a drum and a Drum Processor (DP). It is capable of swapping programs and data in or out of AP memory at a rate of 1 million 16-bit words per second, under control of the CP. It has storage capacity for 1 to 8 million words.

- The file storage subsystem consists of from 1 to 8 disk drives, each capable of storing 14 million words, controlled by a Peripheral Processor (PP) which executes commands given it by the CP. Data is exchanged with AP memory at a rate of approximately 156,000 words per second. Average access time is 45 milliseconds.

- The backup storage system consists of from 1 to 8 magnetic tape units, controlled by the PP which executes commands given it by the CP. Data is exchanged with AP memory at a rate of approximately 15,000 words per second.

- The communication I/O subsystem handles low and high speed lines, full and half duplex, synchronous and asynchronous. It is controlled by the PP which executes commands given it by the CP. Data is exchanged with CP memory in a byte mode, at rates from 10 to 1200 bytes per second depending on line speed. Note that all system I/O is communications oriented.

EQUIPMENT ORGANIZATION

The organization of LOGICON 2+2 System hardware to accomplish these functions is shown in Figure 2-2. This is the same as Figure 2-1, except that the Swapping Storage subsystem is shown to consist of a Drum Processor (DP) and a drum; and the File Storage, Backup Storage, and Communications I/O subsystems are shown to be all implemented with a single Peripheral Processor (PP) connected to the appropriate devices.

The characteristics of each of the devices are described in more detail in the following paragraphs.

Core Memory

There are two core memories in the system, CP and AP memory, named after the programmable processor that executes instructions from each. Each memory has a basic 900 nanosecond cycle time, and stores data in words of 16 bits plus parity.

CP Memory. CP memory can have sizes of from 8K to 32K words in 4K increments. It is accessed by the PP (port priority 1), AP (port priority 2), and by the CP (port priority 3). Read accesses to CP memory may result in a parity error, resulting in action as follows:

- AP access: Other memory parity error trap in AP.
- CP access: Own memory parity error trap in CP.
- PP access: Error interrupt to CP.

AP Memory. AP memory can have sizes of 32K or 48K four-way interleaved, or 65K with eight-way interleaving. Interleaving refers to

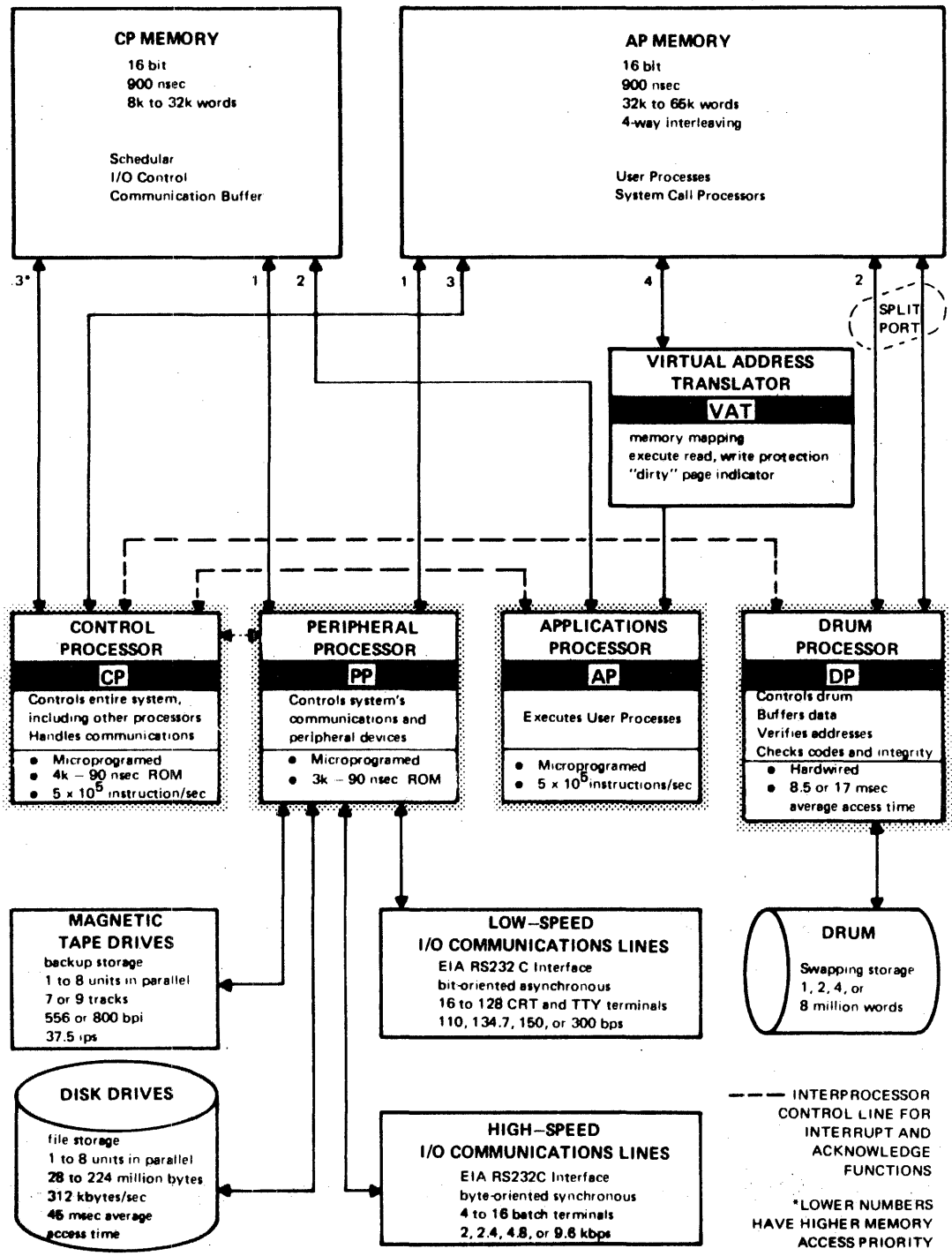


Figure 2-2. LOGICON 2+2 Hardware Organization and Information Flow

assigning addresses in such a way that consecutive locations are in separate banks of memory, so that accesses can be overlapped. Interleaving is important in AP memory, because the swapping storage subsystem makes accesses to consecutive locations at a rate of 1 million words per second, and must be able to overlap accesses in order to "catch up" after interference with other devices has occurred.

AP memory is accessed by the PP (port priority 1), DP (port priority 2), CP (port priority 3), and AP via the VAT (port priority 4). Read accesses to AP memory through ports 1 through 3 can result in parity errors, processed as follows:

- PP access: Error interrupt in CP.
- DP access: Error interrupt in CP.
- CP access: Other memory parity error trap in CP.

Accesses through the VAT are more complex, in that both parity and protection violations may occur. Separate software trap entrances are provided in the AP for own memory parity error, read protection violation, write protect violation, execute protect violation, and indirect address protect violation. The VAT protection mechanism is discussed in more detail later in this section.

Programmable Processors (CP and AP)

The CP and AP are programmable processors with nearly identical instruction repertoires, determined by microcode in a read-only control memory. They are high speed, general purpose, binary computers with single address instructions, each capable of communicating with two separate 900 nanosecond memories of up to 32K words (64K words with a VAT).

The design features include:

- A microprogrammed read-only control memory for flexible and economical internal logic.
- Highly modular integrated circuit construction with etched circuit back planes.
- Programmable memory protection and mapping via the VAT.

- Byte and byte string manipulation instructions for efficient handling of character information.

- Hardware floating point and multiple precision instructions for "number-crunching" applications.

- Hardware stack for efficient processing of recursive routines, multilevel interrupts, and nested system calls.

- Bit, Byte, word, and triple word addressing capability, all fully indexed.

- Real time clock, providing both a "time-of-day" and a countdown clock.

- Stall alarm for detection of program loops.

- Power fail-safe features in hardware and firmware.

The organization of these processors in terms of registers, stack operation, instruction repertoire, and I/O structure is covered in detail in Section III.

Virtual Address Translator (VAT)

The VAT is a program-controlled memory mapping and protection unit that can be inserted on any processor-to-memory access path. In the LOGICON 2+2 System it is used on the AP to AP-memory connection path.

The VAT considers memory to be organized in 512-word pages. It provides memory mapping, access control, and changed data detection for each of 64 pages in system mode virtual memory and each of 64 pages in user mode virtual memory. The two separate mapping and protection sections are called the System Map and User Map, respectively. When the AP is in system mode, all AP accesses to AP memory are made via the System Map. When in user mode, all accesses are made through the User Map (except for certain privileged instructions). The operation of the VAT is shown in Figure 2-3.

The VAT contains 128 registers, 64 for the system map and 64 for the user map. Each register contains the following data:

- A 7-bit physical memory page number

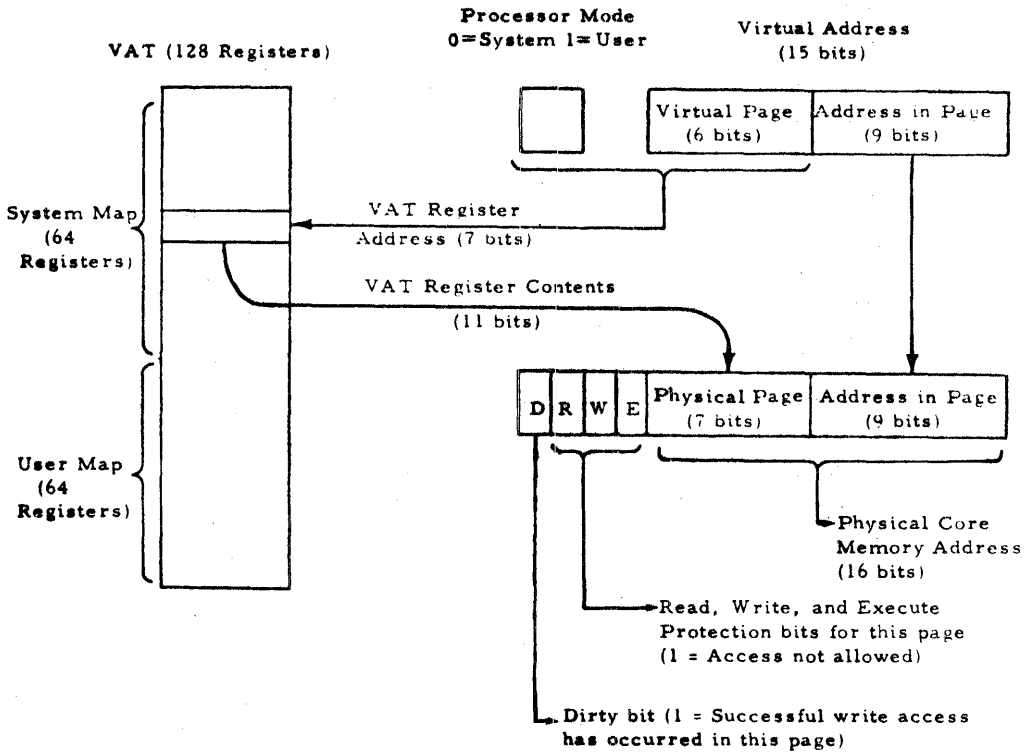


Figure 2-3. VAT Operation

- Read, write, and execute protection bits
- A "dirty" bit, or "data changed" flag.

The contents of the 128 registers may be set by software control, and the dirty bits may be interrogated by software. (See the Load Map, LDMAP, and Locate Leading Dirty Bit, LLDB, instructions.)

When an AP memory access is initiated from the AP, the VAT in conjunction with AP firmware performs the following functions:

- Selects the proper map depending on processor mode (system or user)
- Selects one of 64 registers in the proper map, as addressed by the high order 6 bits of the 15-bit virtual address from the processor.

- Checks protection bits in the selected register as follows:

  --Read access: if read protect bit is on, generate a read violation trap.

  --Write access: if write protect bit is on, generate a write violation trap.

  --Execute access: if execute protect bit is on, generate an execute protection trap.

  --Indirect address access: if both execute and read protect are on, generate an indirect address protection trap. This allows indirect addresses to be accessed in either read only or execute only pages.

- If no protection violation occurs, take the 7-bit physical page number from the selected VAT register, together with the nine least significant bits of the original virtual address, and initiate the appropriate memory access to the word addressed by the resulting 16-bit quantity.

- If a parity error occurs, generate an own memory parity error trap.

- Return the data read to the processor (read references only).

- Set the dirty bit for the page accessed (write reference only).

Memory references made through the VAT have a cycle time of approximately 1 microsecond, as opposed to the normal memory cycle time of 900 nanoseconds.

Drum Processor (DP)

The Drum Processor controls data exchanges between the AP memory (core) of the 2+2 System and the swapping memory (high data-rate drum).

Data Transfer Characteristics. All data transfers handled by the DP are 512-word pages. Associated with each page there is also an address word, a code word, and an integrity word, which assure data integrity and access control.

The characteristics and capabilities of the drum processor have been balanced to enhance overall system performance. The DP operates with a family of fixed-head drums with capacities ranging from 1/2 to 16 million 8-bit bytes. Average access time is 17 ms or 35 ms, depending on drum size.

Data transfer is carried out at sustained rates of up to 2 million bytes per second with burst mode transfers up to 4 million bytes per second. The controller features control word look-ahead to combine a contiguous sector transfer capability with list-driven command word chaining.

Control Characteristics. The DP communicates directly with the Control Processor (CP) by means of interrupts. This direct communication is initiated by the CP to cause the DP to examine its Current State Cell (CSC) Ø in core memory for a state change command.

The DP will send an interrupt to the CP when the DP has:

1. an interrupt condition encountered during data transfer.

2. an interrupt condition encountered during control word access.

Once initiated, via CP generated interrupt, the DP will examine the information contained in two predefined core locations in AP memory designated CSCØ and CSC1. Using this information, the DP then interprets its required mode of operation. When the RUN mode is designated, the CSC1 serves as a pointer to the core location of the initial Data Control Block (DCB) Ø-3. The DCB contains data transfer control information for a single data page. Thereafter, each DCB links with its successor by contiguous locations in a DCB list or by a special link pointer to a non-contiguous core location. The cycle continues until a Last DCB condition, or an interrupt condition capable of terminating the operation, is encountered.

The DP will always send an interrupt to the processor when it terminates operation. It may send interrupts at other times as defined by the DCB commands.

Peripheral Processor

The Peripheral Processor (PP) is a microprogrammed processor that controls disk, tape, and communication I/O functions, as specified by commands in CP memory. The three functions are effectively independent and can best be described separately.

Disk Control. The disk controller controls from 1 to 8 moving-head disk drives. The specifications for each disk drive are as follows:

Number of heads	20
Number of cylinders (positions per head)	200 (plus 3 spares)
Number of tracks	20 x 200 = 4000
Sectors/track	7
Total sectors	7 x 4000 = 28000
Seek time	10 ms track to track, 45 ms average 75 ms maximum.
Rotation Period	25 ms
Bit rate during transfer	2.5 MHZ
Word transfer rate	156.25 KHZ

The PP uses the same address verification, code word and integrity word checks as the DP to insure data protection and data validity.

Up to eight simultaneous seeks, or seven seeks and one simultaneous data transfer may be made. The software interface provides data and command chaining, with a single read or write command causing the transfer of precisely one page of data. Interrupts are sent to the CP when an error condition occurs, or when an interrupt is requested on command completion. Waiting interrupts are linked in a FIFO queue in CP memory.

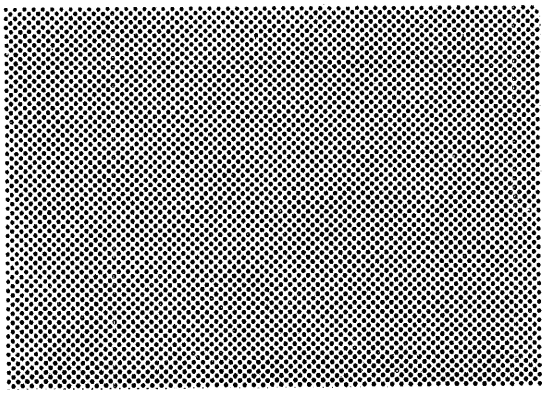
Tape Control. The tape controller controls from one to eight magnetic tape drives, in any mixture of seven and nine track varieties. Standard IBM compatible tape drives are used, with a nominal tape speed of 37.5 ips, for a word transfer rate of 15 KHZ at 800 bits per inch. One data transfer at a time may be performed.

The software interface provides data and command chaining, with scatter/gather within a single record if desired. Interrupts are sent to the CP when an error condition occurs or when an interrupt is requested on command completion. Waiting interrupts are linked in a FIFO queue in CP memory.

Communication Control. The communications input/output system controls up to 128 asynchronous and 16 synchronous lines. Asynchronous lines may be added in groups of 16; synchronous lines in groups of four.

Asynchronous lines are compatible with Bell 103A data sets at rates of 110, 134.7, 150, and 300 BAUD. Synchronous lines are compatible with 202 series data sets at rates of 2000, 2400, 4800, and 9600 BAUD.

The software interface with the CP provides both character and buffer mode, full and half duplex, with data chaining in buffer mode. Waiting interrupts are linked in a first-in first-out (FIFO) queue for servicing in the proper order by the CP.



III...

Processor Organization

This section describes the basic organization of the Application Processor (AP) and the Control Processor (CP) in the LOGICON 2+2 System. Both processors are essentially identical in their physical characteristics and the information given in this section applies equally to understanding the internal organization of either the AP or the CP as hardware items.

REGISTERS

The programmable registers in the processors are described in the following paragraphs. All registers contain 16 bits, although the quantities stored in some registers may normally utilize fewer bits.

A = Accumulator Register

The primary accumulator in the machine. Arithmetic, logical and shift operations are performed directly on this register. It may be linked with U to form a 32-bit accumulator UA. It may be linked with U and E to form a 48-bit accumulator UAE.

U = Upper Accumulator Register

Some arithmetic, logical, and shift operations are performed directly on this register. In other cases, it is linked with the A register to form a 32-bit accumulator UA or with A and E to form a 48-bit accumulator UAE.

E = Exponent Register

Contains the exponent in floating point operations. The exponent is expressed as a 2's complement number. This register can be loaded from memory or other registers. It has very limited arithmetic and logical capabilities. It can be linked with the U and A registers to form a 48-bit accumulator UAE.

X = Index Register

Indexing may be performed on 15-bit word addresses, 16-bit byte addresses, or 16-bit 2's complement displacements. Arithmetic operations on this register do not affect the overflow or carryout status indicators.

P = Program Counter Register

This register generally contains the address of the next instruction to be executed. In forming relative addresses in basic instructions it contains the address of the current instruction. The register is 16 bits long but the addresses it contains are all 15-bit quantities.

B = Base of Stack Register

This register contains the 15-bit address of the base of the stack as seen by the main program or subroutine currently running. Attempts to "pop" the stack beyond this address will result in a stack underflow trap. If the high order bit is set, erroneous results may result from tests on B.

T = Top of Stack Register

This register contains the 15-bit address of the next word to be pushed into the stack. This address should not be less than the address contained in B nor greater than that contained in L as the result of a stack operation. Checks are made before the stack operation. Note that checks are made only in the appropriate direction (i.e., check for overflow on a "push," and underflow on a "pop").

L = Limit of Stack Space Register

This register contains the 15-bit address of the first word beyond the stack (i.e., the address of the first word the stack is not allowed to occupy). An attempt to "push" the stack beyond this address results in a stack overflow trap.

S = Status Register

Bits in this register describe the current status of the machine. Bit positions within the word are defined in this section under the discussion of Status Register.

REPRESENTATION OF INFORMATION

The binary system of notation is used throughout the LOGICON 2+2 System.

In the "fixed-point arithmetic" case of addition, subtraction, and comparison, operands and results are considered as binary numbers in 2's complement form. Subtraction, for example, is carried out internally by adding the 2's complement of the subtrahend.

The assumed location of the binary point has significance only for multiplication and division. For integer arithmetic, the binary point may be assumed to the right of the least-significant bit position (i.e., to the right of bit position 15); and for fractional arithmetic, the position of the binary point may be assumed to the left of the most-significant position (i.e., between bit positions 0 and 1).

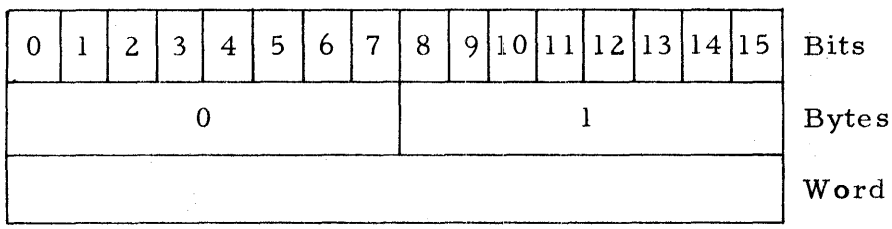
Floating point numbers are stored with the mantissa in absolute value and sign, and the exponent stored in two's complement.

The processor is fundamentally organized to deal with 16-bit grouping of information. Special features are also included for ease of manipulating bits, bytes, and multiple words as groups. These bit groupings are used by the hardware and software to represent a variety of forms of information.

Machine Word

The machine word consists of 16 bits. The numbering of bit position, character positions, words, etc. increases in the direction of conventional reading from the most-to-least significant.

Data transfers between processor and memory are bit, byte, and word oriented as illustrated below.



Alphanumeric Data

Alphanumeric data are represented by 8-bit bytes. One machine word contains two bytes or characters. The character set used for most purposes is standard ASCII.

One Word Binary Integers

For the "algebraic" group of instructions, results are regarded as signed binary numbers, the leftmost bit being used as a sign bit (a 0 being plus and 1 minus). When the sign is positive all the bits represent the absolute value of the number; and when the sign is negative, they represent the 2's complement of the absolute value of the number. Overflow occurs when the magnitude of a number does not fit within a given word or register. That is, if the carryout of the sign position does not agree with the resultant sign (bit position), overflow has occurred. There are no conditions for underflow. A signed integer ranges from -2^{15} through $2^{15} - 1$.

For the "logical" group of instructions, results are regarded as unsigned, positive binary numbers in the range of 0 through $2^{16} - 1$.

Three Word Binary Integers

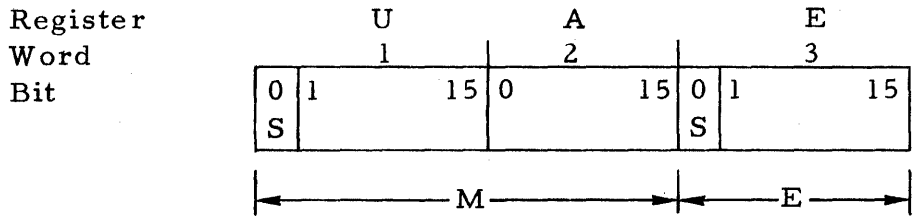
The three word integers are sign magnitude, the left most bit of the first word is sign followed by 47 bits of magnitude. The range of three extended integers is from $-(2^{47} - 1)$ through $2^{47} - 1 = 140, 737, 488, 355, 327$. Overflow occurs when the magnitude of a number does not fit within the 47 bits.

Three Word Binary Floating-Point Numbers

The instruction set contains instructions for binary floating-point arithmetic with numbers of two-word precision. The lower word represents the integral exponent E in 2's complement form, and the upper two words (32 bits) represent the fractional mantissa M in sign magnitude form. The notation for a floating-point number N is:

$$N = M \times 2^{E}$$

The three word format is shown below. S represents the sign bit.



Any number with an absolute value in the range of 10^{-9863} through 10^{9863} can be represented to more than nine significant decimal digits.

For normalized floating-point numbers, the binary point is placed at the left of the most significant bit of the mantissa. Numbers are normalized by shifting the mantissa (and adjusting the exponent) until no leading zeros are present in the mantissa.

To maintain accuracy, the lowest possible exponent (-32768) together with a zero mantissa has been defined as the machine representation of the number zero.

Four Word Binary Floating Point Numbers

These numbers are similar to the three word floating point with the exception of one more word of precision. This permits a number with an absolute value in the range of 10^{-9863} through 10^{9863} to be represented to more than 14 significant decimal digits. Arithmetic in this format is performed by software subroutines.

PROCESSOR FEATURES

Stack

The 2+2 processor includes three hardware registers that define a stack, or last-in first-out (LIFO) list structure. The registers are B (base of stack), T (top of stack) and L (limit of stack). The core storage locations making up the stack are (B) to (L-1) inclusive, and (T) is the address of the next word to be pushed into the stack.

Many 2+2 processor instructions push data into the stack (by storing data at the location in T and incrementing T) or pop data out of the stack (by decrementing T and reading data from the resulting location in T).

A subroutine call saves the current base of stack (B) value as well as (P), and resets B to the new value of T. This presents the subroutine with an empty stack insofar as any pushing or popping of entries is concerned. A subroutine return can then use this new value of B to restore the stack to the state it had at subroutine entry, even if T had not been restored to its initial setting.

System call, system return, trap entry, interrupt entry and interrupt return also use the stack mechanism, but reset B, T, and L to a special system stack while the system call, trap, or interrupt is being processed. When the processor is in user mode, the system stack is defined by a BASE location stored in memory location 4 of the system

address space, and a LIMIT location stored in location 5 of the system address space.

Skips and the Delayed Skip (DSK) Instruction

Many instructions allow conditional skips. All skips are skips of instructions rather than of words. Since instructions may be either one or two words long, skips are generated by reading up the next instruction and decoding it to determine whether it is a one-word instruction or the first word of a two-word instruction, then incrementing (P) by one or two.

CAUTION

> Do not skip data words! If a data word is skipped, and it happens to have the format of a two-word instruction, then the word following it will also be skipped. If that is a one-word instruction, it will be skipped; if it is a two-word instruction, its first word will be skipped and the machine will try to execute its address.

If reading up an instruction to be skipped results in an execute violation or a parity error, the corresponding trap processor is entered at its second word, as shown:

Trap Condition	Transfers Control to:
Execute Violation	(00032_8)
Execute Violation During Skip	$(00032_8)+1$
Parity Error	(00034_8)
Parity Error During Skip	$(00034_8)+1$

The Delayed Skip (DSK) instruction is used to force a skip, under program control. It is particularly useful for generating normal and exception returns from subroutines and system calls because it produces a true instruction skip, whereas incrementing the return address would produce a word skip. A DSK causes the instruction following the next instruction to be skipped; that is, the DSK does not affect the execution of the instruction immediately following it, but the instruction that would normally follow that one is skipped unconditionally. A DSK

can be used before any legal machine instruction except HLT, SCALL, CPRS or another DSK; if one is used preceding any of these instructions, it is ignored.

When a DSK is used before a conditional skip instruction – e.g., ACA, FCPS, MINC, SKXEI, etc. – the next instruction is skipped unconditionally, and if the skip condition was satisfied the one after that is also skipped. Thus, a DSK preceding a conditional skip instruction forces one extra skip. (This is the reason DSK is ignored before a CPRS instruction; the CPRS by itself can skip two instructions, and there is no provision in the processor for a skip of three instructions.)

When a DSK precedes an instruction that causes a transfer of control – e.g., JMP, JSPM, CALL, RTRN, SRTRN, etc. – the skip occurs after the jump, and the instruction jumped to is the one skipped. When a DSK precedes a conditional jump – e.g., JZE, XJP, TJP, etc. – the instruction skipped is either the one jumped to or else the one immediately following the jump instruction, depending on whether or not the jump condition is satisfied.

Traps and interrupts are essentially transparent to DSKs, even in those cases (e.g., floating point overflow and underflow) where the trap processor itself uses a DSK ahead of its SRTRN to skip over the instruction that caused the trap. If that instruction was preceded by a DSK, then the return will skip two instructions.

Processor Mode

The processor has two modes of operation, a System mode, used in Monitor programs, and a User mode, used by all programs running under control of the Monitor. The current mode of operation of the processor is indicated in bit 9 of the Status register (0 for system mode, 1 for user mode). The difference between modes and the ways of switching between modes are described in this section.

In System mode, all instructions including privileged instructions may be executed. On memory accesses made through the VAT, the system map is used. In User mode, an attempt to execute a privileged instruction will result in a privileged instruction execution trap. Memory references made through the VAT use the user map.

A system call (SCALL) instruction executed in User mode results in a transition to System mode. Furthermore, the user stack is "sealed off" and the stack registers are reset to use a system stack, defined

by locations stored in cells 4 (BASE) and 5 (LIMIT). Further system calls while in system mode continue to use the same system stack. As each system call is completed and control returned via a system return instruction, control eventually reverts to the original user mode code. At this point the processor status reverts to user mode, the system stack is abandoned, and use of the user stack resumed in the configuration it had at the time of the original system call.

An interrupt or trap occurring while in user mode similarly causes a transition to system mode, and to the system stack. The interrupt return or system return reverts to user mode and the user stack. Note that trap processing routines exit via the system return (SRTRN) instruction.

Status Register

The status register contains several items of information about the current state of the processor. Its format is as shown in Figure 3-1.

Fixed Point Carryout. This bit contains the carry or borrow from the high order (sign) bit position of the last fixed point add or subtract instruction affecting (A) or (U) executed, or zero if it has since been reset by a SKNCO (skip if no carry out) instruction.

Fixed Point Overflow. This bit contains a one if an arithmetic overflow has occurred since the last SKNOF (skip if no overflow) instruction. An overflow has occurred if the correct arithmetic result of an operation cannot be expressed in the register that the answer should go into. Note that overflow is not reset by arithmetic operations; it can only be set. Carryout may be either set or reset by arithmetic instructions.

Certain instructions are classified as being for primarily logical or address operations and do not effect overflow or carryout. The description of each individual instruction has this information. The general rules are as follows:

- Fixed point add and subtract operations on A and U, except for the interregister instructions RADD and RSUB, may affect carryout and overflow.

- Add and subtract to memory (ADDM, SUBM) may affect carryout and overflow, but MINC and MDEC do not.

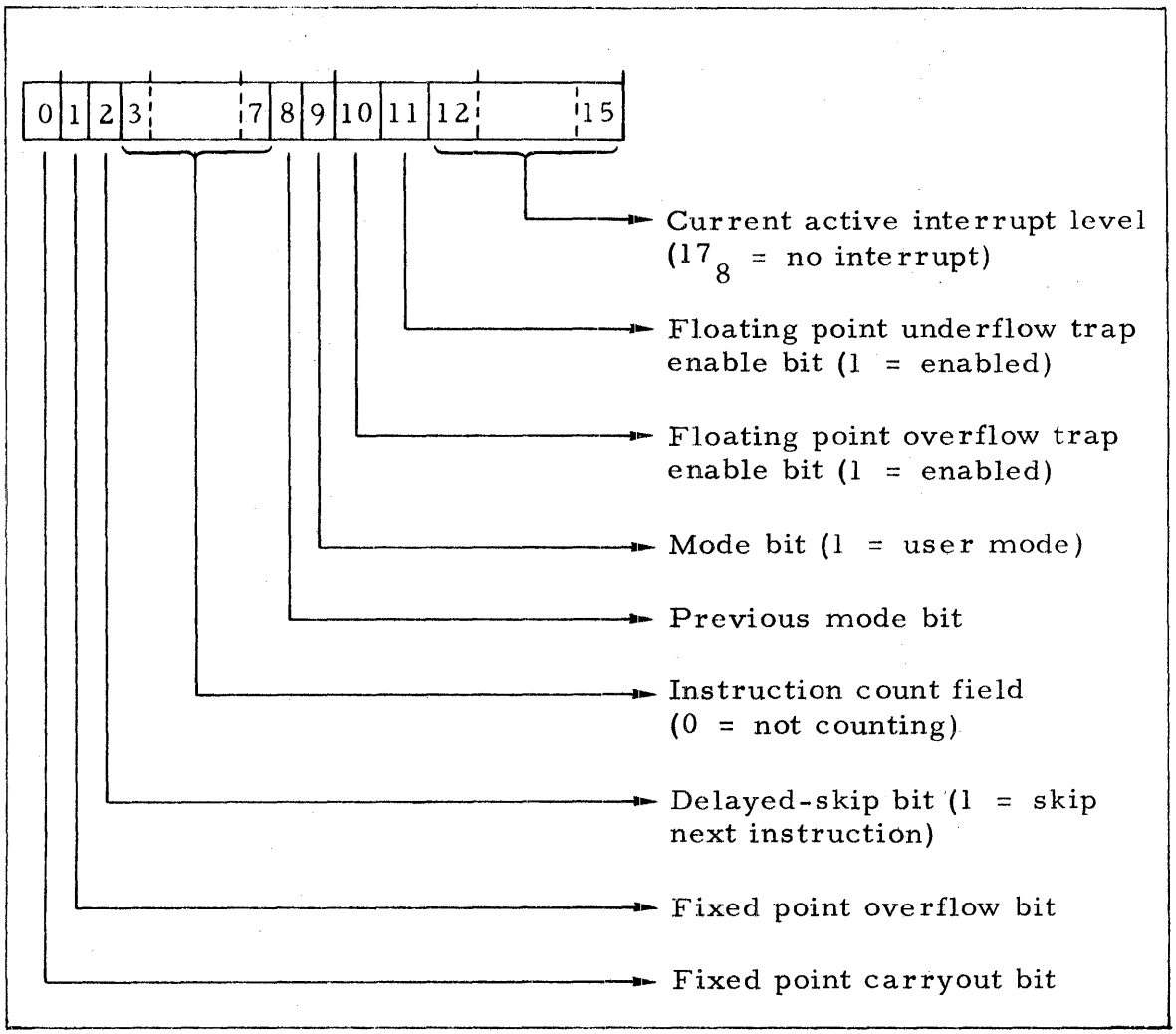


Figure 3-1. Status Register Contents

- Fixed point division may set overflow.
- Fixed point multiply (MPA AND MPAS, but not MPX) sets overflow if the results will not fit in one register (this is an exception to the general definition of overflow).
- Arithmetic left shifts may set overflow.

Delayed-Skip Bit. When this bit is set on completion of an instruction, the instruction that would have normally been executed next is skipped, and the following instruction executed. This feature is used, for example, to give skip returns to system call instructions. Refer to description of "Skips and the DSK Instruction".

Instruction Count Field. The processor may be placed in a "trap after executing n instructions" mode for $0 \leq n \leq 30$. This field of the status register contains a number one greater than the count of instructions remaining before the trap. A count of zero means no trap is pending. This feature will be used primarily for debugging operations.

The count may be started by a system routine, by inserting a non-zero count in the saved status register location in the system stack, and then executing a system return instruction. This returns control to the calling routine with the instruction count set in the status register. The calling routine may execute exactly the indicated number of instructions, and then an instruction count trap will occur.

The following special comments apply to the implementation of the count mechanism:

- A system call counts as one instruction, independent of the number of instruction executions it may invoke.
- Interrupt routines are not counted.
- Many trap conditions repeat the instruction on which the trap occurred, following execution of the trap processor. The instruction on which the trap occurred will be counted both on the aborted execution and the final execution. Trap processors must be written to correct for this condition.
- If a user program is swapped out while the count is in process, the count will be saved and resumed when that user is swapped in again. This feature is a property of the time sharing monitor and not of the processor hardware.

Mode and Previous Mode. Processor mode (0 = system, 1 = user) is stored in the status register. The previous mode bit is the state of the mode bit prior to the last system call, trap, or interrupt (i.e., in the routine that originally called the current system routine: not in a lower level routine that was called and returned control to the current routine).

Floating Point Trap Enable. Floating point overflow and underflow traps may be enabled and disabled by setting these bits. Note that they can be changed only by system code. When enabled, a floating point overflow (underflow) results in the corresponding trap.

A floating point overflow occurs when the result of a floating point operation is too large to be normalized (i.e.,

$$|R| > (1 - 2^{-31}) \times 2^{32767} \cong .708 \times 10^{9864}) .$$

A floating point underflow occurs when the result of a floating point operation is too small to be normalized (i.e.,

$$|R| < 1 \times 2^{-32769} \cong .353 \times 10^{-9864}) .$$

Current Interrupt Level. The current interrupt level is kept in the status register. It varies from 0 (highest priority) to 15 (lowest priority), with several states being effectively unused because of special hardware features. The state 15 is used to indicate "no active interrupt."

Traps, System Calls and Interrupts

Traps, system calls, and interrupts are very similar, the main differences between them being their causes. Traps are caused by the detection of internal error conditions, system calls are caused by the execution of SCALL instructions, and most interrupts are caused by external events which set bits in the external interrupt register. The exception is system stack overflow. This is an internal error, and as such ought to cause a trap, but to avoid conflicts it is made to cause an interrupt instead.

All system calls and most traps are always enabled, but two traps — floating point overflow and floating point underflow — can be enabled or disabled selectively by bits in the status register. When set to one, bit number 10 of the status register enables the floating point overflow trap, and bit number 11 enables the floating point underflow trap. Whether or not these traps are enabled, a floating point overflow always sets (U, A, E) to a very large floating point number (in octal: 077777, 000000, 077777), and a floating point underflow always sets it to a floating zero (in octal: 000000, 000000, 100000).

Most interrupts can be enabled or disabled selectively by using the SIM, SIL and RIL instructions. The SIL (Set Interrupt Lockout) instruction disables all interrupts except system stack overflow. The RIL (Reset Interrupt Lockout) instruction enables all interrupts allowed by the current interrupt mask. The interrupt mask is controlled

by the SIM (Set Interrupt Mask) instruction. At the beginning of each instruction, (before (P) is incremented) the interrupt mask is "anded" with the external interrupt register, and a check is made to see if there are any enabled interrupts, waiting to be honored, of higher priority than the current level as defined in the status register bits 12-15. The system stack overflow interrupt is, again, the exception. Since it is caused by an internal error rather than an external event it never appears in the external interrupt register, and therefore it cannot be disabled.

All traps and system calls, and all interrupts except the power-down interrupt, are processed as follows:

1. The VAT is set to system mode. This causes all subsequent memory references to use the system address space, regardless of the mode bit in the status register.

2. If the current status register mode bit indicates user mode, a transition to the system stack is made as follows:

   a. The contents of cells 00004 and 00005 are read out as the system stack BASE and LIMIT pointers, respectively.

   b. (B) is stored at the address given by (BASE), (T) is stored at (BASE)+1, and (L) is stored at (BASE)+2.

   c. B is set to (BASE), T is set to (BASE)+3, and L is set to (LIMIT).

   This procedure establishes a new system stack, which will be used for all traps, system calls and interrupts as long as the machine remains in system mode.

3. Registers are saved in the system stack as follows: The current status register (S) is stored at (T), (X) is stored at (T)+1, (U) is stored at (T)+2, (A) is stored at (T)+3, (E) is stored at (T)+4. If a trap or interrupt is being processed, (P) is stored at (T)+5; if a system call is being processed, (P)+1 is stored at (T)+5. Thus a trap or interrupt is set to return to the instruction where it occurred, and a system call is set to return to the next instruction. (B) is stored at (T)+6.

4. B and T are both set to (T)+7.

5. If an interrupt is being processed,

   a. Its priority level is saved as the current interrupt level.

   b. The current interrupt is acknowledged, thereby resetting the external interrupt line.

   c. The current interrupt level is stored in the low-order four bits (bits 12 - 15) of the status register, thereby disabling that level and all lower levels of interrupts.

6. The mode bit in the status register (bit number 9) is copied into the previous mode bit (bit number 8).

7. The status register is cleared except for the previous mode bit and the current interrupt level field (bits 12 through 15).

8. If a memory-related trap is being processed, X is set to the address where the error or violation occurred. In the AP, read write, execute and indirect reference violations, own-memory parity errors and indirect reference level overflows all produce mapped addresses in "specified map" form (i.e., the sign bit of (X) indicates which map was in use), but other-memory parity errors produce unmapped addresses. In the CP, all such traps produce unmapped addresses.

9. The appropriate dedicated cell is read and its contents are loaded into P. If a trap is being processed that resulted from an execute violation or own-memory parity error occurring during a skip, (P) is incremented by one.

10. If (T) > (L), (L) is incremented by 14 and a system stack overflow interrupt is generated; otherwise, normal execution resumes beginning at (P).

The dedicated transfer cell addresses for the various types of interrupts, traps and system calls are shown in Tables 3-1, 3-2, and 3-3.

TABLE 3-1. INTERRUPT ENTRY LOCATIONS
(IN SYSTEM ADDRESS SPACE)

Interrupt Level	Octal Location of Entry Address	Function
0	10	Power down interrupt entry
1	11	System stack overflow interrupt entry
2	12	Stall alarm interrupt entry (CP), Unused (AP)
3	13	Drum interrupt entry (CP), Unused (AP)
4	14	Disk interrupt entry (CP), Unused (AP)
5	15	Communications interrupt entry (CP), Unused (AP)
6	16	Magnetic tape interrupt entry (CP), Unused (AP)
7	17	Unused
8	20	Unused
9	21	Unused
10	22	Interprocessor interrupt #1 entry
11	23	Interprocessor interrupt #2 entry
12	24	Countdown clock interrupt entry

It is particularly important to note that the trap, system call and interrupt processor does not behave like a typical stack instruction (e.g., PUSHM) with respect to stack overflows. If a stack overflow occurs the words are pushed in anyway, then (L) is incremented by 14 and a system stack overflow interrupt is generated, which itself pushes in 7 more words. Accordingly, when the system stack LIMIT pointer (cell 00005) is initialized it must be set to an address at least 14 words less than the last word actually available in the system stack's storage space.

This is why a system stack overflow must generate an interrupt instead of a trap. If it did generate a trap, the software routine that processes it might get interrupted before it has had time to extend the

TABLE 3-2. TRAP ENTRY LOCATIONS (IN SYSTEM ADDRESS SPACE)

Trap	Octal Location of Entry Address	Notes
Read protection violation	00030	(X) = address. AP only.
Write protection violation	00031	(X) = address. AP only.
Execute protection violation	00032	(X) = address. AP only. Transfer is to (00032)+1 if detected during skip.
Indirect address protection violation (Both Read and Execute protected)	00033	(X) = address. AP only.
Own-memory parity error	00034	(X) = address. Transfer is to (00034)+1 if detected during skip.
Other-memory parity error	00035	(X) = other-memory address (unmapped).
Illegal instruction	00036	
Privileged instruction in user mode	00037	
Indirect reference level overflow (more than 8 levels)	00040	(X) = address of next level.
Stack overflow during user mode	00041	
Stack underflow during user mode	00042	
Floating point overflow	00043	(U, A, E) = 077777, 000000, 077777.

TABLE 3-2. TRAP ENTRY LOCATIONS (IN SYSTEM ADDRESS SPACE) (Cont)

Trap	Octal Location of Entry Address	Notes
Floating point underflow	00044	(U, A, E) = 000000, 000000, 100000.
Instruction counter countdown	00045	

TABLE 3-3. SYSTEM CALL ENTRY LOCATIONS (IN SYSTEM ADDRESS SPACE)

Processor	System Call Number		Octal Location of Entry Address
	Decimal	Octal	
CP	0	0	00050
	1	1	00051
	.	.	.
	.	.	.
	.	.	.
	23	27	00077
AP	0	0	00050
	1	1	00051
	.	.	.
	.	.	.
	.	.	.
	255	377	00447

NOTE: System call numbers 24-255 are trapped as illegal instructions in the CP.

stack. This would result in another stack overflow and another, nested, trap. By making system stack overflow a very high priority interrupt, all other interrupts can be locked out while it is being processed.

However, there is one interrupt, power down, that must be of higher priority even than system stack overflow if it is to be processed in time. Since it must be able to interrupt the system stack overflow

processor, it cannot use the stack; accordingly, a power down interrupt is processed differently than any other interrupt, as follows:

1. The map is set to system mode.

2. The contents of cell 00010, the dedicated address pointer for external interrupt line number 0, are read out.

3. (P) is stored at the address given by (10).

4. The current status register is stored at (10)+1.

5. The mode bit in the status register (bit number 9) is copied into the previous mode bit (bit number 8).

6. The status register is cleared except for the previous mode bit. By setting the active interrupt level to zero, this disables all interrupts.

7. P is set to (10)+2.

8. Normal execution resumes beginning at (P).

NOTE

The only registers saved by a power down interrupt are (P) and status. It is up to the interrupt routine to save (X), (U), (A), (E), (B), (T) and (L).

Except for floating point overflow and floating point underflow, all error conditions that result in traps (or interrupts, in the case of system stack overflows) are detected before any changes sufficient to prevent restarting the instruction are made to either registers or memory cells. Thus, in most cases where the cause of the trap is something that can be fixed, the trap processor can fix it and then restart the instruction. When this is done, the fact that a trap occurred is essentially invisible to the instruction.

Processing of trap conditions and system cells is terminated by a SRTRN (system return) instruction. Processing of interrupts is

terminated by an IRTRN (interrupt return) instruction. These instructions operate in very nearly the same way:

- The T register is restored to the value it had at entry to the trap, system call, or interrupt processing routine: (B)-7→T.

- All processor registers are restored from the stack:

  ((T))→S
  ((T)+1)→X [on SRTRN, only if XS bit set]
  ((T)+2)→U [on SRTRN, only if US bit set]
  ((T)+3)→A [on SRTRN, only if AS bit set]
  ((T)+4)→E [on SRTRN, only if ES bit set]
  ((T)+5)→P
  ((T)+6)→B

- Check for stack underflow:

  If (B) > (T), generate stack underflow trap

- Restore B, T, L to user mode stack if a transition to user mode was made: If the mode bit in the status register is now 1, then

  ((T)-3)→B
  ((T)-1)→L
  ((T)-2)→T.

- In IRTRN, enable all interrupt levels allowed by the current interrupt mask, up to but not including the interrupt level being returned to.

Clock

The processor has a one millisecond real-time clock, used to furnish both a time-of-day clock and a countdown clock for interval timing. The time-of-day clock is a double precision (32 bit) quantity stored in memory locations 00001 (most significant) and 00002 (least significant) in system mode address space. It is incremented once every millisecond. Thus, cell two counts in a cyclic pattern, repeating every 65,538 milliseconds, and cell one counts with a least significant bit of 65.538 seconds. The time-of-day clock may be set by software to any desired starting value.

A countdown clock, stored in cell 00003 of system mode address space, is decremented once every millisecond. When it reaches zero, an interrupt is set (priority level 12) to software. If not reset by software, it will supply an interrupt every 65.538 milliseconds (counting from 177777_8 to 0).

Stall Alarm

The processor may be equipped with a stall alarm feature. This device furnishes an interrupt (priority 2) approximately one second after the last reset signal it receives from software. Thus, if not reset for a period of more than one second, it furnishes an interrupt that may be used to detect inadvertent program loops.

In the LOGICON 2+2 System, the CP is equipped with a stall alarm — the AP is not. The stall alarm is reset by executing a DOUT instruction with X = 036000.

Programmer Panel (Prototype)

The prototype LOGICON 2+2 System uses two modified DSC model 4153 programmer's control panels as its programmer panel. After determination of requirements through use, these panels will be replaced with a single integrated system control panel. The physical layout of each of the panels is as shown in Figure 3-2.

One panel is connected to the Applications Processor (AP), and one to the Control Processor (CP). The lights and controls have the following functions:

- POWER: (CP only) Controls system power on/off.
- DISPLAY: Controls the display in LOCATION and DATA lights, and the interpretation of DATA switches, when the processor is halted.
- MODE: Determines the processor's response to an activation of the START switch.

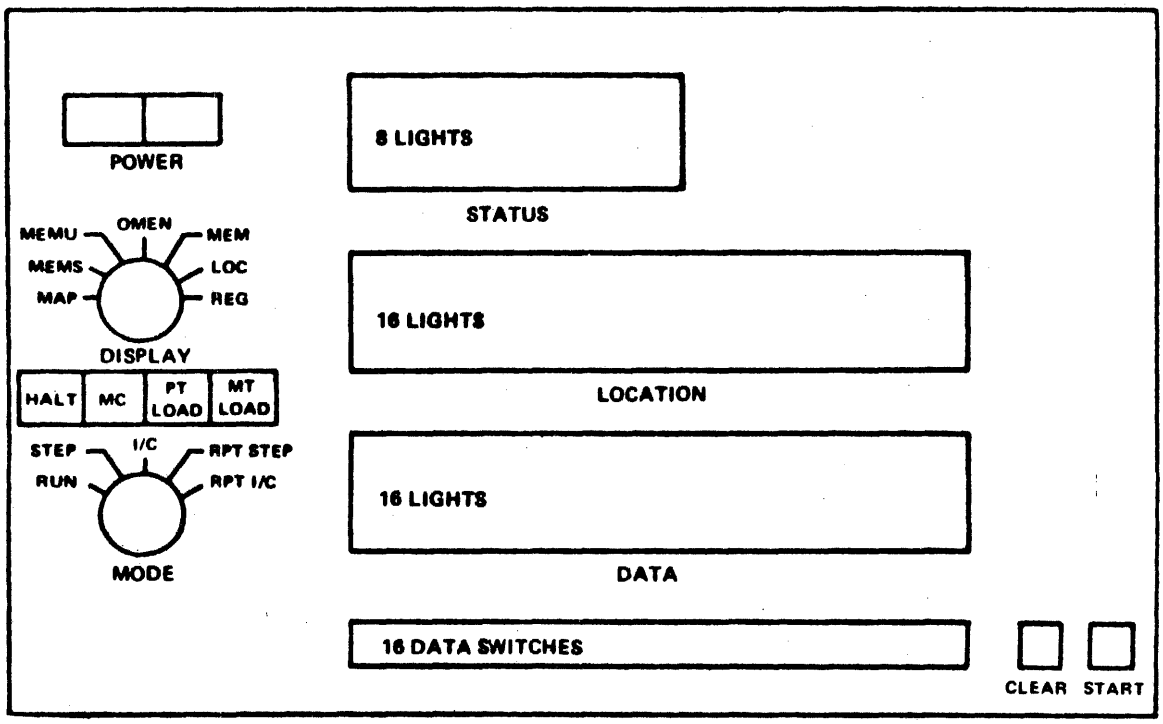


Figure 3-2. Programmer Panel Layout

- STATUS LIGHTS: Display processor status when in halt. Light meanings are (left to right):

HALT	USER MODE	PARITY	NOT USED	SS OF	SS UF	READ VIOL	WRITE VIOL

System Crash Conditions (Ref. HALT CONDITION)

- System stack overflow
- System stack underflow
- System read violation or parity error
- System write violation

- LOCATION Lights: Display a 16-bit number that tells, in conjunction with the DISPLAY switch, what is being displayed in DATA Lights. Used during halt only.

- DATA Lights: Display a 16-bit number, as selected by the DISPLAY switch and an internal address register (which is displayed in LOCATION Lights). Used during halt only.

- DATA switches: A bank of 16 switches used to enter data into DATA Lights and thence into internal registers if desired. Also sampled by software via the load A from console (LDAC) instruction. Off or 0 in center position, or on 1 in momentary down or latching up positions.

- CLEAR: Causes the register driving DATA Lights to be cleared, used in data entry during halt mode only.

- START: Causes an action determined by the MODE switch. Used during halt only.

- HALT: Generates an interrupt to the processor that causes it to enter the halt mode.

- MC: Sends a master clear to the I/O system, clears all internal registers (P, A, E, U, X, B, T, L, and STATUS).

- PT LOAD: Bootstrap load from paper tape, setting P to an address specified on tape. PARITY light activated if a checksum occurs. (CP only.) The format of a paper tape for Bootstrap load is shown in Appendix A.

- MT LOAD: Magnetic tape bootstrap, into page $\emptyset$ of system memory, setting P to contents of cell $\emptyset$. PARITY light activated if checksum error occurs. The format of a magnetic tape record for Bootstrap load is shown in Appendix A.

The data displayed in the LOCATION and DATA light registers is controlled by the DISPLAY switch as follows:

- MAP, display addressed map cell. (AP only)

- MEMS, display addressed cell in own memory using system map. (Mapped in AP, unmapped in CP)

- MEMU, display addressed cell in own memory using user map. (Mapped in AP, unmapped in CP)

- OMEN, display addressed cell in other memory.

- MEM, display addressed cell in own memory using map corresponding to processor mode. (Mapped in AP, unmapped in CP)

- LOC, display LOCATION register used in memory and map display.

- REG, display register selected by DATA switches:

BIT	REGISTER
0	P
1	X
2	U
3	A
4	E
5	B
6	T
7	L
8	STATUS
9	INTERRUPT MASK

Entry of data is controlled by the DATA and CLEAR switches. The displayed data is in DATA Lights, and the address of the data is in LOCATION Lights. In all display modes except REGISTER, the DATA and CLEAR switches may be used to modify the displayed data: The DATA switches are logically "ored" bit by bit into the DATA light register. All bit positions not held on by activated DATA switches may be cleared by the CLEAR switch.

An activation of the START switch when the MODE switch is in inspect and change (I/C) has the following effect on display:

- MAP or memory display:

  DATA Lights → addressed cell, increment address, display new address and contents

- LOC display

  DATA Lights → LOCATION Lights

- REG display

If LOCATION Lights contain single 1: complement LOCATION Lights. In the resulting mode, the DATA and CLEAR switches can modify the DATA Light register.

If LOCATION Lights contain single 0: DATA Lights → (Register indicated by LOCATION Lights); 0 → LOCATION Lights

An activation of the START switch in other MODE switch settings operates as follows:

- RUN: START causes the processor to enter the run mode, executing instructions starting at the location in P. Execution will continue until a HLT instruction is executed, the HALT switch is activated, or a system crash condition occurs.

- STEP: START causes the processor to execute the instruction at P, advance P to next instruction, and halt.

- I/C (Inspect and change): START causes display or change of the memory cell, map entry, or register selected via the DISPLAY switch.

- RPT STEP: causes the step mode to be repeated once per clock interrupt (1 millisecond) as long as START is depressed.

- RPT I/C: causes the I/C mode to be repeated at clock interrupt rate as long as START is depressed, or until a memory parity error is detected.

HALT Condition

The panel is operable only when the processor is in a HALT condition. When halted, the processor may be started by two methods:

- Place the MODE switch in RUN, and activate the START switch. The processor will execute instructions beginning at the address in P.

- An interprocessor interrupt at level 11 will cause the processor to start executing instructions beginning at the address in cell ∅ of the current modes' address space.

When running, the processor can be halted by one of the following means:

- Depressing the HALT switch on the panel,
- Executing a HLT instruction,
- Encountering a System Crash condition.

In the case of a SYSTEM CRASH condition, the processor does not immediately enter the halt condition: it lights a panel light indicating the condition, and hangs up in a short firmware loop. This allows firmware analysis via the microprogram panel. To cause the processor to enter the HALT condition from this state, activate the HALT switch.

The SYSTEM CRASH conditions are:

1. System Stack underflow.
2. System Stack overflow occurring on entry from user mode (when the system stack should be empty).
3. System Read Violation or parity error in dedicated location or in the system stack.
4. System Write Violation in a dedicated location or in the system stack.

Power Up/Down

The LOGICON 2+2 System is designed to respond automatically to and recover from power transients. The mechanism for accomplishing this involves hardware, firmware, and software. The sequence of operations implemented in hardware-firmware for power up and power down is as follows:

Power Up. The processor starts in a master cleared state, and checks to see if power came on as a result of a processor power switch activation. If so, it is assumed that this is a "cold start"; the processor enters the HALT state, awaiting operator intervention (normally a bootstrap load). If not, it is assumed that this is a recovery

from a transient power failure; the processor transfers control to the software instruction at the location contained in memory cell 0 (power up entry location).

The mechanism for determining whether power came on as a result of a start switch activation is logically as shown in Figure 3-3.

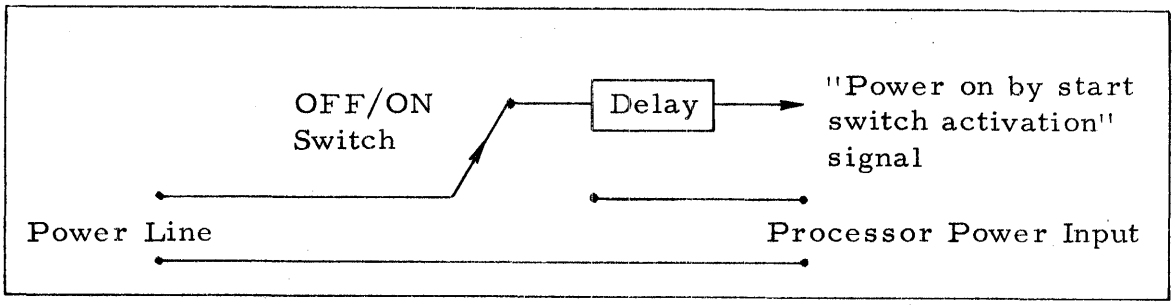


Figure 3-3. Processor Power Switch

Power Down. When power to the processor falls outside certain limits, an interrupt (priority level 0) is furnished to software at least five milliseconds before power goes outside safe operating limits. Thus software has five milliseconds to shut down in an orderly manner and save the state of the machine in core memory.

Data in memory is protected during power transients so that a full recovery may be made by properly-functioning software.

When power passes another threshold, instruction execution is terminated until power up. The power-up sequence has been previously described. It is possible for a power-down interrupt to occur, but for power to not actually go down. A software time-out routine can detect this condition. In this event, software must simulate the power-up recovery sequence.

DEDICATED MEMORY LOCATIONS

Certain memory locations in system mode address space are dedicated to special functions by hardware/firmware. These are summarized in Table 3-4.

TABLE 3-4. DEDICATED MEMORY LOCATIONS

OCTAL LOCATION	FUNCTION
0	Power up/Interprocessor Start entry
1	M. S. W. of clock
2	L. S. W. of clock
3	Countdown clock
4	System stack BASE pointer
5	System stack LIMIT pointer
6	Unused
7	Unused
10-24	Interrupt entry locations*
25	Unused
26, 27	Drum Processor CSC (AP only)
26	Unused (CP)
27	CP/PP interlock cell (CP only)
30-45	Trap entry locations**
46, 47	Unused
50-77	System call 0-23 entry locations
100-447	System call 24-255 entry locations (AP)

*In the LOGICON 2+2 System configuration, the interrupt entry locations are assigned as shown in Table 3-1.
**Trap entry locations are defined in Table 3-2.

INSTRUCTION FORMATS

Machine instructions are of 10 different format types. In addition, some of these formats may be subdivided into one or more sub-formats. Functionally, the formats are divided as follows:

1. Basic instructions.
2. Miscellaneous instructions.

3. System Calls.
4. Multi-register and register bit instructions.
5. Memory bit instructions.
6. Two-word general instructions.
7. Register operation instructions.
8. Single register shift instructions.
9. Multiple register shift instructions.
10. Immediate data instructions.

The specific formats (1 thru 10) are shown in Table 3-5, Instruction Formats. The symbols appearing in the Effective Address column of Table 3-5 are discussed in the following paragraphs under Addressing.

ADDRESSING

General

There are several modes of addressing used in the 2+2 processors. The procedure for calculating the effective address of an instruction is a function both of the instruction format and of the setting of several bits within the instruction.

Symbols

The following symbols are used in defining the addressing modes:

Y The effective address of an instruction: a 15-bit number if it is a word address, or a 16-bit number if it is a byte address.

D The signed (two's complement if negative) 8-bit displacement field in a basic (format 1) instruction. Its value is in the range $-128 \leq D \leq 127$.

P The program location register. P contains the 15-bit address of the first word of the instruction for which the address is being calculated.

X The index register. Generally effectively a 15-bit quantity, although the high order bit is used in certain cases.

TABLE 3-5. INSTRUCTION FORMATS

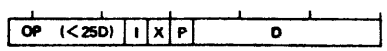
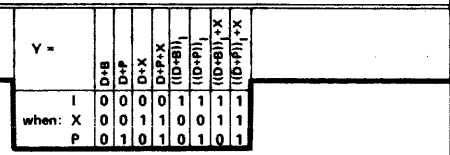
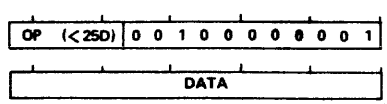
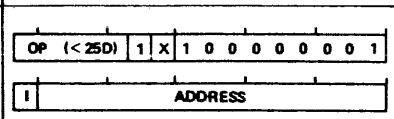
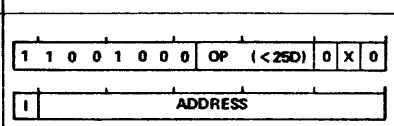
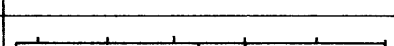
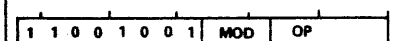
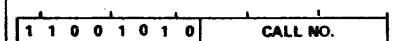
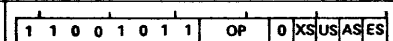
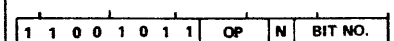
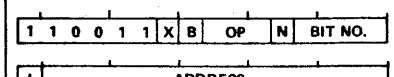
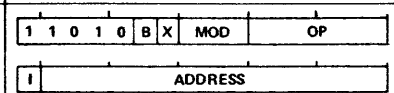
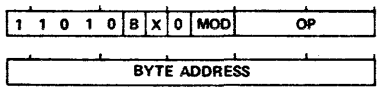
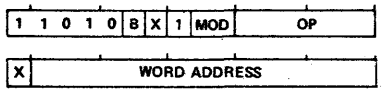
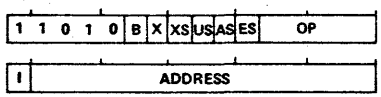
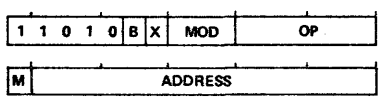
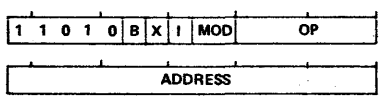
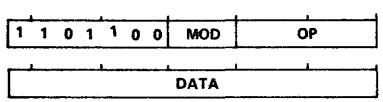
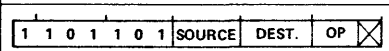
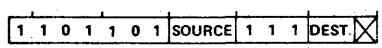
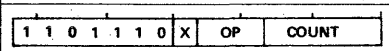
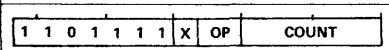
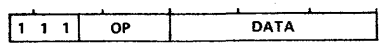
NUMBER	INSTRUCTION TYPE	FORMAT	EFFECTIVE ADDRESS	COMMENTS
1A	BASIC:			The value of Y depends on the settings of the I, X, and P designators.
1B	IMMEDIATE FORM OF BASIC			
1C	TWO-WORD FORM OF BASIC, WITHOUT B:		$Y = \left((P+1)\right)_I [+X]$	
1D	TWO-WORD FORM OF BASIC, WITH B:		$Y = \left((P+1)+B\right)_I [+X]$	
2A	MISCELLANEOUS:		Not Applicable	
2B	MISCELLANEOUS:		Not Applicable	
3	SYSTEM CALL:		Not Applicable	256 Calls allowed in AP 24 Calls allowed in CP
4A	MULTI-REGISTER:		Not Applicable	Registers selected by XS, US, AS, ES.
4B	REGISTER BIT:		Not Applicable	N is a bit number indexing flag: if N = 1, X is added to bit number.
5	MEMORY BIT:		if N = 0: $Y = (P+1)\ [+B]_I\ [+X]$ if N = 1: $Y = \left((P+1)\ [+B]\right)_I \left[+ \text{Integral part of } \frac{X + \text{Bit} =}{16}\right]$	Bit no. is given absolutely in instruction Bit no. is given by 4 low-order bits of X + Bit
6A	NORMAL TWO-WORD:		$Y = \left((P+1)\ [+B]\right)_I [+X]$	

TABLE 3-5. INSTRUCTION FORMATS (Cont)

NUMBER	INSTRUCTION TYPE	FORMAT	EFFECTIVE ADDRESS	COMMENTS
6B	TWO-WORD, DIRECT BYTE ADDRESS:		Y = (P + 1) [+B] [+X]	Note that a 16-bit byte Address results
6C	TWO-WORD, INDIRECT BYTE ADDRESS		Y = ((P + 1) [+B]) [+X]	Note that a 16-bit byte Address results. Only one level of indirect addressing is possible
6D	MULTIPLE LOAD AND STORE:		Y = ((P + 1) [+B])$_I$ [+X]	Registers selected by XS, US, AS, ES in that order are loaded from/stored in consecutive addresses
6E	"SPECIFIED MAP":		Y = (P + 1) [+B] [+X]	Note that a 16-bit result is obtained: if the most significant bit is 0, use the system map; if it is 1, use the user map.
6F	"OTHER MEMORY":		if I = 0: Y = (P + 1) [+B] [+X] if I = 1: Y = ((P + 1) [+B]) [+X]	Note that a 16-bit result is obtained. AP memory can go to 65K, and is addressed directly from the CP by these instructions. Only one level of indirect addressing is possible.
6G	TWO-WORD IMMEDIATE:		Not Applicable	
7A	OPERATE (EXC. RNEG):		Not Applicable	
7B	RNEG:		Not Applicable	
8	ONE-REGISTER SHIFT:		Not Applicable	
9	MULTIPLE REGISTER SHIFT:		Not Applicable	
10	IMMEDIATE		Not Applicable	

B The base of stack register. Always a 15-bit quantity.

(L) The 16-bit contents of the memory location with address L, mapped if made by the AP.

$(L)_I$ The result of running down an indirect address chain: the quantity L is interpreted as a 16-bit number. If the most significant bit is 0, then the remaining 15-bits constitute $(L)_I$. Otherwise the remaining 15 bits are used as a memory address to get a new 16-bit value for L, and the process is repeated. If more than eight memory references are made without reaching the end of the indirect reference chain (a number with a most significant bit of 0), then an indirect address trap is generated.

[+X] Add the value in the X register if and only if the X designator in the instruction is set (=1). Indexing is usually done after any indirect address references are completed.

[+B] Add the value in the B register if and only if the B designator in the instruction is set (=1). The contents of B are always added before starting an indirect reference chain. Note that the indirect address bit is tested before adding B, so that the indirect address bit is never affected by B.

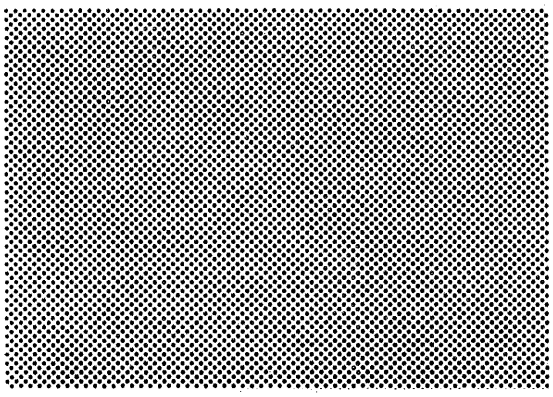
General Addressing Conventions

Basic Instructions. I, X, and P bits determine the use of the displacement field in addressing as follows:

- Indirect Bit (I). If equal to 1, addressed word is start of an indirect address chain.
- Index Bit (X). If equal to 1, X is added to effective address (after completing indirect reference).
- Relative to P Bit (P). If equal to 1, displacement D is added to program counter P. If equal to 0, displacement D is added to B (exception: I = 0, X = 1, P = 0 results in displacement being added to X).

Two-Word Instructions. In general, the contents of the second word of the instruction are conditionally added to B (if B bit is set), an indirect address chain is traced, and X is conditionally added (if X bit is set).

$$Y = \Big((P + 1)\,[+B]\Big)_I\,[+X]$$



IV...

Instruction Repertoire

GENERAL

In describing each instruction, five items may appear underlined preceding the instruction summary. The items are:

Mnemonic Name OpCode (Mod) Format

LOADS AND STORES

LDX Load X 00 1A, B, C, D

(y) → X

Load the contents of memory into the X register.
Modifiers: P, X, E, B, *, =
Refer to Table 4-1 for the allowable address modifiers (and legal combinations) for basic instructions under formats 1A, B, C, D.

1A formats. In general, these are used most frequently to load and store variables where the "address" is within the range of the displacement field, D. That is,

$$-128 \leq \text{address} \leq 127$$

is within the range of the current instruction.

1B formats. These are used, generally, to access constants. It is the closest thing to a literal in the 2+2 assembler. Literal pools, themselves, do not exist.

1C formats. This is the extended or two-word form of the basic instruction and is used to access address outside the range of the displacement field.

TABLE 4-1. ADDRESS MODIFIERS FOR BASIC INSTRUCTION FORMATS 1A, B, C, D

where,

* = Indirect Addressing
X = Indexing
P = Relative to P counter
B = Relative to B register
E = Extended address format (2 word form)

and,

- All other combinations of modifiers are illegal
- The modifiers may be specified in any order
- "A" may not be a literal but is the address of the word to be loaded

Format	Example	Address Code	Modifier Codes			Restrictions
			I	X	P	
1A	LDA A	P+D	0	0	1	D = A-P; -128 ≤ D ≤ 127
1A	LDA* A	(P+D)	1	0	1	D = A-P; -128 ≤ D ≤ 127
1A	LDA A,X	D+X or	0	1	0	if -128 ≤ A ≤ 127
		P+D+X	0	1	1	if A < -128 or A > 127
1A	LDA A,XP	P+D+X	0	1	1	D = A-P; -128 ≤ D ≤ 127
1A	LDA* A,X	(P+D)+X	1	1	1	D = A-P; -128 ≤ D ≤ 127
1A	LDA A,B	B+D	0	0	0	D = A-P; -128 ≤ D ≤ 127
1A	LDA* A,B	(B+D)	1	0	0	D = A-P; -128 ≤ D ≤ 127
1A	LDA* A,BX	(B+D)+X	1	1	0	D = A-P; -128 ≤ D ≤ 127
1B	LDA =A	P+1				
1C	LDA A,E	(P+1)				
1C	LDA* A,E	((P+1))				
1C	LDA A,EX	(P+1)+X				
1C	LDA* A,XE	((P+1))+X				
1D	LDA A,BE	B + (P + 1)				
1D	LDA* A,EB	(B + (P + 1))				
1D	LDA A,BXE	B + (P + 1) + X				
1D	LDA* A,EXB	(B + (P + 1)) + X				

1D format. The format is used to force address translation relative to the base of stack or B register. This is needed only if the displacement added to the contents of B is not in the -128 to +127 range, or if both B and X are to be added to the displacement.

LDXEA Load X with Effective Address 04(0) 6A

Y → X

Load the effective address of memory into the X register.

Modifiers: B, X, *

LDXI Load X, Immediate 00 10

LIT9 → X

The 9-bit literal contained in bit positions 7-15 of the instruction is loaded into the X register. The sign of the literal is extended through X_{0-6}.

Modifiers: None.

STX Store X 01 1A, C, D

(X) → y

Store the contents of the X register in memory location y.

Modifiers: P, X, B, E, *

XXM Exchange X and Memory 05(0) 6A

(y) → X; (X) → y

The contents of the X register and memory location y are exchanged.

Modifiers: B, X, *

LDU Load U 02 1A, B, C, D

(y) → U

Load the contents of memory into the U register.

Modifiers: P, X, E, B, *, =

LDUI LoadU, Immediate 01 10

LIT9 → U

The 9-bit literal contained in bit positioning 7-15 of the instruction is loaded into the U register. The sign of the literal is extended through U_{0-6}.

Modifiers: None.

STU Store U 03 1A, C, D

(U) → y

Store the contents of the U register in memory location y.

Modifiers: P, X, B, E, *

LDA Load A 04 1A, B, C, D

(y) → A

Load the contents of memory into the A register.

Modifiers: P, X, E, B, *, =

LDAEA Load A with Effective Address 04(2) 6A

y → A

Load the effective address of memory into the A register.

Modifiers: B, X, *

LDAI Load A, Immediate 02 10

LIT9 → A

The 9-bit literal contained in bit positions 7-15 of the instruction is loaded into the A register. The sign of the literal is extended through A_{0-6}.

Modifiers: None.

STA Store A 05 1A, C, D

(A) → y

Store the contents of the A register in memory location y.

Modifiers: P, X, B, E, *

XAM Exchange A and Memory 05(22) 6A

(y) → A; (A) → y

The contents of the A register and memory location y are exchanged.

Modifiers: B, X, *

LDE Load E 06 1A, B, C, D

(y) → E

Load the contents of memory into the E register.

Modifiers: P, X, B, E, *, =

LDEI Load E, Immediate 03 10

LIT9 → E

The 9-bit literal contained in bit positions 7-15 of the instruction is loaded into the E register. The sign of the literal is extended through E_{0-6}.

Modifiers: None.

STE Store E 07 1A, C, D

(E) → y

Store the contents of the E register in memory location y.

Modifiers: P, X, B, E, *

LDM Load Multiple 01, 41 6D

$(y, \ldots, y+n,\ 0 \leq n \leq 3) \rightarrow$ X, U, A, and/or E

The selected registers, X, U, A, and/or E, are loaded from the contents of memory locations y, y+1, . . . , y+n, where n is determined by the number of registers selected. The variable field of the instruction has three subfields: the selected registers, the memory address, and modifiers, if any. For example:

```
LDM  EU, A, X
```

The contents of A + X is loaded into the U register, and A + X + 1 is loaded into the E register. Registers are always loaded in the order X, U, A, and/or E, no matter how the order is specified in the symbolic instruction.

Modifiers: B, X, *

STM Store Multiple 02, 42 6D

(X, U, A, and/or E) $\rightarrow y, \ldots, y+n,\ 0 \leq n \leq 3$

The contents of the X, U, A, and/or E registers are stored in memory locations y, y+1, . . . , y+n, where n is determined by the number of registers selected.

The variable field for this instruction contains three subfields: the selected registers, the memory address, and modifiers; if any. For example:

```
STM   AX, A, B
```

The contents of the X and A registers are stored in memory locations B+A and B+A+1, respectively. Registers are always stored in the order X, U, A, and/or E, no matter how the order is specified in the symbolic instruction.

Modifiers: B, X, *

PUSHM Push Multiple 0 4A

(X, U, A, and/or E) $\rightarrow$ (T), . . . , (T) + n; $0 \leq n \leq 3$
(T) + n + 1 $\rightarrow$ T

The contents of the selected registers, X, U, A, and/or E are stored in consecutive locations defined by the contents of the top of stack pointer, T. T is then incremented by n + 1 so that the pointer is set to the next available word in the stack. Registers are pushed into the stack in the order X, U, A, and/or E, no matter how the order is specified in the symbolic instruction.

Stack overflow trap if (T) > (L).

Modifiers: None.

POPM Pop Multiple 1 4A

((T) - 1), . . . , ((T) - n - 1) → E, A, U, and/or X;
0 ≤ n ≤ 3; (T) - n - 1 → T

The selected registers, E, A, U, and/or X are loaded from the memory location specified by the top of stack pointer, T.

T is then decremented by n + 1 to reflect the next available word in the stack. Registers are popped from the stack into registers in the order E, A, U, and/or X, regardless of the order specified in the symbolic instruction.

Stack underflow trap if (T) < (B).

Modifiers: None.

PUSHN Push Null 06(0) 6A, G

(T) + (y) → T

The contents of the top of stack pointer, T, is incremented by the contents of the memory location y. There are two forms to the instruction:

6A format – Normal two word form

PUSHN A

The contents of A are added to the T register.

Modifiers: B, X, *

6G format – Immediate or literal form

PUSHN =A

The address or value A is added to the T register.

Modifiers: None.

Stack overflow trap if (T) > (L). Stack underflow trap if (T) < (B).

Modifiers: None.

"Specified map" instructions will be set to select the user map, which is the map in which the address is valid.

MSKM Mask Mode Bit 56(0) 2

The complement of the previous mode bit in the status register is logically "anded" with bit 0 of the X register, and the result left in bit 0 of the X register. That is, if the previous mode bit is 1, bit 0 of the X register is cleared. This is useful for passing addresses back from a system call to a calling routine: if the calling routine is system code, then bit 0 (map select bit) is left alone. If the calling routine is user code, then bit 0 (which is no longer map select in user code) is cleared.

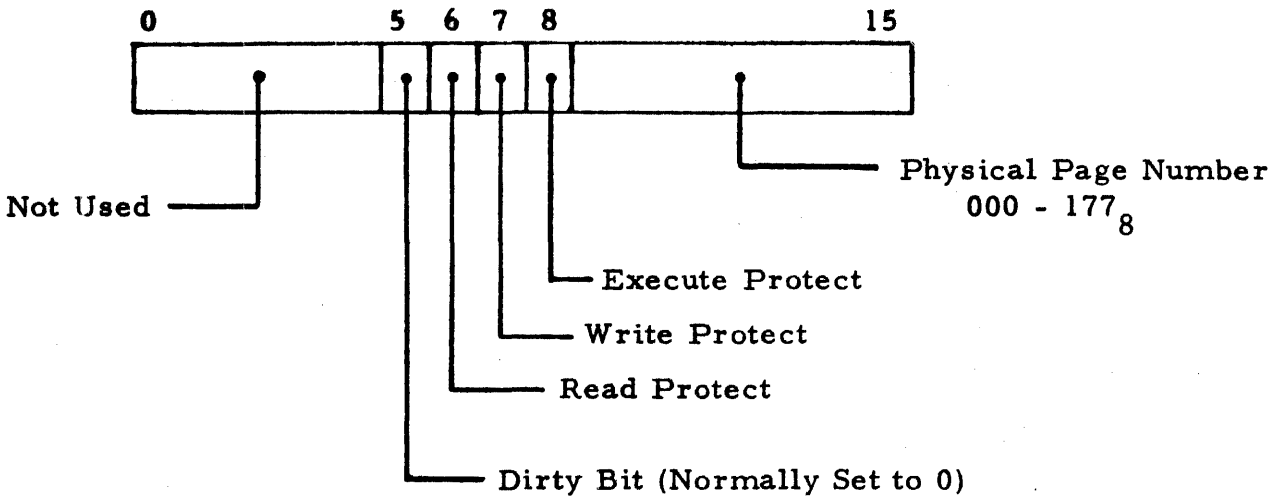
INPUT OUTPUT

LDAC Load A from Console Switches 57 2

The 16 data switches on the programmer's console are interrogated and their state placed in the A register.

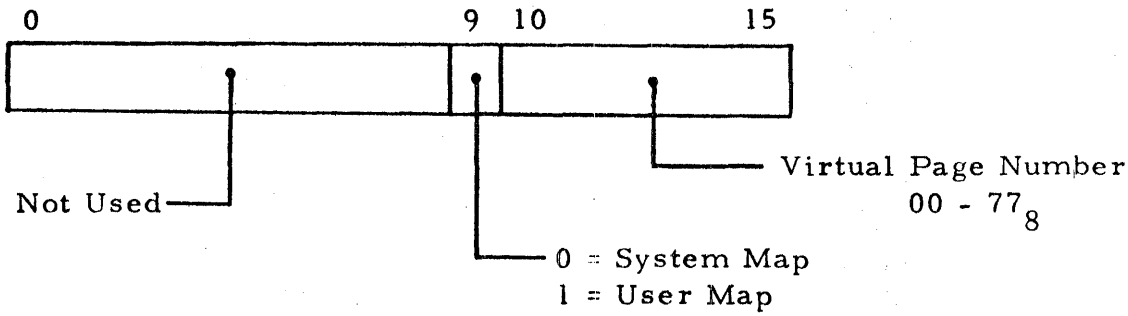
LDMAP Load Map 60 2

A number of consecutive map entries are set from consecutive core locations: the starting map page number is in A (00-77, system map pages 00 to 77; 100-177, user map pages 00 to 77), the starting core location is in X, and the number of map cells to be loaded is in U. The format of the core locations to be transferred to the map is as follows:



LLDB Locate Leading Dirty Bit 70 2

The map entries are inspected, beginning at the page number in X, for a "dirty" bit that is set. If one is found, the next instruction will be skipped and X will contain the page number of the page containing the dirty bit. If none is found, the next instruction will be executed with no skip. The format of the X register when a dirty bit is found is as follows:



SIM SET Interrupt Mask 75 6A, 6G

The operand is logically ANDed with a constant of 137777B (all 1's except for the system stack overflow interrupt mask bit) and placed in the software interrupt mask register. The firmware interrupt mask register is then loaded from the software mask down to, but not including, the bit number specified in bits 12-15 of the current status register. The system stack overflow interrupt is generated by firmware rather than by an external signal, so it is always enabled regardless of the contents of the mask registers.

DOUT Direct Output 75 2

The contents of the X register are placed on the I/O address lines. The contents of the A register are placed on the I/O data lines, and an I/O cycle is initiated. See descriptions of the I/O system for the address codes used to access the various I/O devices.

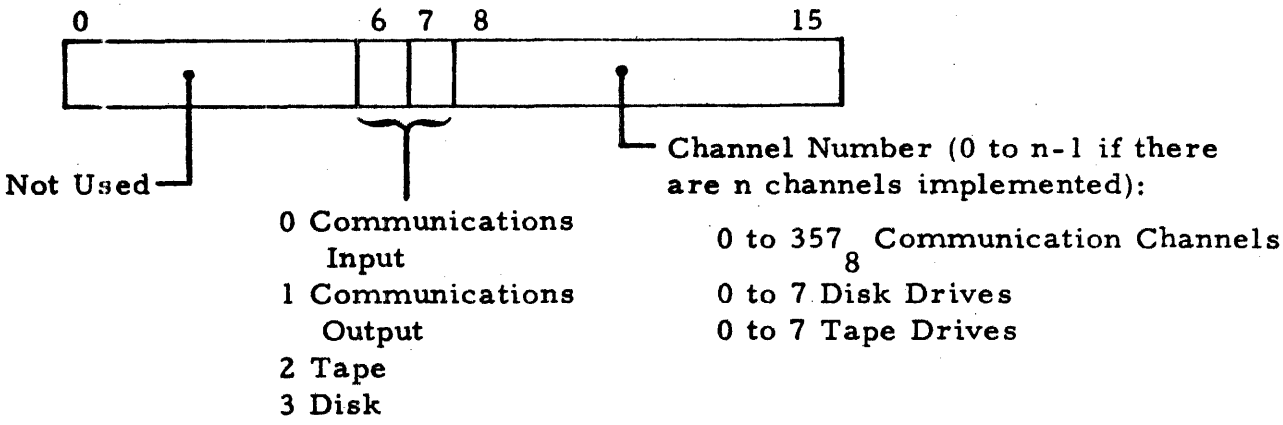
DIN Direct Input 74 2

The contents of the X register are placed on the I/O address lines. The I/O data lines are sampled after an appropriate delay and the data sampled is placed in the A register. See descriptions of the I/O system for the address codes used to access the various I/O devices.

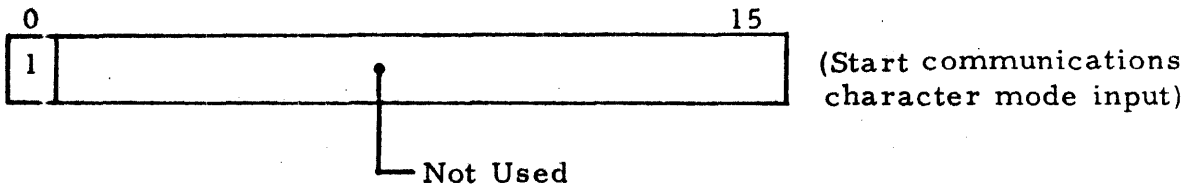
IOC Input/Output Control 71 2

This instruction is used in the Disk, Tape, and Communications I/O subsystems. A channel code is given in X and a function code in A, to control input or output to a tape unit, disk drive or communications channel. The register formats are:

X Register (Channel Code)



A Register (Function Code)



SIL Set Interrupt Lockout 72(0) 2

The firmware interrupt mask is set to zero, locking out all interrupts except system stack overflow. The software mask is unchanged.

RIL Release Interrupt Lockout 73(0) 2

The firmware interrupt mask is loaded from the software mask down to, but not including, the bit number specified in bits 12-15 of the current status register.

SRTRN System Return 2 4A

This instruction is used to return from system calls. It resets the status, program location counter, and stack pointers to the states they had when the system call was entered. It also restores any of the registers X, U, A, and E that are not used for passing parameters. If a return to user mode occurs, then the stack pointers are shifted back to the user stack. Symbolically,

(B)-7 → T
((T)) → S
((T)+1) → X if X flagged in instruction
((T)+2) → U if U flagged in instruction
((T)+3) → A if A flagged in instruction
((T)+4) → E if E flagged in instruction
((T)+5) → P
((T)+6) → B
If (B)>(T), stack underflow trap

If mode is now user (after S is restored), then:

((T)-3) → B
((T)-1) → L
((T)-2) → T

The instruction count mechanism may be activated by setting ((B)-7) properly before executing SRTRN.

IRTRN Interrupt Return 64 2

Return from an interrupt routine, restoring registers to the state they had when the interrupt became active:

(B)-7 → T

((T)) → S

((T)+1) → X

((T)+2) → U

((T)+3) → A

((T)+4) → E

((T)+5) → P

((T)+6) → B

If (B)>(T), stack underflow trap

If mode is now user (after S is restored), then:

((T)-3) → B

((T)-1) → L

((T)-2) → T

The firmware interrupt mask is loaded from the software interrupt mask down to but not including the bit number specified in bits 12-15 of the restored status register. This enables all interrupts of higher priority (lower number) than the one to which the return is made.

HLT Halt 77 2

The processor enters the halt mode, lights the HALT status light on the control panel, and stops executing instructions. The programmers control panel is enabled while the processor is in the halt mode.

CHARACTER INSTRUCTIONS

LDC Load Character 64(0) 6B, C

$(y_B) \rightarrow A_{8-15}$; $0 \rightarrow A_{0-7}$

Load the contents of byte location y_B into bit positions 8-15 of the A register. Bit positions 0-7 of A are set to zero.

Modifiers: B, X, *

STC Store Character 64(1) 6B, C

$(A_{8-15}) \rightarrow y_B$

Store bit positions 8-15 of the A register into byte location y_B. The A register is unchanged.

Modifiers: B, X, *

CPRS Compare Strings 052 2

Two byte strings in memory are compared. The byte addresses of the first character of each string must initially be contained in the X and A registers. The number of characters to be compared must be contained in the U register.

A simple ASCII comparison is performed, character by character. Hence, "G" is > "F", and "5" is > "4".

If the string designated in the A register > string in the X register, the next sequential instruction is executed.

If the string designated in the A register = string in the X register, the next sequential instruction is skipped, and execution continues with the following instruction.

If the string designated in the A register < string in the X register, the next two sequential instructions are skipped, and execution continues with the following instruction.

If an equal compare is made, the contents of the X and A registers point one character beyond the last character compared. If an unequal compare is made, the contents of the X and A registers point to the characters found to be unequal.

The CPRS instruction is interruptable and may be restarted.

Modifiers: None.

GFC Get First Character 65(0) 6A

$((y)_B) \rightarrow A_{8-15}$; $0 \rightarrow A_{0-7}$

Memory location y contains a byte address used to access a string. The instruction loads the contents of the specified byte address into bit positions 8-15 of the A register. Bit positions 0-7 of the A register are set to zero.

The byte address referred to above is interpreted as a string pointer. A string is thought of as being defined by two string pointers: a left pointer (LP), and a right pointer (RP). For purposes of utilizing the character instructions, these pointers are thought of as occurring in pairs, left and right, respectively. The pointers are described in more detail in the subsequent discussion of the GFCT instruction. The reader is referred to this section for further explanation.

The GFC instruction simply loads one byte of a string into the A register. No modification of the string pointers occurs. Therefore, repeated execution of a GFC instruction results in repeatedly loading the same byte.

Modifiers: B, X, *

GFCT Get First Character with Test 65(1) 6A

String is tested for null; If not null, $((y)_B) \rightarrow A_{8-15}$; $0 \rightarrow A_{0-7}$

Assume that the memory word pair BA and BA + 1 are memory locations containing byte addresses for two string pointers – the left pointer and right pointer, respectively,

BA	LP
+1	RP

Both the left and right pointers (LP and RP) are 16 bit byte addresses. The left pointer indicates the first byte of the string. The right pointer is set at the last byte of the string plus one.

The length of a designated string is always defined as RP-LP. A string is defined as null if LP ≥ RP. That is, if the left pointer has caught up with or passed beyond the right pointer. All "get" and "insert" character instructions access and modify strings via the left and right string pointers.

The instruction GFCT executes in the following manner. First the string pointers indicated at memory locations y and y + 1 are tested for a null string. If the left pointer is greater than or equal to the right pointer (LP ≥ RP), the string is null, and execution continues with the next sequential instruction. The contents of the A register are unchanged.

If the string is not null, the contents of the byte address specified in memory location y are loaded into bit positions 8-15 of the A register. Bit positions 0-7 of A are set to zero. The next sequential instruction is skipped and execution continues with the following instruction.

Modifiers: B, X, *

GCI Get Character and Increment 65(2) 6A

$((y)_B) \rightarrow A_{8-15}$; $0 \rightarrow A_{0-7}$; $(y) + 1 \rightarrow y$

Memory location y contains a byte address used to access a string. The instruction loads the contents of the specified byte address into bit positions 8-15 of the A register. Bit positions 0-7 of A are set to zero, and the byte address is incremented by one.

Modifiers: B, X, *

GCIT Get Character and Increment with Test 65(3) 6A

String is tested for null; If not null, $((y)_B) \rightarrow A_{8-15}$; $0 \rightarrow A_{0-7}$;

$(y) + 1 \rightarrow y$

The string pointers indicated at memory locations y and y + 1 are tested for a null string. If the left pointer is greater than or equal to the right pointer (LP ≥ RP), the string is null and execution resumes at the next sequential instruction. The contents of A are unchanged, and the left pointer is not incremented.

If the string is not null, the contents of the byte address specified in memory location y are loaded into bit positions 8-15 of the A register.

Bit position 0-7 of A are set to zero, and the byte address left pointer is incremented by one. The next sequential instruction is skipped, and execution resumes with the following instruction.

Modifiers: B, X, *

IFC Insert First Character 65(4) 6A

$(A_{8-15}) \rightarrow (y)_B$

The contents of bit positions 8-15 of the A register replace the contents of the byte address referred to by the contents of memory location y of the instruction.

The byte in the A register is placed in the byte address defined by the left pointer of the string. This instruction may develop a null string since no test concerning the right pointer is made.

Modifiers: B, X, *

IFCT Insert First Character with Test 65(5) 6A

String is tested for null; If not null, $(A_{8-15}) \rightarrow (y)_B$

The string pointers indicated at memory location y and y + 1 are tested for a null string. If the left pointer is greater than or equal to the right pointer ($LP \geq RP$), the string is null and execution resumes at the next sequential instruction. The byte specified by the left pointer is unchanged.

If the string is not null, the contents of bit positions 8-15 of the A register replace the contents of the byte address referred to by the contents of memory location y of the instruction. The next sequential instruction is skipped and execution resumes with the following instruction.

Modifiers: B, X, *

ICI Insert Character and Increment 65(6) 6A

$(A_{8-15}) \rightarrow (y)_B$; $(y) + 1 \rightarrow y$

The contents of bit positions 8-15 of the A register replace the contents of the byte address referred to by the contents of memory location y of the instruction. The byte address (left pointer) is incremented by one.

This instruction may develop a null string since no test concerning the right pointer is made.

Modifiers: B, X, *

ICIT Insert Character and Increment, with Test 65(7) 6A

String is tested for null; if not null, $(A_{8-15}) \rightarrow (y)_B$; $(y) + 1 \rightarrow y$

The string pointers indicated at memory locations y and y + 1 are tested for a null string. If the left pointer is greater than or equal to the right pointer (LP ≥ RP), the string is null and execution continues with the next sequential instruction. The byte specified by the left pointer is unchanged, and the left pointer is not incremented.

If the string is not null, the contents of bit positions 8-15 of the A register replace the contents of the byte address referred to by the contents of memory location y of the instruction. The byte address (left pointer) is incremented by one. The next sequential instruction is skipped and execution continues with the following instruction.

Modifiers: B, X, *

PRIVILEGED INSTRUCTIONS

The machine operates in either system mode or user mode. The system mode is the basic operating mode of the computer. In this mode, all legal operations are permissible. It is assumed that there is a resident monitor that controls and supports the operation of all other programs.

The user mode is the normal problem-solving mode of the computer. In this mode, certain privileged instructions are prohibited. Privileged instructions are those relating to input/output and to changes in the basic control state of the computer. Any attempt by a program to execute a privileged instruction while the computer is in the user mode results in a trap that returns control to the resident monitor. This unconditionally aborts execution of the instruction and may result in aborting the job or program.

A user program cannot directly change the computer mode from user to system. However, the user program can gain direct access to certain privileged program operations by means of the System Call instructions. The operations available through System Calls are established by the resident monitor.

LDAOM Load A from Other Memory 74(0) 6F

The addressed cell in "other memory" is loaded into the A register. If executed in the AP, the addressed cell in CP memory will be obtained. If executed in the CP, the (unmapped) addressed cell in AP memory will be obtained. No protection violation is possible in either case. One level of indirect addressing through own, not other, memory is allowed.

STAOM Store A in Other Memory 74(2) 6F

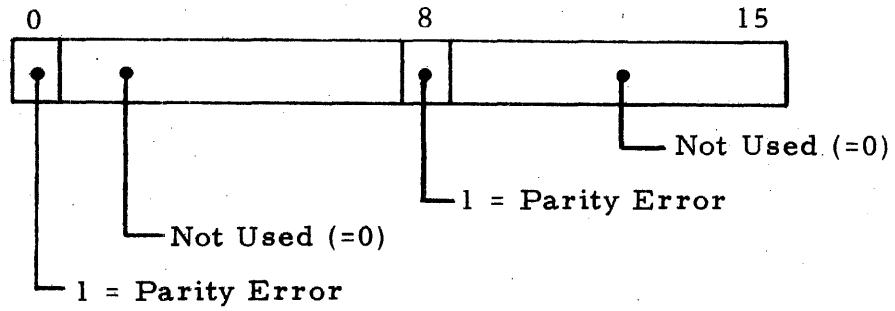
The contents of the A register are stored in the addressed cell in "other memory." If executed in the AP, the contents of A are stored in CP memory. If executed in the CP, the contents of A are stored in AP memory (unmapped). No protection violation is possible. One level of indirect addressing through own, not other, memory is allowed.

TSLOM Test and Set Lock in Other Memory 74(3) 6F

The contents of the addressed cell in other memory are set to 0; if the previous contents of bit 15 of the addressed cell in other memory were 1, skip. Otherwise take a normal return. This instruction is used to interlock critical areas of code between processors. Note that the address is unmapped and that no protection violation is possible. One level of indirect addressing through own, not other, memory is allowed.

LDAOMF Load A From Other Memory With Force 74(1) 6F

The addressed cell in other memory (as in LDAOM) is loaded into the A register. No parity trap is permitted. The contents of the memory status register at the completion of the memory reference are loaded into the U register with the format



One level of indirect addressing through own, not other, memory is allowed.

LDASM Load A through Specified Map 73(0) 6E

The addressed cell is loaded into the A register, using bit 0 of the final address as a "map select" bit. Bit #0=0 means use system map, bit #0=1 means use user map. (AP only)

STASM Store A through Specified Map 73(2) 6E

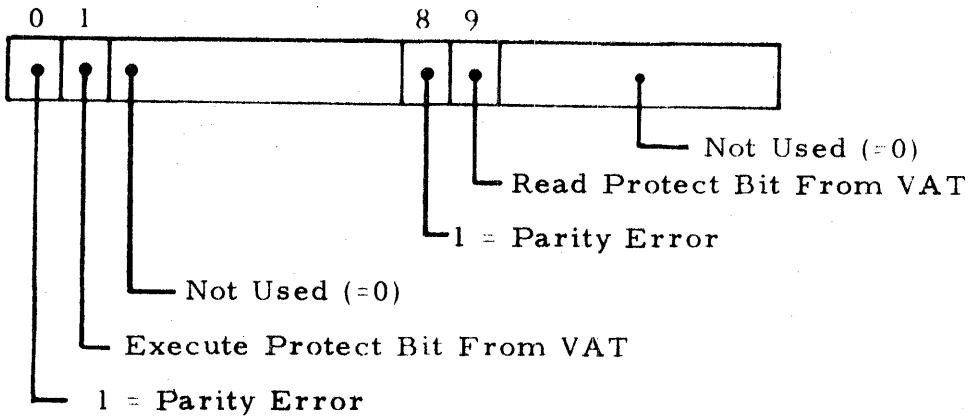
The contents of the A register are stored in the addressed cell, using the specified map as in LDASM. (AP only)

LDXSM Load X through Specified Map 73(3) 6E

The addressed cell is loaded into the X register using the specified map as in LDASM. Then the "specified map" bit is logically "ored" with bit 0 of the X register. This instruction is used for referencing addresses: the address obtained will have a "specified map" bit appended that specifies the map used to read the address. Thus the address will be interpreted through the map through which it was addressed. (AP only)

LDASMF Load A through Specified Map with Force 73(1) 6E

The contents of the addressed cell in AP memory are loaded into the A register using the specified map as in LDASM. No parity or protection traps are allowed. The contents of the memory status register after memory reference are loaded into the U register with the format



MRGM Merge Mode Bits 55(0) 2

The "previous mode" bit of the computer's status register is logically "ored" with bit 0 of the X register, and the result left in bit 0 of the X register. This is useful for passing parameter addresses from one system call to another: if the parameter address came from user code, then the high order bit (map select bit in

POPN Pop Null 06(1) 6A, G

(T) - (y) → T

The contents of the top of stack pointer, T, is decremented by the contents of memory location y. There are two forms to the instruction:

6A format – Normal two word form

POPN A

The contents of A are subtracted from the T register.

Modifiers: B, X, *

6G format – Immediate or literal form

POPN = A

The address or value A is subtracted from the T register.

Modifiers: None.

Stack overflow trap if (T) (L).
Stack underflow trap if (T) (B).

LDB Load B 07 6A, G

(y) → B

The contents of memory are loaded into the base of stack pointer B. There are two forms to the instruction:

6A format – Normal two word form

LDB A

The contents of A are loaded into the B register.

Modifiers: B, X, *

6G format – Immediate or literal form

LDB = A

The address or value A is loaded into the B register.

Modifiers: None.

STB Store B 10(0) 6A

(B) → y

Store the contents of the B register in memory location y.

Modifiers: B, X, *

LDSP Load Stack Pointers 11(0) 6A

(y, y + 1, y + 2) → B, T, L

Load the contents of memory locations y, y + 1, and y + 2 into the stack pointers B, T, and L, respectively.

Stack overflow trap if (T) > (L)
Stack underflow trap if (T) < (B)

Modifiers: B, X, *

LDBTL Load B, T, and L - 11(1) 6A

This instruction is the same as LDSP except that no stack overflow or underflow checks are made.

STSP Store Stack Pointers 10(1) 6A

(B, T, L) → y, y + 1, y + 2

Store the contents of the B, T, and L registers in memory locations y, y + 1, and y + 2.

Modifiers: B, X, *

STZ Store Zeros 12(n) 6A

$0 \rightarrow y, \ldots, y + n - 1; \; 1 \leq n \leq 8$

Words of zeros are placed in memory locations y, . . . , y + n - 1.

The variable field for this instruction contains three subfields: the number n (absolute expression), the beginning memory address, and modifiers, if any.

Modifiers: B, X, *

LSABM Load Sign of A from Bit in Memory 1 5

$(y_i) \rightarrow A_o$; (A_{1-15}) unchanged

where i is a designated bit number.

The sign position of the A register is loaded from the bit position of memory location y, designated by the bit number in the variable field of the instruction.

The variable field of the instruction has three subfields: the bit number (absolute expression), the memory address, and modifiers, if any.

The rules for modifiers X, and N are the same as those defined for the SETBM instruction.

Modifiers: X, B, N, *

SSABM Store Sign of A in Bit in Memory 2 5

$(A_o) \rightarrow y_i$

where i is a designated bit number.

The sign position of the A register is stored in the bit position of memory location y, designated by the bit number in the variable field of the instruction.

The variable field of the instruction contains three subfields: the bit number (absolute expression), the memory address, and modifiers, if any.

The rules for modifiers X, and N are the same as those defined for the SETBM instruction.

Modifiers: X, B, N, *

MOVE Move Word String 003 2

Move (U) words from (X) to (A)

N words, specified in the U register, are moved from a source memory location, specified in the X register, to a destination memory location, specified in the A register. The instruction may be interrupted and restarted without affecting its execution.

Modifiers: None.

CLX Clear X 00 10

This instruction is the same as LDXI 0

CLU Clear U 01 10

This instruction is the same as LDUI 0

CLA Clear A 02 10

This instruction is the same as LDAI 0

CLE Clear E 03 10

This instruction is the same as LDEI 0

LDF Load Floating Point Registers 41(3) 60

This instruction is the same as LDM UAE.

STF Store Floating Point Registers 42(3)

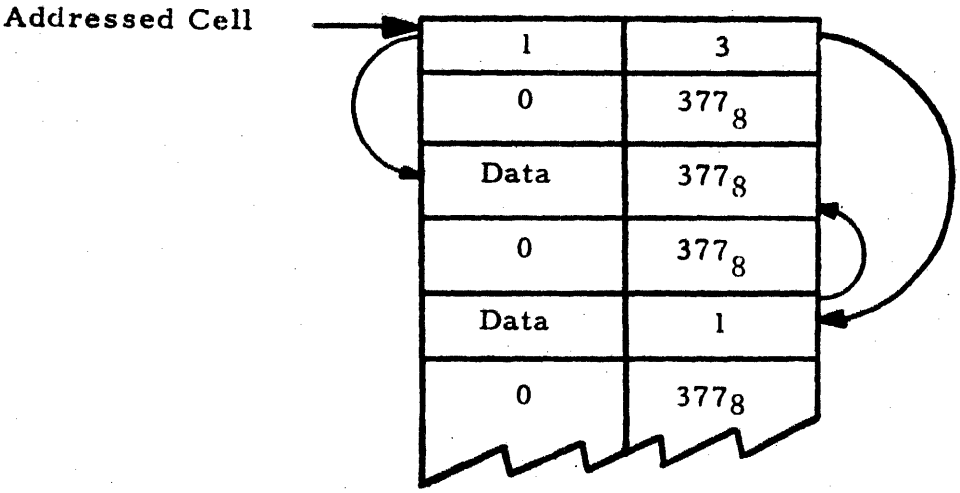
This instruction is the same as STM UAE.

LDD Load Double 41(3) 6D

This instruction is the same as LDM UA.

LINK Link Item Item Into FIFO List 67(0) 6A

The LINK and DLINK instructions address a first in, first out (FIFO) queue with the following structure:



The addressed cell contains start and end pointers for the elements of the queue. The cell following the addressed cell is queue entry number 0. There are a maximum of 377_8 entries in the queue (0 to 376_8). A pointer of 377_8 is used to mean "no pointer". Thus if the queue is empty, the addressed cell will contain 377_8 in each byte. The example shows a queue of two entries, number 3 and number 1.

The LINK instruction operates as follows:

The contents of the X register, $0 \leq (X) \leq 376_8$, are the entry number to be added to the queue. This entry must not already be linked into the queue, (i. e., must have a forward pointer of 377, and must not be pointed to by the queue end pointer). If this test fails then a no skip return is given. Otherwise a new queue entry is added to the end, with the data given in the low order 8 bits of A.

DLINK Remove Item from FIFO List 67(1) 6A

The number of the first item in the queue is placed in X, the data is placed in the lower byte of A, the item is removed from the queue, and a skip return is given. If there are no items in the queue a no skip return is given.

INTER-REGISTER INSTRUCTIONS

RCPY Register Copy 0 7A

(S) → D

where S may be X, U, A, E, B, T, L, or 1
and D may be X, U, A, E, B, T, or L.

The contents of the source register S are loaded into the destination register D. For example,

```
RCPY   1E
```

places the constant 1 in the E register.

Modifiers: None.

RNEG Register Negate — 7B

(S) → D

where S may be X, U, A, E, B, T, L, or 1
and D may only be X, U, A, or E.

The contents of the source register S are negated and the result is loaded into the destination register D. When the source and destination registers are the same the argument need only be specified once. Hence,

```
RNEG   UU
```

is equivalent to RNEG U

Modifiers: None.

RXCH Register Exchange 005-012 2

(S) → D; (D) → S

where S may be X, U, A, or E
and D may be X, U, A, or E.

The contents of the specified registers are exchanged.

For example,

RXCH AE

exchanges the contents of the A and E registers.

Modifiers: None.

XSA Extend Sign of A 014 2

$(A_0) \rightarrow U$

The sign of A, bit position 0, is extended through the U register. This instruction is very useful in preparing a single word argument for a double word instruction - as in a fixed point divide, etc.

Modifiers: None.

RDS Read Status 015 2

(Status) → A

where (Status) = machine status.

The contents of the (Status) register are placed into the A register. Bit positions and functions are described in Table 4-2, Status Word Contents.

Modifiers: None.

FIXED-POINT ARITHMETIC

ADX Add to X 12 1A, B, C, D

(X) + (y) → X

The contents of memory location y are added to the contents of the X register.

Modifiers: P, X, E, B, *, =

ADXI Add to X, Immediate 04 10

(X) + LIT0 → X

The 9-bit literal contained in bit positions 7-15 of the instruction (with sign extended) is added to the contents of the X register.

Modifiers: None.

ADXIS Add to X, Immediate and Skip 05 10

Skip if X = 0; if not set (X) + LIT9 → X

If X is zero, the next sequential instruction is skipped and the following instruction is executed. If X is not zero, the literal in bit positions 7-15 is added to the X register as in the ADXI instruction.

Modifiers: None.

SBX Subtract from X 16 1A, B, C, D

(X) - (y) → X

The contents of memory location y are subtracted from the contents of the X register.

Modifiers: P, X, E, B, *, =

RSBX Reverse Subtract X 15(1) 6A, G

(y) - (X) → X

The contents of the X register are subtracted from the memory location y.

Modifiers: P, X, E, B, *, =

MPX Multiply X 13(0) 6A, G

(X) * (y) → X

The contents of the X register and memory location y are multiplied. The result is placed in the X register.

Modifiers: B, X, *, =

ADU ADD to U 14(0) 6A, G

(U) + (y) → U

The contents of memory location y are added to the contents of the U register. Overflow (OF) may be set. Carryout (CO) is set or reset.

Modifiers: B, X, *, =

ADUI Add to U, Immediate 06 10

(U) + LIT9 → U

The 9-bit literal contained in bit positions 7-15 of the instruction (with sign extended) is added to the U register. Overflow (OF) may be set. Carryout (CO) is set or reset.

Modifiers: None.

SBU Subtract from U 14(1) 6A, G

(U) - (y) → U

The contents of memory location Y are subtracted from the U register. Overflow (OF) may be set. Carryout (CO) is set or reset.

Modifiers: B, X, *, =

ADA Add to A 10 1A, B, C, D

(A) + (y) → A

The contents of memory location y are added to the A register. Overflow (OF) may be set. Carryout (CO) is set or reset.

Modifiers: P, X, E, B, *, =

ADAI Add to A, Immediate 07 10

(A) + LIT9 → A

The 9-bit literal contained in bit positions 7-15 of the instruction (with sign extended) is added to the A register. Overflow (OF) may be set. Carryout (CO) is set or reset.

Modifiers: None.

SBA Subtract from A 14 1A, B, C, D

(A) - (y) → A

The contents of memory location y are subtracted from the A register. Overflow (OF) may be set. Carryout (CO) is set or reset.

Modifiers: P, X, E, B, *, =

RSBA Reverse Subtract A 14(2) 6A, G

(y) - (A) → A

The contents of the A register are subtracted from memory location y. The result is placed in the A register. Overflow (OF) may be set. Carryout (CO) is set or reset.

Modifiers: B, X, *, =

MPA Multiply A 13(1) 6A, G

(A) * (y) → U, A

The contents of the A register and memory location y are multiplied. The two word product is placed in the extended accumulator U, A. If

the product does not fit in one register, overflow (OF) is set. That is, if either $(A_0) = 0$ and (U) = 0, or if $(A_0) = 1$, and (U) = 177777.

Modifiers: B, X, *, =

DVUA Divide U and A 16(0) 6A, G

(U, A) / (y) → A; Remainder → U

The contents of the extended accumulator U, A are divided by the contents of memory location y. The quotient is placed in the A register. The remainder is placed in the U register. Overflow (OF) may be set.

Modifiers: B, X, *, =

DVA Divide A 16(1) 6A, G

(A) / (y) → A; Remainder → U

The contents of the A register are divided by the contents of memory location y. The quotient is placed in the A register. The remainder is placed in the U register. Overflow (OF) may be set.

Modifiers: B, X, *, =

RDVA Reverse Divide A 16(3) GA, G

(y) / (A) → A; Remainder → U

The contents of memory location y are divided by the contents of the A register. The quotient is placed in the A register. The remainder is placed in the U register. Overflow (OF) may be set.

Modifiers: B, X, *, =

RADD Register Add 1 7A

(D) + (S) → D

where S may be X, U, A, E, B, T, L, or 1
and D may be X, U, A, E, B, T, or L.

The contents of the source register S are added to the contents of the destination register D. If the source and destination registers are the same, the argument need only be specified once. Hence,

RADD XX

is equivalent to RADD X

Modifiers: None.

RSUB Register Subtract 2 7A

(D) - (S) → D

where S = X, U, A, E, B, T, L, or 1

and D = X, U, A, E, B, T, or L.

The contents of the source register S are subtracted from the contents of the destination register D. If the source and destination registers are the same, the argument need only be specified once. Hence,

RSUB TT

is equivalent to RSUB T

Modifiers: None.

ADDM Add to Memory 17(0) 6A

(y) + (A) → y

The contents of the A register are added to the contents of memory location y. Overflow may be set. Carryout is set or reset. The contents of A are not changed.

Modifiers: B, X, *

SUBM Subtract from Memory 17(1) 6A

(y) - (A) → y

The contents of the A register are subtracted from the contents of memory location y. Overflow may be set. Carryout is set or reset. The contents of A are not changed.

Modifiers: B, X, *

MINC Memory Increment, Skip 20(SC) 6A

(y) + 1 → y; Skip on Condition

The contents of memory location y are incremented by one. The contents of memory location y are compared to zero. If the specified condition is met, the next sequential instruction is skipped and the following instruction is executed.

The variable field of the instruction may have three subfields. They are: The skip condition, the address, and modifiers, if any. For example,

MINC GE, A

The contents of A are increased by one. If the result is "greater than or equal to" (GE) zero, the next sequential instruction is skipped.

Refer to Table 4-2 for the mnemonics and meaning for all skip condition instructions.

TABLE 4-2. CONDITIONS FOR ALL SKIP/JUMP INSTRUCTIONS

Condition	Meaning	Skip/Jump Condition bits 7, 8, 9 of instruction
N	Never Skip (Jump)	000
GT	Greater than	001
EQ	Equal	010
GE	Greater than or Equal to	011
LT	Less than	100
NE	Not Equal	101
LE	Less than or Equal to	110
U	Unconditional Skip (Jump)	111

MDEC Memory Decrement, Skip 21(SC) 6A

(y) - 1 → y; Skip on Condition

The contents of memory location y are decremented by one. If the specified condition is met, the next sequential instruction is skipped and the following instruction executed.

The variable field of instruction may have three subfields. They are: the skip condition, the address, and modifiers, if any. For example,

MDEC EQ, A

The contents of A are decremented by one. If the result is zero, the next sequential instruction is skipped.

Modifiers: B, X, *

TAD Triple Add 26(0) 6A

(U,A,E) + (y, y + 1, y + 2) → U,A,E

The contents of the memory locations y, y + 1, and y + 2 are added to the contents of U,A,E. If the sign magnitude add results in an overflow, the results are right shifted one and the overflow bit is set.

Modifiers: B, X, *

NTAD Negate Triple Add 26(3) 6A

-(U,A,E) - (y, y +1, y + 2) → U,A,E

The contents of U,A,E and (y, y + 1, y + 2) are negated. After negated both the results are added and placed in U, A, E. If overflow occurs, the results are right shifted one and the overflow bit is set.

Modifiers: B, X, *

TSB Triple Subtract 26(1) 6A

(U,A,E) - (y, y +1, y + 2) → U,A,E

The contents of the memory locations (y, y + 1, y + 2) are subtracted from the contents of U,A,E. If overflow occurs, the results are right shifted one and the overflow bit is set.

Modifiers: B, X, *

RTSB Reverse Triple Subtract 26(2) 6A

Y, y + 1, y + 2) - (U,A,E) → U,A,E

The contents of U,A,E are subtracted from (y, y + 1, y + 2) and the results placed in U,A,E. If an overflow occurs, the results are right shifted one and the overflow bit is set.

Modifiers: B, X, *

TMP Triple Multiply 24(1) 6A

(U,A,E) * (y, y + 1, y + 2) → U,A,E

This is a 3 word (sign magnitude) integer multiply. If an overflow occurs, the overflow bit will be set.

Modifiers: B, X, *

TMPF Triple Multiply Fractional 24(3) 6A

(U,A,E) (y, y + 1, y + 2) → U,A,E

This instruction was implemented for use within a 4 word floating point multiply routine. This instruction ignores both input signs and uses the sign bit of the result to return an extra bit of significance. For example:

$$\overline{010}\,0 * \overline{010}\,0 = \overline{010}\,0$$
$$\overline{110}\,0 * \overline{010}\,0 = \overline{010}\,0$$
$$\overline{110}\,0 * \overline{110}\,0 = \overline{010}\,0$$
$$01111\ldots1 * 01111\ldots1 = 1111\ldots$$

This instruction does not affect overflow or carryout bits.

TDV Triple Divide 25(1) 6A

(U,A,E) / (y, y +1, y + 2) → U,A,E

This is a 3 word (sign magnitude) integer divide. If any number is divided by zero, overflow will occur and the overflow bit will be set.

Modifiers: B, X, *

TDVF Triple Divide Fractional 25(3) 6A

(U,A,E) / (y, y + 1, y + 2) → U,A,E

This instruction was implemented for use within a 4 word floating point divide routine. This instruction ignores both input signs and uses the sign bit of the result to return the most significant bit of the results. For example:

$$\overline{010\ 0}/\overline{010\ 0} = \overline{10\ 0}$$
$$\overline{110\ 0}/\overline{010\ 0} = \overline{10\ 0}$$
$$\overline{110\ 0}/\overline{110\ 0} = \overline{10\ 0}$$
$$0111\ldots\ 1/\overline{010\ 0} = 1111\ldots\ 1$$
$$\overline{010\ 0}/01111\ldots\ 1 = 01000\ldots$$

This instruction does not affect overflow or carryout bits.

ADAS Add to A, Stack 020 2

((T) - 1) + (A) → A; (T) - 1 → T

The most current item in the stack (pointed to by the top-of-stack pointer T) is added to the contents of the A register. The entry is popped from the stack when the contents of the T register are decremented by 1.

Overflow may be set. Carryout is set or reset. Stack underflow trap occurs if (T) < (B).

Modifiers: None.

SBAS Subtract from A, Stack 021 2

(A) - ((T) - 1) → A; (T) - 1 → T

The most current item in the stack (pointed to by the top-of-stack pointer T) is subtracted from the contents of the A register. The entry is popped from the stack when the contents of the T register are decremented by one.

Overflow may be set. Carryout is set. Carryout is set or rest. Stack underflow trap occurs if (T) < (B).

Modifiers: None.

RSBAS Reverse Subtract from A, Stack 022 2

((T) - 1) - (A) → A; (T) - 1 → T

The contents of the A register are subtracted from the most current item in stack (pointed to by the top-of-stack pointer T). The result is placed in the A register. The entry is popped from the stack when the contents of the T register are decremented by one.

Overflow may be set. Carryout is set or reset. Stack underflow occurs if (T) < (B).

Modifiers: None

MPAS Multiply A, Stack 023 2

(A) * ((T) - 1) → U, A; (T) - 1 → T

The most current item in the stack is multiplied by the contents of the A register. The double word product is placed into the extended accumulator U, A. The entry is popped from the stack when the contents of the T register are decremented by one.

Stack underflow trap occurs if (T) < (B). Overflow is set if either $(A_0) = 0$ and (U) ≠ 0, or, if $(A_0) = 1$ and (U) ≠ 177777. That is, if the product does not fit into one register.

Modifier: None.

DVAS Divide A, Stack 024 2

(A)/((T)-1) → A; (T) - → T; Remainder → U

The contents of the A register are divided by the most current item in the stack. The quotient is placed in the A register, and the remainder is placed in the U register. The entry in the stack is popped when the contents of the T register are decremented by one.

Overflow may be set. Stack underflow trap occurs if (T) < (B).

Modifiers: None.

RDVAS Reverse Divide A, Stack 025 2

((T) -1)/(A) → A; (T) - 1 → T; Remainder → U

The most current item in the stack is divided by the contents of the A register. The quotient is placed in the A register, and the remainder is placed in the U register. The entry in the stack is popped when the contents of the T register are decremented by one.

Overflow may be set. Stack underflow trap occurs if (T) < (B).

Modifiers: None.

ADUS Add to U, Stack 25(0) 2

((T) - 1) + (U) → A; (T) - 1 → T

The most current item in the stack (pointed to by the top-of-stack pointer T) is added to the contents of the U register. The entry is popped from the stack when the contents of the T register are decremented by 1.

Overflow may be set. Carryout is set or reset. Stack underflow trap occurs if (T) < (B).

Modifiers: None.

SBUA Subtract from A, Stack 021 2

(U) - ((T) - 1) → U; (T) - 1 → T

The most current item in the stack (pointed to by the top-of-stack pointer T) is subtracted from the contents of the U register. The entry is popped from the stack when the contents of the T register are decremented by one.

Overflow may be set. Carryout is set. Carryout is set or reset. Stack underflow trap occurs if (T) < (B).

DVUAS Divide UA, Stack 27(0) 2

(U, A) / ((T) - 1) → A Remainder → U (T) - 1 → T

The contents of the U, A registers are divided by the most current item in the stack. The quotient is placed in the A register, and the remainder

is placed in the U register. The entry in the stack is popped when the contents of the T register are decremented by one.

Overflow may be set. Stack underflow trap occurs if (T) < (B).

Modifiers: None.

ADXS Add to X, Stack 020 2

((T) - 1 + (X) → X; (T) - 1 → T

The most current item in the stack (pointed to by the top-of-stack pointer T) is added to the contents of the X register. The entry is popped from the stack when the contents of the T register are decremented by 1.

Overflow may be set. Carryout is set or reset. Stack underflow trap occurs if (T) < (B).

Modifiers: None.

SBXS Subtract from X, Stack 021 2

(X) - ((T) - 1 → A; (T) - 1 → T

The most current item in the stack (pointed to by the top-of-stack pointer T) is subtracted from the contents of the X register. The entry is popped from the stack when the contents of the T register are decremented by one.

Overflow may be set. Carryout is set. Carryout is set or reset. Stack underflow trap occurs if (T) < (B).

Modifiers: None.

RSBXS Reverse Subtract from X, Stack 022 2

((T) - 1) - (X) → X; (T) - 1 → T

The contents of the X register are subtracted from the most current item in stack (pointed to by the top-of-stack pointer T). The result is placed in the X register. The entry is popped from the stack when the contents of the T register are decremented by one.

Overflow may be set. Carryout is set or reset. Stack underflow occurs if (T) < (B).

Modifiers: None.

MPXS Multiply X, Stack 023 2 C

(X) * ((T) - 1) → X; (T) - 1 → T

The most current item in the stack is multiplied by the contents of the X register. The product is placed into the X register. The entry is popped from the stack when the contents of the T register are decremented by one.

Stack underflow trap occurs if (T) < (B).

Modifier: None.

TADS Triple Add Stack 33(0) 2

$$(U,A,E) + ((T) - 3, (T) - 2, (T) - 1) \rightarrow U,A,E;\ (T) - 3 \rightarrow T$$

The three most current words in the stack (pointed to by the top of stack pointer T) are added to the three-word accumulator U, A, E. The value is automatically popped from the stack when the T register is decremented by three. If the sign magnitude add results in an overflow, the results are right shifted one and the overflow bit is set.

An overflow or underflow trap may occur. Stack underflow trap occurs if initially (T) - 3 < (B).

Modifiers: None.

NTADS Negative Triple Add Stack 33(3) 2

$$-(U,A,E) - ((T - 3, (T) - 2, (T) - 1) \rightarrow U,A,E;\ (T) - 3 \rightarrow T$$

U, A, E and its 3 most current words in the stack are negated, then added together and results placed in U, A, E. The value is automatically popped from the stack when the T register is decremented by three.

If overflow occurs, the results are right shifted one and the overflow bit is set.

Modifiers: None.

TSBS Triple Subtract, Stack 33(1) 2

$$(U,A,E) - ((T) - 3, (T) - 2, (T) - 1 \rightarrow U,A,E;\ (T) - 3 \rightarrow T$$

The three most current words in the stack (pointed to by the top-of-stack pointer T) is subtracted from the three-word accumulator U, A, E. The value is automatically popped from the stack when the T register is decremented by three. If overflow occurs, the results are right shifted one and the overflow bit is set.

An overflow or underflow trap may occur. Stack underflow trap occurs if initially (T) - 3 < (B).

Modifiers: None.

RTSBS Reverse Triple Subtract, Stack 33(2) 2

((T) - 3, (T) - 2, (T) - 1) - (U,A,E) → U,A,E; (T) - 3 → T

The quantity in the three-word accumulator U,A,E is subtracted from the three most current words in the stack. The result is placed in U,A,E. The item is automatically popped from the stack when the T register is decremented by three. If an overflow occurs, the results are right shifted one and the overflow bit is set.

An overflow or underflow may occur. Stack underflow trap occurs if initially (T) - 3 < (B).

Modifiers: None.

TMPS Triple Multiply, Stack 31(1) 2

(U,A,E) * ((T) - 3, (T) - 2, (T) -1) → U,A,E; (T) - 3 → T

The 3 word (sign magnitude) integer in the accumulator U,A,E, is multiplied by the three most current words in the stack. The product is placed in U,A,E. Three words are automatically popped from the stack when the T register is decremented by three. If an overflow occurs, the overflow bit will be set.

A stack overflow or underflow may occur. Stack underflow trap occurs if initially (T) - 3 < (B).

Modifiers: None.

TMPFS Triple Multiple Fractional Stack 31(3) 2

(U,A,E) * ((T) - 3, (T) - 2, (T) - 1) → U,A,E; (T) - 3 → T

The quantity in the three-word accumulator U,A,E, is multiplied by the three most current words in the stack. The product is placed in U,A,E. The 3 words are automatically popped from the stack when the T register is decremented by three.

The instruction was implemented for use within a 4 word floating point multiply routine. This instruction ignores both input signs and uses the sign bit of the result to return an extra bit of significance.

$$0\overline{10\ldots0} * 0\overline{10\ldots0} = 0\overline{10\ldots0}$$
$$1\overline{10\ldots0} * 0\overline{10\ldots0} = 0\overline{10\ldots0}$$
$$1\overline{10\ldots0} * 1\overline{10\ldots0} = 0\overline{10\ldots0}$$
$$01111\ldots1 * 01111\ldots1 = 1111\ldots$$

This instruction does not affect overflow or carryout bits.

A stock overflow or underflow may occur. Stack underflow trap occurs if initially (T) - 3 < (B).

Modifiers: None.

TDVS Triple Divide, Stack 32(1) 2

$$(U,A,E) / ((T) - 3, (T) - 2, (T) - 1) \rightarrow U,A,E; (T) - 3 \rightarrow T$$

The 3 word (sign magnitude) integer in the accumulator U, A, E, is divided by the three most current words in the stack. The quotient is placed in U, A, E. The item is automatically popped from the stack when the T register is decremented by three. If any number is divided by zero, overflow will occur and the overflow bit will be set.

A stack overflow or underflow trap may occur. Stack underflow trap occurs if initially (T) - 3 < (B).

Modifiers: None.

TDVFS Triple Divide Fractional, Stack 32(3) 2

$$(U,A,E) / ((T) - 3, (T) - 2, (T) - 1) \rightarrow U,A,E; (T) - 3 \rightarrow T$$

The quantity in the three-word accumulator U, A, E, is divided by the three most current words in the stack. The quotient is placed in U, A, E. Three words are automatically popped from the stack when the T register is decremented by three.

This instruction was implemented for use within a 4 word floating point divide routine. This instruction ignores both input signs and uses the sign bit of the result to return the most significant bit of the results. For example:

$$\begin{aligned}
&010\overline{\quad}0/010\overline{\quad}0 = 10\overline{\quad}0\\
&110\overline{\quad}0/010\overline{\quad}0 = 10\overline{\quad}0\\
&110\overline{\quad}0/110\overline{\quad}0 = 10\overline{\quad}0\\
&01111\ldots 1/010\overline{\quad}0 = 1111\ldots 1\\
&010\overline{\quad}0/01111\ldots 1 = 01000\ldots
\end{aligned}$$

This instruction does not affect overflow or carryout bits.

A stack overflow or underflow trap may occur. Stack underflow trap occurs if initially (T) - 3 < (B).

Modifiers: None.

TNEG Triple Negate 53 2

If [(U) ⊕ (A) ⊕ (E)] 0 Then (Uo) → Uo

This instruction changes the sign bit of the U register if U, A, and E are not all equal to zero. If U = 0, A = 0, and E = 0 the instruction is a NOP.

Modifiers: None

FLOATING POINT ARITHMETIC

FAD Floating Add 23(0) 6A

(U, A, E) + (y, y + 1, y + 2) → U, A, E

The floating-point quantity contained in memory locations y, y + 1, and y + 2, is added to the contents of the three-word floating-point accumulator U, A, and E. The result is normalized.

A floating-point overflow or underflow trap may occur.

Modifiers: B, X, *

NFAD Negated Floating Add 23(3) 6A

-(U, A, E) - (y, y + 1, y + 2) → U, A, E

U, A, E and the floating-point quantity contained in memory locations y, y + 1, y + 2, are negated. After negating the two quantities are add and the results placed in U, A, E.

A floating-point overflow or underflow trap may occur.

Modifiers: B, X, *

FSB Floating Subtract 23(1) 6A

(U, A, E) - (y, y + 1, y + 2) → U, A, E

The floating-point quantity contained in memory locations y, y + 1, y + 2, is subtracted from the contents of the three-word accumulator U, A, E. The result is normalized.

A floating-point overflow or underflow trap may occur.

Modifiers: B, X, *

RFSB Reverse Floating Subtract 23(2) 6A

(y, y + 1, y + 2) - (U, A, E) → U, A, E

The floating-point quantity contained in the 3 word accumulator U, A, E, is subtracted from the contents of memory locations y, y + 1, y + 2. The result is normalized and placed into the three-word accumulator U, A, E.

A floating-point overflow or underflow trap may occur.

Modifiers: B, X, *

FMP Floating Multiply 24(0) 6A

(U, A, E) * (y, y + 1, y + 2) → U, A, E

The floating-point quantity contained in the three-word accumulator, U, A, E, is multiplied by the contents of memory location y, y + 1, y + 2. The result is normalized.

A floating-point overflow or underflow trap may occur.

Modifiers: B, X, *

FDV Floating Divide 25(0) 6A

(U, A, E) / (y, y + 1, y + 2) → U, A, E

The floating-point quantity contained in the three-word accumulator U, A, E, is divided by the contents of the memory locations y, y + 1, y + 2. The result is normalized and placed in U, A, E.

A floating-point overflow or underflow or underflow trap may occur.

Modifiers: B, X, *

RFDV Reverse Floating Divide 25(2) 6A

(y, y + 1, y + 2) / (U, A, E) → U, A, E

The contents of memory locations y, y + 1, y + 2 are divided by the floating-point quantity contained in the three-word accumulator U, A, E. The result is normalized and placed in U, A, E.

A floating-point overflow or underflow trap may occur.

Modifiers: B, X, *

FADS Floating Add, Stack 30(0) 2

(U, A, E) + ((T) - 3, (T) - 2, (T) - 1) → U, A, E; (T) - 3 → T

The most current floating-point quantity in the stack (pointed to by the top of stack pointer T) is added to the three-word accumulator U, A, E.

The value is automatically popped from the stack when the T register is decremented by three.

A floating-point overflow or underflow trap may occur. Stack underflow trap occurs if initially (T)-3 < (B).

Modifiers: None.

NFADS Negated Floating Add, Stack 30(3) 2

$$-(U, A, E) - ((T) - 3, (T) - 2, (T) - 1) \rightarrow U, A, E; (T) - 3 \rightarrow T$$

U, A, E and the 3 most current words in the stack are negated, then added together and results placed in U, A, E. The value is automatically popped from the stack when the T register is decremented by three.

A floating-point overflow or underflow trap may occur. Stack underflow trap occurs if initially (T)-3 < (B).

Modifiers: None.

FSBS Floating Subtract, Stack 30(1) 2

$$(U, A, E) - ((T) - 3, (T) - 2, (T) - 1) \rightarrow U, A, E; (T) - 3 \rightarrow T$$

The most current floating-point quantity in the stack (pointed to by the top-of-stack pointer T) is subtracted from the three-word accumulator U, A, E. The value is automatically popped from the stack when the T register is decremented by three.

A floating-point overflow or underflow trap may occur. Stack underflow trap occurs if initially (T) - 3 < (B).

Modifiers: None.

RFSBS Reverse Floating Subtract, Stack 30(2) 2

$$((T) - 3, (T) - 2, (T) - 1) - (U, A, E) \rightarrow U, A, E; (T) - 3 \rightarrow T$$

The floating-point quantity in the three-word accumulator U, A, E is subtracted from the most current floating-point item in the stack. The result is placed in U, A, E. The item is automatically popped from the stack when the T register is decremented by three.

A floating-point overflow or underflow may occur. Stack underflow trap occurs in initially (T) - 3 < (B).

Modifiers: None.

FMPS Floating Multiply, Stack 31(0) 2

(U, A, E) * ((T) - 3, (T) - 2, (T) - 1) → U, A, E; (T) - 3 → T

The floating-point quantity in the three-word accumulator U, A, E, is multiplied by the most current floating point item in the stack. The product is placed in U, A, E. The item is automatically popped from the stack when the T register is decremented by three.

A floating-point overflow or underflow may occur. Stack underflow trap occurs if initially (T) - 3 < (B).

Modifiers: None.

FDVS Floating Divide, Stack 32(0) 2

(U, A, E) / ((T) - 3, (T) - 2, (T) - 1) → U, A, E; (T)-3 → T

The floating-point quantity in the three-word accumulator U, A, E, is divided by the most current floating-point item in the stack. The quotient is placed in the U, A, E. The item is automatically popped from the stack when the T register is decremented by three.

A floating-point overflow or underflow trap may occur. Stack underflow trap occurs if initially (T)-3 < (B).

Modifiers: None.

RFDVS Reverse Floating Divide, Stack 32(2) 2

((T) - 3, (T) - 2, (T) - 1) / (U, A, E) → U, A, E; (T) -3 → T

The most current floating-point item in the stack is divided by the contents of the three-word accumulator U, A, E. The quotient is placed in U. A, E. The item is automatically popped from the stack when the T register is decremented by three.

A floating-point overflow or underflow trap may occur. Stack underflow trap occurs if initially (T)-3 < (B).

Modifiers: None.

FIX Fix a Floating-point Number 041 2

The floating-point quantity contained in the three-word accumulator U, A, E is converted to a fixed-point integer and placed in the A register.

Overflow may be set.

Modifiers: None.

FLOAT Float an integer 042 2

The fixed-point integer in the A register is converted to a floating-point quantity and placed in the three-word accumulator U, A, E.

Modifiers: None

NORM Floating Normalize 043 2

The instruction normalizes the floating-point quantity contained in the three-word accumulator U, A, E.

Modifiers: None.

FNEG Floating Negate 54(0) 2

If $[(U) \oplus (A)] \neq 0$ then $(\overline{U_0}) \rightarrow U_0$

If U and A are not both zero, the sign bit of U will be changed. If U = 0 and A = 0 the instruction is a NOP.

LOGICAL INSTRUCTIONS

In Boolean operations, the operators $\otimes$, $\oplus$, and $\ominus$ have the definitions shown in Table 4-3.

TABLE 4-3 DEFINITIONS OF BOOLEAN OPERATIONS

Operator	Meaning	Definition
$\otimes$	AND; intersection	$0 \otimes 0 = 0$ $0 \otimes 1 = 0$ $1 \otimes 0 = 0$ $1 \otimes 1 = 1$
$\oplus$	OR, inclusive, union	$0 \oplus 0 = 0$ $0 \oplus 1 = 1$ $1 \oplus 0 = 1$ $1 \oplus 1 = 1$
$\ominus$	EXCLUSIVE OR, symmetric difference	$0 \ominus 0 = 0$ $0 \ominus 1 = 1$ $1 \ominus 0 = 1$ $1 \ominus 1 = 0$

ANX AND with X 27(0) 6A, G

(X) ⊗ (y) → X

The contents of memory location y are ANDed with the contents of the X register.

Modifiers: B, X, *, =

ANU AND with U 27(1) 6A, G

(U) ⊗ (y) → U

The contents of memory location y are ANDed with the contents of the U register.

Modifiers: B, X, *, =

ANUI AND with U, Immediate 10 10

(U) ⊗ LIT9 → U

The 9-bit literal contained in bit position 7-15 of the instruction (with sign extended) is ANDed with the contents of the U register.

Modifiers: None.

ANUA And with U and place results in A 27(3) 6A, 6G

(U) ⊗ (y) → A

The contents of memory location y are ANDed with the contents of the U registers (U is left unchanged) and the results are placed in the A registers.

ANA AND with A 20 1A, B, C, D

(A) ⊗ (y) → A

The contents of memory location y are ANDed with the contents of the A register.

Modifiers: P, X, B, E, *, =

ANAI AND with A, Immediate 11 10

(A) ⊗ LIT9 → A

The 9-bit literal contained in bit positions 7-15 of the instruction (with sign extended) is ANDed with the contents of the A register.

Modifiers: None.

ORA OR with A 22 1A, B, C, D

(A) $\oplus$ (y) → A

The contents of memory location y are ORed with the contents of the A register.

Modifiers: P, X, B, E, *, =

ORAI OR with A, Immediate 12 10

(A) $\oplus$ LIT9 → A

The 9-bit literal contained in bit positions 7-15 of the instruction (with sign extended) is ORed with the contents of the A register.

Modifiers: None.

XRA EXCLUSIVE OR with A 24 1A, B, C, D

(A) $\ominus$ (y) → A

The contents of memory location y are EXCLUSIVE ORed with the contents of the A register.

Modifiers: P, X, B, E, *, =

XRAI EXCLUSIVE OR with A, Immediate 13 10

(A) $\ominus$ LIT9 → A

The 9-bit literal contained in bit positions 7-15 of the instruction (with sign extended) is EXCLUSIVE ORed with the contents of the A register.

Modifiers: None.

RAND Register AND 3 7A

(S) $\otimes$ (D) → D

where S may be X, U, A, E, B, T, L, or 1
and D may be X, U, A, E, B, T, or L

The contents of the source register S are ANDed with the contents of the destination register D. For example.

```
RAND EX
```

the E register is ANDed with the X register. The result is placed in the X registers.

Modifiers: None.

SETBA Set Bit in A 3 4B

$1 \rightarrow A_i$

where i is a designated bit number.

The bit number in the A register, designated in the variable field of the instruction, is set to a one. The bit number specified must be an absolute expression. There are two forms of the instruction.

```
SETBA    3
```

sets bit position 3 of the A register to a one; and,

```
SETBA    3,N
```

where N is a modifier indicates the bit number (in the example, 3), is modified by the X register with the result, truncated to four bits, used as the effective bit number.

Modifiers: N

CLRBA Clear Bit in A 4 4B

$0 \rightarrow A_i$

where i is a designated bit number.

The bit position in the A register, designated by the bit number in the variable field of the instruction, is set to zero. The bit number specified must be an absolute expression. There are two forms of the instruction.

```
CLRBA    13
```

sets bit position 13 of the A register to zero; and,

```
CLRBA    13,N
```

where N specifies a modifier indicating the bit number is modified by the X register with the result truncated to four bits, used as the effective bit number.

Modifiers: N

CMPBA Complement Bit in A 5 4B

$$-(A_i) \quad A_i$$

where i is a designated bit number.

The bit position in the A register, designated by the bit number in the variable field of the instruction, is complemented. The bit number specified must be an absolute expression.

Modifiers: N

SETBM Set Bit in Memory 3 5

$$1 \rightarrow y_i$$

where i is a designated bit number.

The bit position of memory location y, designated by the bit number in the variable field of the instruction, is set to a one.

The variable field of the instruction contains three subfields: the bit number (absolute expression), the memory address, and modifiers, if any.

If the instruction is modified by X and not N, the memory address is modified by all 16 bits of the X register.

If the instruction is modified by N and not X, the bit number is modified by the low order four bits of the X register (modulo 16).

If the instruction is modified by both X and N, the memory address is modified by the high order 12 bits of the X register (bit positions 0-11); and the bit number is modified by the low order four bits of the X register (bit positions 12-15). If there is carryout after modifying the bit number (> 15), the carry is added to the memory address calculation for y.

Modifiers: X, B, N, *

CLRBM Clear Bit in Memory 4 5

$$0 \rightarrow y_i$$

where i is a designated bit number.

The bit position of memory location y, designated by the bit number in the variable field of the instruction, is set to zero.

The variable field of the instruction contains 3 subfields: the bit number (absolute expression), the memory address, and modifiers, if any.

If the instruction is modified by X and not N, the memory address is modified by all 16 bits of the X register.

If the instruction is modified by N and not X, the bit number is modified by the low order 4 bits of the X register (modulo 16).

If the instruction is modified by both X and N, the memory address is modified by the high order 12 bits of the X register (bit positions 0-11); and the bit number is modified by the low order four bits of the X register (bit positions 12-15). If there is a carryout after modifying the bit number (1 > 15), the carry is added to the memory address calculation for y.

Modifiers: X, B, N, *

CMPBM Complement Bit in Memory 5 5

$$-(y_i) \rightarrow y_i$$

where i is a designated bit number.

The bit position of memory location y, designated by the bit number in the variable field of the instruction, is complemented.

The variable field of the instruction contains three subfields: the bit number (absolute expression), the memory address, and modifiers, if any.

The rules for modifiers X, and N are the same as those defined for the SETBM instruction.

Modifiers: X, B, N, *

ANAS AND with A, Stack 22(1) 2

$$(A) \otimes ((T) - 1) \rightarrow A; \quad (T) - 1 \rightarrow T$$

The most current item in the stack (pointed to by the top-of-stack pointer T) is ANDed with the contents of the A register. The entry is

automatically popped from the stack when the contents of the T register are decremented by one.

Stack underflow trap occurs if (T) < (B).

Modifiers: None.

ORAS OR with A, Stack 22(2) 2

$$(A) \oplus ((T) - 1) \rightarrow A; \quad (T) - 1) \rightarrow T$$

The most current item in the stack (pointed to by the top-of-stack pointer T) is ORed with the contents of the A register. The entry is automatically popped from the stack when the contents of the T register are decremented by one.

Stack underflow trap occurs if (T) < (B).

Modifiers: None.

XRAS EXCLUSIVE OR with A, Stack 22(3) 2

$$(A) \ominus ((T) - 1) \rightarrow A; \quad (T) - 1 \rightarrow T$$

The most current item in the stack (pointed to by the top-of-stack pointer T) is EXCLUSIVE ORed with the contents of the A register. The entry is automatically popped from the stack when the contents of the T register are decremented by one.

A stack underflow trap occurs if (T) < (B).

Modifiers: None.

ANXS AND with X, Stack 22(0) 2

$$(X) \otimes ((T) - 1) \rightarrow X; \quad (T) - 1 \rightarrow T$$

The most current item in the stack (pointed to by the top-of-stack pointer T) is ANDed with the contents of the X register. The entry is automatically popped from the stack when the contents of the T register are decremented by one.

Stack underflow trap occurs if (T) < (B).

Modifiers: None.

SHIFT INSTRUCTIONS

A note concerning modification of shift counts by indexing. On all shifts, indexing is performed modulo 2^5 (or 2^6 for double register shifts). The sign of the result is bit position 11 (or 10). That is, the 5- or 6-bit shift count is added to the contents of the X register. The result is treated modulo 2^5 (or 2^6).

LLX/LRX Logical Left/Right Shift X 0 8

The contents of the X register are shifted left or right C (C = count) places, with zeros filling vacated bit positions. Bits shifted past bit position 0 (left), or bit position 15 (right) are lost.

The direction of shift is determined by the count, C, after indexing. If count > 0, then left shift. If count < 0, then right shift.

Modifiers: X

ALU/ARU Arithmetic Left/Right Shift U 1 8

The contents of the U register are shifted left or right C (C = count) places. If the shift is to the left, zeros are filled into vacated bit positions on the right. If the shift is to right, the contents of bit position 0 are filled into vacated positions on the left. Bits shifted past bit positions 0 (left) or bit position 15 (right) are lost.

The direction of shift is determined by the count, C, after indexing. If count > 0, then left shift. If count < 0, then right shift.

If the sign bit changes during a left shift, overflow is set.

Modifiers: X

LLU/LRU Logical Left/Right Shift U 2 8

The contents of the U register are shifted left or right C (C = count) places, with zeros filling vacated bit positions. Bits shifted past bit position 0 (left), or bit position 15 (right) are lost.

The direction of shift is determined by the count, C, after indexing. If count > 0, then left shift. If count < 0, then right shift.

Modifiers: X

RLU/RRU Rotate Left/Right U 3 8

The contents of the U register are circular shifted left or right C places. For a rotate left, bits shifted past bit position 0 are placed into bit position 15. For a rotate right, bits shifted past bit position 15 are placed into bit position 0.

No bits are lost. The direction of rotation is determined by the count, C, after indexing. If count > 0, then rotate left. If count < 0, then rotate right.

Modifiers: X

ALA/ARA Arithmetic Left/Right Shift A 4 8

The contents of the A register are shifted left/right C places. If the shift is to the left, zeros are filled into vacated bit positions on the right. If the shift is to the right the contents of bit position 0 are filled into vacated positions on the left. Bits shifted past bit position 0 (left), or bit position 15 (right) are lost.

The direction of shift is determined by the count, C, after indexing. If count > 0, then left shift. If count < 0, then right shift.

If the sign bit changes during a left shift, overflow is set.

Modifiers: X

LLA/LRA Logical Left/Right Shift A 5 8

The contents of the A register are shifted left or right C places, with zeros filling vacated bit positions. Bits shifted past bit position 0 (left), or bit position 15 (right) are lost.

The direction of shift is determined by the count, C, after indexing. If count > 0, then left shift. If count < 0, then right shift.

Modifiers: X

RLA/RRA Rotate Left/Right A 6 8 —

The contents of the A register are circular-shifted left or right C places. For a rotate left, bits shifted past bit position 15 are placed into bit position 0.

No bits are lost. The direction of rotation is determined by the count, C, after indexing. If count > 0, then rotate left. If count < 0, then rotate right.

Modifiers: X

LLUAE/LRUAE Logical Left/Right Shift UAE 0 9

The contents of the three registers U, A, E are shifted left or right C places, with zeros filling vacated bit positions. For a left shift, bits shifted past bit position 0 of E are placed into bit position 15 of the A register; bits shifted past bit position 0 of A are placed in bit position 15 of U, bit shifted past bit position 0 of U are lost. For a right shift, bits shifted past bit position 15 of E are lost.

The direction of shift is determined by the count, C, after indexing. If count > 0, then left shift. If count < 0, then right shift.

Modifiers: X

ALUA/ARUA Arithmetic Left/Shift U, A 1 9

The contents of the double U, A registers are shifted left or right C places. If the shift is to the left, bits shifted past bit position 0 of A are placed into bit position 15 of the U register; bits shifted past bit position 0 of U are lost. If the shift is to the right, bits shifted past bit position 15 of U are placed into bit position 0 of the A register; bits shifted past bit position 0 are filled into vacated positions on the left; bits shifted past bit position 15 of A are lost.

The direction of shift is determined by the count, C, after indexing. If count > 0, then left shift. If count < 0, then right shift.

If the sign bit of U changes during a left shift, overflow is set.

Modifiers: X

LLUA/LRUA Logical Left/Right Shift U, A 2 9

The contents of the double U, A registers are shifted left or right C places, with zeros filling vacated bit positions. For a left shift, bits shifted past bit position of 0 of A are placed into bit position 15 of the U register; bits shifted past bit position 0 of U are lost. For a right shift, bits shifted past bit position 15 of U are placed into bit position 0 of the A register; bits shifted past bit position of A are lost.

The direction of shift is determined by the count, C, after indexing. If count > 0 then left shift. If count < 0, then right shift.

Modifiers: X

RLUA/RRUA Rotate Left/Right U, A 3 9

The contents of the double U, A registers are circular shifted left or right C places. For a rotate left; bits shifted past bit position 0 of U are placed into bit position 15 of the A register; bits shifted past bit position 0 of A are placed into bit position 15 of the U register. For a rotate right; bits shifted past bit position 15 of A are placed into bit position 0 of the U register; bits shifted past bit position 15 of U are placed into bit position 0 of the A register.

No bits are lost. The direction of rotation is determined by the count, C, after indexing. If count > 0, then rotate left. If count < 0, then rotate right.

Modifiers: X

LLO Locate Leading One 044 2

The contents of the A register are searched and shifted to locate a leading one bit.

If the contents of A are zero, the next sequential instruction is executed.

If the contents of A are non-zero, A is shifted left until a one bit is shifted into bit position zero. Bits are zero filled from the right of A. The X register is incremented by the number of shifts that have occurred. Bit position 0 of the A register is set to zero. The next sequential instruction is skipped and execution continues with the following instruction.

Modifiers: None.

COMPARES AND TESTS

SKXEI Skip if X Equal, Immediate 14 10

Skip if (X) = LIT9

If the 9-bit literal contained in bit position 7-15 of the instruction (with sign extended) is equal to the contents of the X register, the next

sequential instruction is skipped. Otherwise, the next instruction is executed.

Modifiers: None.

SKXNI Skip if X Not Equal, Immediate 15 10

Skip if (X) ≠ LIT9

If the 9-bit literal contained in bit positions 7-14 of the instruction (with sign extended) is not equal to the contents of the X register, the next sequential instruction is skipped. Otherwise, the next instruction is executed.

Modifiers: None.

SKAE Skip if A Equal to Memory 26 1A, B, C, D

Skip if (A) = (y)

If the contents of the A register are equal to the contents of memory location y, the next sequential instruction is skipped. Otherwise, the next instruction is executed.

Modifiers: P, X, B, E, *, =

SKAN Skip if A Not Equal to Memory 30 1A, B, C, D

Skip if (A) ≠ (y)

If the contents of the A register are not equal to the contents of memory location y, the next sequential instruction is skipped. Otherwise, the next instruction is executed.

Modifiers: P, X, B, E, *, =

SKAEI Skip if A Equal, Immediate 16 10

Skip if (A) = LIT9

If the 9-bit literal contained in bit positions 7-15 of the instruction (with sign extended) is equal to the contents of the A register, the next sequential instruction is skipped. Otherwise, the next instruction is executed.

Modifiers: None.

SKANI Skip if A Not Equal, Immediate 17 10

Skip if (A) ≠ LIT9

If the 9-bit literal contained in bit positions 7-15 of the instruction (with sign extended) is not equal to the contents of the A register, the next sequential instruction is skipped. Otherwise, the next instruction is executed.

ACX Arithmetic Compare X 30(SC) 6A, G

(X) [AC] (y); Skip on Condition

The contents of the X register are algebraically compared to the contents of memory location y. If the specified skip condition is met, the next sequential instruction is skipped. Otherwise the next instruction is executed.

The variable field of the instruction may have three subfields: the skip condition, the address, and modifiers, if any.

Refer to Table 3-2 for the mnemonics and meaning of all skip conditions.

Modifiers: B, X, *, =

ACU Arithmetic Compare U 31(SC) 6A, G

(U) [AC] (y); Skip on Condition

The contents of the U register are algebraically compared to the contents of memory location y. If the specified skip condition is met, the next sequential instruction is skipped. Otherwise, the next instruction is executed.

The variable field may have three subfields: the skip condition, the address, and modifiers, if any.

Refer to Table 4-2 for the mnemonics and meaning of all skip conditions.

Modifiers: B, X, *, =

ACA Arithmetic Compare A 32(SC) 6A, G

(A) [AC] (y); Skip on Condition

The contents of the A register are algebraically compared to the contents of memory location y. If the specified skip condition is met, the next sequential instruction is skipped Otherwise, the next instruction is executed.

The variable field may have three subfields: the skip condition, the address, and modifiers, if any.

Refer to Table 4-2 for the mnemonics and meaning of all skip conditions.

Modifiers: B, X, *, =

ACE Arithmetic Compare E 33(SC) 6A, G

(E) [AC] (y); Skip on Condition

The contents of the E register are algebraically compared to the contents of memory location y. If the specified skip condition is met, the next sequential instruction is skipped. Otherwise, the next instruction is executed.

The variable field may have three subfields: the skip condition, the address, and modifiers, if any.

Refer to Table 3-2 for the mnemonics and meanings of all skip conditions.

Modifiers: B, X, *, =

FCP Floating Compare 22(SC) 6A

UAE [AC] (y, y + 1, y + 2) Skip on Condition

The normalized floating point number in UAE is algebraically compared to the normalized floating point number in memory location y, y + 1, y + 2. If the specified skip condition is met, the next sequential instruction is skipped. Otherwise, the next instruction is executed.

The variable field may have three subfields: the skip condition, the address, and modifiers, if any.

Refer to Table 4-2 for the mnemonics and meaning of all skip conditions.

Modifiers: B, X, *, =

FCPS Floating Compare, Stack 36(SC) 2B

U.A.E [AC] (T) - 3, (T) - 2, (T) -1 Skip on Condition

The normalized floating point number in UAE is algebraically compared to the normalized floating point in the Stack. If the specified skip condition is met, the next sequential instruction is skipped. Otherwise, the next instruction is executed.

The variable field may have three subfields: the skip condition, the address, and modifiers, if any.

Refer to Table 4-2 for the mnemonics and meaning of all skip conditions.

The value of the T register is unchanged in either case.

Modifiers: B, X, *, =

LCX Logical Compare X 34(SC) 6A, G

(X) [LC] (y); Skip on Condition

The contents of the X register are logically compared to the contents of memory location y. If the specified skip condition is met, the next sequential instruction is skipped. Otherwise, the next instruction is executed. Refer to Table 4-2 for the mnemonics and meaning of all skip conditions.

The variable field may have three subfields: the skip condition, the address, and modifiers, if any.

Modifiers: B, X, *, =

LCU Logical Compare U 35(SC) 6A, G

(U) [LC] (y); Skip on Condition

The contents of the U register are logically compared to the contents of memory location y. If the specified skip condition is met, the next sequential instruction is skipped. Otherwise, the next instruction is executed. Refer to Table 4-2 for the mnemonics and meaning of all skip conditions.

The variable field may have three subfields: the skip condition, the address, and modifiers, if any.

Modifiers: B, X, *, =

LCA Logical Compare A 36(SC) 6A, G

(A) [LC] (y); Skip on Condition

The contents of the A register are logically compared to the contents of memory location y. If the specified skip condition is met, the next sequential instruction is skipped.

Otherwise, the next instruction is executed. Refer to Table 3-2 for the mnemonics and meaning of all skip conditions.

The variable field may have three subfields: the skip condition, the address and modifiers, if any.

Modifiers: B, X, *, =

LCE Logical Compare E 37(SC) 6A, G

(E) [LC] (y); Skip on Condition

The contents of the E register are logically compared to the contents of memory location y. If the specified skip condition is met, the next sequential instruction is skipped. Otherwise, the next instruction is executed. Refer to Table 4-2 for the mnemonics and meaning of all skip conditions.

The variable field may have three subfields: the skip condition, the address, and modifiers, if any.

Modifiers: B, X, *, =

MSK Memory Skip 40(SC) 6A, G

(y) [AC] 0; Skip on Condition

The contents of memory location y are algebraically compared to zero. If the specified skip condition is met, the next sequential instruction is skipped. Otherwise, the next instruction is executed. Refer to Table 4-2 for the mnemonics and meaning of all skip conditions.

The variable field may have three subfields: the skip condition, the address, and modifiers, if any.

Modifiers: B, X, *, =

SKZA Skip if Zero in A 6 4B

Skip if $(A_i) = 0$

where i is a designated bit number.

Bit A_i of the A register is tested. If the bit is a zero, the next sequential instruction is skipped If the bit in A is a one, the next instruction is executed. There are two forms to the instruction.

```
SKZA  5
```

checks bit position 5 of the A register for a zero. If the bit is zero, the skip is executed. Otherwise, no skip. And,

```
SKZA  5,N
```

where N is a modifier indicates the bit number (in the example, 5) is modified by the low order 4 bits of the X register (bit positions 12-15).

Modifiers: N

SKOA Skip if $(A_i) = 1$ 7 4B

where i is a designated bit number.

Bit A_i of the A register is tested. If the bit is a one, the next sequential instruction is skipped. If the bit is a zero, the next instruction is executed. There are two forms to the instruction.

```
SKOA  14
```

skips the next sequential instruction if A_{14} is a one. And,

```
SKOA  0,N
```

modifies the bit number, 0, by the low order 4 bits of the X register to form the effective bit number, i.

Modifiers: N

SKZM Skip if Zero in Memory, y_i 6 5

Skip if $(y_i) = 0$

where i is a designated bit number.

The bit position of memory location y, designated by the bit number in the variable field, is tested. If the bit is zero, the next sequential instruction is skipped. If the bit is one, the next instruction is executed.

The variable field of the instruction contains three subfields: the bit number (absolute expression), the memory address, and modifiers, if any.

If the instruction is modified by X and not N, the memory address is modified by all 16 bits of the X register.

If the instruction is modified by N and not X, the bit number is modified by the low order 4 bits of the X register (modulo 16).

If the instruction is modified by both X and N, the memory address is modified by the high order 12 bits of the X register (bit positions 0-11); and the bit number is modified by the low order 4 bits of the X register (bit positions 12-15). If there is any carryout after modifying the bit number (15), the carry is added to the memory address calculations for y.

Modifiers: X, B, N, *

SKOM Skip if One in Memory, y_i 7 5

Skip if $(y_i) = 1$

where i is a designated bit number.

The bit position of memory location y, designated by the bit number in the variable field, is tested. If the bit is one, the next sequential instruction is skipped. If the bit is zero, the next instruction is executed.

The variable field may contain 3 subfields: the bit number (absolute expression), the memory address, and modifiers, if any.

The rules for modifiers X, and N are the same as those defined for the SKZM instruction.

Modifiers: X, B, N, *

SKNOF Skip if No Overflow 45(0) 2

Skip if (OF) = 0; 0 → OF

The overflow indicator is tested. If overflow is not set ((OF) = 0), then the next sequential instruction is skipped. If overflow is set ((OF) = 1), then the very next instruction is executed. In both cases, the overflow indicator is reset (0 OF)

Modifiers: None

SKNCO Skip if No Carryout 46(0) 2

Skip if (CO) = 0; 0 → CO

The carryout indicator is tested. If carryout is not set ((CO) = 0), then the next sequential instruction is skipped. If carryout is set ((CO) = 1), then the very next instruction is executed. In both cases, the carryout indicator is reset (0 → CO).

Modifiers: None

TSL Test and Set Lock 43(0) 6A

Skip if (y) 15 = 1 0 → y

Bit position 15 of memory location y is tested. If the bit is a one, the next sequential instruction is skipped. If bit position 15 of y is zero, the next instruction is executed. In both cases, memory location y is set to zero.

Modifiers: B, X, *

DSK Delayed Skip 47(0) 2

After the next instruction has been executed, the instruction logically following it will be skipped.

JUMPS

JMP Jump Unconditionally 11 1A, C, D

$y \rightarrow P$

The next instruction executed is determined by the effective address specified in the variable field of the instruction.

Modifiers: P, X, B, E, *

JZE Jump if A Zero 13 1A, C, D

If (A) = 0 then $y \rightarrow P$

If the contents of the A register are zero, control is transferred to the instruction specified by the variable field. Otherwise, the next instruction in sequence is executed.

Modifiers: P, X, B, E, *

JNZ Jump if A Non Zero 15 1A, C, D

If $(A) \neq 0$ then $y \rightarrow P$

If the contents of the A register are non-zero, control is transferred to the instruction specified by the variable field. Otherwise, the next sequential instruction is executed.

Modifiers: P, X, B, E, *

JPL Jump if A Plus 17 1A, C, D

If $(A) \geq 0$ then $y \rightarrow P$

If the contents of the A register are greater than or equal to zero, control is transferred to the instruction specified by the variable field. Otherwise, the next sequential instruction is executed.

Modifiers: P, X, B, E, *

JMI Jump if A Minus 21 1A, C, D

If (A) < 0 then y → P

If the contents of the A register are less than zero, control is transferred to the instruction specified by the variable field. Otherwise, the next sequential instruction is executed.

Modifiers: P, X, B, E, *

XJP X Jump 44(JC) 6A

If (X) condition met, then y → P

The contents of the X register are algebraically compared to zero. If the specified jump condition is met, control is transferred to the instruction specified by the address portion of the variable field. Otherwise, the next sequential instruction is executed.

The variable field of the instruction contains 3 subfields: the jump condition, the address, and modifiers, if any.

Refer to Table 4-2 for the mnemonics and meaning of all jump conditions.

Modifiers: B, X, *

UJP U Jump 45(JC) 6A

If (U) condition is met, then y → P

The contents of the U register are algebraically compared to zero. If the specified jump condition is met, control is transferred to the instruction specified by the address portion of the variable field. Otherwise, the next sequential instruction is executed.

The variable field of the instruction contains 3 subfields: the jump condition, the address, and modifiers, if any.

Refer to Table 4-2 for the mnemonics and meaning of all jump conditions.

Modifiers: B, X, *

AJP A Jump 46(JC) 6A

If (A) condition met, then y → P

The contents of the A register are algebraically compared to zero. If the specified jump condition is met, control is transferred to the instruction specified by the address portion of the variable field. Otherwise, the next sequential instruction is executed.

The variable field of the instruction contains 3 subfields: the jump conditions, the address, and modifiers, if any.

Refer to Table 4-2 for the mnemonics and meaning of all jump conditions.

Modifiers: B, X, *

EJP E Jump 47(JC) 6A

If (E) condition met, then y → P

The contents of the E register are algebraically compared to zero. If the specified jump condition is met control is transferred to the instruction specified by the address portion of the variable field. Otherwise, the next sequential instruction is executed.

The variable field of the instruction contains 3 subfields: the jump condition, the address, and modifiers, if any.

Refer to Table 4-2 for the mnemonics and meaning of all jump conditions.

Modifiers: B, X, *

TJP Triple Jump, UA 50(JC) 6A

If (U), A, E) condition met, then y → P

The contents of the triple U, A, E registers are algebraically compared to zero. If the specified condition is met, control is transferred to the instruction specified by the address portion of the variable field. Otherwise, the next sequential instruction is executed.

The variable field of instruction contains 3 subfields: the jump condition, the address, and modifiers, if any.

Refer to Table 4-2 for the mnemonics and meaning of all jump conditions.

Modifiers: B, X, *

IJXN Jump if Non Zero, Increment X 25 1A, C, D

If (X) ≠ 0 then y → P; (X) + 1 → X

If X is non-zero, control is transferred to the instruction specified by the variable field. Otherwise, the contents of the X register are incremented by one and the next sequential instruction is executed.

Modifiers: P, X, B, E, *

DJXN Jump if Non Zero, Decrement X 27 1A, C, D

If (X) ≠ 0 then y → P; (X) - 1 → X

If X is non-zero, then control is transferred to the instruction specified by the variable field. Otherwise, the contents of the X register are decremented by one and the next sequential instruction is executed.

Modifiers: P, X, B, E, *

IJMP Indirect Jump 55(0) 6A

(y) → P

The address of the next instruction to be executed is determined by the effective address specified in the variable field of the instruction.

Modifiers: B, X, *

SUBROUTINE AND SYSTEM LINKAGE

JSPX Jump, Store P in X 23 1A, C, D

(P) + 1 → X; y → P

The contents of the P register plus one, are stored in the X register. Control is transferred to the instruction specified by the variable field.

Modifiers: P, X, B, E, *

JSPM Jump, Store P in Memory 56(7) 6A

$$(P) + 1 \rightarrow (y); \quad y + 1 \rightarrow P$$

The contents of the P register plus one, are stored in the location specified by memory location y. Control is transferred to memory location y + 1.

Modifiers: B, X, *

CALL Subroutine Call Linkage 63(7) 6A

$$(P) + 1 \rightarrow (T); (B) \rightarrow (T) + 1; (T) + 2 \rightarrow B \rightarrow T; \quad y \rightarrow P$$

The contents of the P register plus one, is stored in the location specified by the top of the stack pointer T. The contents of the base of stack B register is stored in the location specified by the T register plus one. The address, specified by the contents of the T register plus two, is stored in both the B and T registers. Control is transferred to the instruction specified by the variable field.

In this way, the return location from the subroutine, and the original values of the B and T registers are preserved. Various subroutine stack pointers are automatically protected from routine to routine.

A stack overflow trap occurs if $(T) \geq (L)$.

Modifiers: B, X, *

RTRN Subroutine Return Linkage 050 2

$$(B) - 2 \rightarrow T; \quad ((T) + 1) \rightarrow B; \quad ((T)) \rightarrow P$$

The contents of the base of stack B register minus two, is stored in the T register. This restores the top of stack T to the value at the time of the subroutine CALL. The contents of the value contained in the top of stack pointer T plus one, is stored in the B register. This restores the value of the B register to the value at the time of the subroutine CALL.

Control is transferred to the memory location contained in the T register. This effectively returns control to the instruction following the subroutine CALL with the values of the B and T registers restored.

A stack underflow track occurs if $(T) < (B)$.

Modifiers: None.

SCALL System Call – 3 –

System Calls are those calls concerned with defining, determining, and manipulating the system environment of a process.

There are limited number of System Calls simulated by Tymshare. These calls are primarily used to:

- open and close files
- I/O to and from files, TTY
- manipulate edited lines

Most of the calls will have counterparts in the final version of the LOGICON 2 + 2 assembler. They are identified in Table 4-4 with mnemonics. Some calls are unique to the Tymshare version, and, as such, no mnemonics are defined for them.

There are two allowable forms to the System Call instruction. The general form is:

(label) SCALL (System Call no.)

Note that System Call numbers are octal. All existing (simulated) calls can be made via this form. Those calls having a unique mnemonic may be accessed by the name as:

(label) (System Call Name)

For example, the Get Edited Line System Call, GEL, may be written

(label) SCALL 204B

or, (label) GEL

Table 4-4 shows all of the simulated System Calls, characteristics, and any anomalies. For a detailed description of each of the System Calls, the user is referred to the LOGICON 2+2 MONITOR SPECIFICATION.

Modifiers: None.

IJSPX Indirect Jump, Store P in X 55(1) 6A

$(P) + 1 \rightarrow X$; $(y) \rightarrow P$

The contents of the P register plus one, is stored in the X register. Control is transferred to the address specified by the variable field.

Modifiers: B, X, *

IJSPM Indirect Jump, Store P in Memory 55(2) 6A

$(P) + 1 \rightarrow ((y))$; $(y) + 1 \rightarrow P$

The contents of the P register plus one, is stored at the address specified by memory location y. Control is transferred to the location following the address specified by the variable field.

Modifiers: B, X, *

ICALL Indirect Subroutine Call Linkage 55(3) 6A

$(P) + 1 \rightarrow (T)$; $(B) \rightarrow (T) + 1$; $(T) + 2 \rightarrow B \rightarrow T$; $(y) \rightarrow P$

The contents of the P register plus one, is stored in the location specified by the top of the stack pointer T. The contents of the base of stack B register is stored in the location specified by the T register plus one. The address, specified by the contents of the T register plus two, is stored in both the B and T registers. Control is transferred to the address specified by the variable field.

In this way, the return location from the subroutine, and the original values of the B and T registers are preserved. Various subroutine stack pointers are automatically protected from routine to routine.

A stack overflow trap occurs if $(T) \geq (L)$.

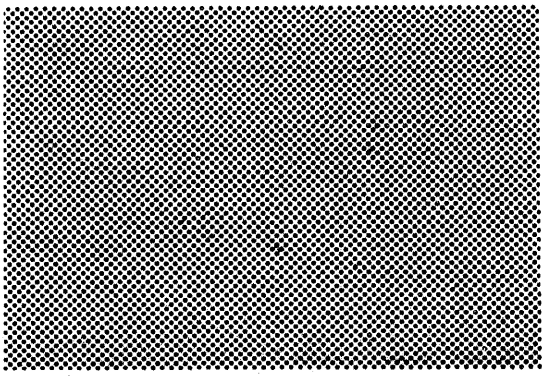
Modifiers: B, X, *

TABLE 4-4. SIMULATED SYSTEM CALLS

Name		Number	Input	Output	Function
TCI	Terminal Character Input	200_8	-----	$(A)_{8-15}$ = character	Input one character from TTY
TCO	Terminal Character Output	201_8	$(A)_{8-15}$ = character	-----	Output one character to TTY
SLEM	Set Line Editing Mode	202_8	Same as 2 + 2 spec except (A) and (U) ignored	-----	Same as 2 + 2 spec
LLEM	Leave Line Editing Mode	203_8	-----	-----	Same as 2 + 2 spec
GEL	Get Edited Line	204_8	Same as 2 + 2 spec	Same as 2 + 2 spec	Same as 2 + 2 spec
BEL	Build Edited Line	205_8	Same as 2 + 2 spec	Same as 2 + 2 spec	Same as 2 + 2 spec
GSEL	Get Status of Edited Line	206_8	Same as 2 + 2 spec	Same as 2 + 2 spec	Same as 2 + 2 spec
--	Open File for Input	207_8	(U, A) = string pointers	(A) = file no. (X) = file size (U) = file type	Open specified file for input. Skip return if O.K.

TABLE 4-4. SIMULATED SYSTEM CALLS (Cont)

	Name	Number	Input	Output	Function
--	Open File for Output	210_8	(U,A) = string pointers. (X) = file type	(A) = file no.	Open specified file for output. Skip return if O.K.
--	I/O character to or from file	211_8	(U) = file no. (A) = character if file open for output	(A) = character if file open for input	Input or Output one character from or to file
--	Close File	212_8	(U) = file no.	-----	Close specified file.
--	Simulated Drum Call	213_8	(A) = drum block number (max. 200_{10}). (U) = core block number (unmapped and always in CPU regardless of from which processor the call is executed). (X) = 0 read from drum. (X) ≠ 0 write on drum.	-----	Reads or writes 512 words from/ to the simulated drum. Skip return if O.K. Illegal instruction trap if ;L LSIM command not executed before SCALL 213B.



V...

Input/Output

GENERAL

The input/output equipment with which the LOGICON 2+2 processors interface is of two types: 1) storage devices, consisting of magnetic tape transports, disk drives, and a high-speed drum, and 2) user communication terminals, of both the low-speed (asynchronous) interactive and the high-speed (synchronous) batch processing varieties.

The tape and disk storage devices serve as storage for program and data files. Both of these devices, as well as the communications terminals, are driven from the Peripheral Processor (PP). The drum is used for swapping selected portions of active user program and data files into and out of main core memory, and since its role in overall user response and system performance is so critical, it is driven by a processor dedicated to that end, the Drum Processor (DP).

The PP interfaces directly with a tape controller and a disk controller which act as interfaces between the processor and storage devices themselves. The tape controller accommodates from one to eight tape transports, the disk controller between 1 and 8 disk drives. The PP interfaces with the communication terminals by way of either a synchronous communications adapter or an asynchronous communications adapter. Each synchronous adapter handles a group of four full duplex channels; each synchronous adapter handles 16 full duplex channels. If the terminals are remotely located, which is usually the case, they are additionally separated from the processor by common carrier lines and a MODEM at each end of the connection.

The DP is itself a controller, is hard-wired rather than microprogrammed, and interfaces directly with the drum storage device.

The input/output transactions and associated control processing of both the PP and the DP are supervised and indirectly managed by the Control

Processor (CP). The controlling and controlled processors communicate with each other through specified, shared, core memory locations and by a variety of status interrupts generated by the controlled processors. The relationships of the PP and DP to the CP are essentially analogous, as will be shown.

CP SOFTWARE INTERFACES

User processes being executed in the Applications Processor (AP) which desire to perform an input/output transaction with a user terminal or with the file system are provided with a set of System Calls which can be invoked for the device and for the purpose intended; these I/O commands are described in detail in the Resident Monitor specification. When such a System Call is made, the Monitor software in the CP establishes one or more Buffer Control Blocks (BCB) in CP core memory and uses the IOC command to set up a Channel Control Call (CCC), also in CP core. The Monitor program, of course, responds to many such I/O requests and must coordinate both the performance and the results of these transactions. Therefore, in a disk transaction, for example, the Monitor maintains "bookkeeping" responsibility for such things as the availability, assignment and distribution of data pages and for the scheduling of read/write requests.

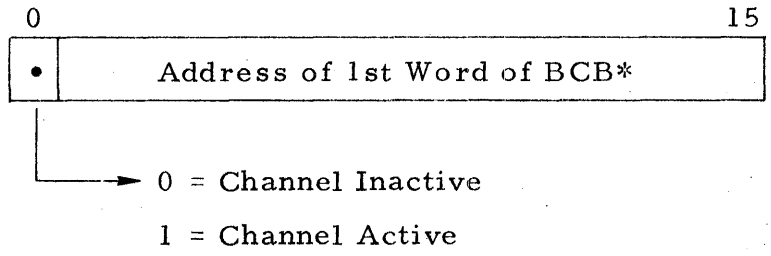
In a similar manner, the CP Monitor program schedules drum swapping transfers to and from AP core memory, along with performing all necessary bookkeeping and cleanup operations. Drum Control Blocks (DCB) and Current State Calls (CSC) are established in AP core for control and communication with the DP in a manner analogous to the BCBs and CCCs set up for the PP. In this case, however, the initiative for drum input/output rests solely with CP software, concurrent with, but not originating from the execution of user processes.

Peripheral Processor (PP)

The following paragraphs describe the CCC, BCB and interrupt mechanisms through which the CP/PP interface is implemented.

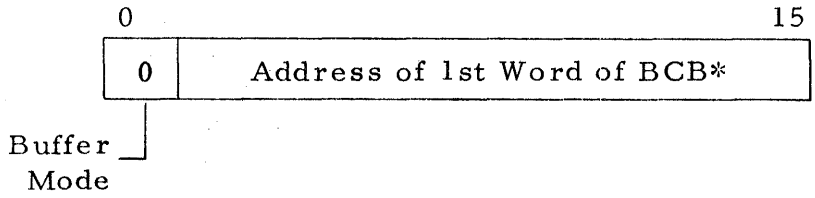
Channel Control Cells (CCCs). The CCCs occupy dedicated locations in CP memory: those for the tape channel are in cells 100_8-107_8; those for the disk channels are in cells 110_8-117_8; those for the communication channels follow next and are divided into a group of input CCCs followed by a group of output CCCs, each channel having a cell of each type. The total number of CCCs is then 8 + 8 + 2 (16m + 4n), where m is the number of asynchronous adapters and n the number of synchronous adapters in a given system configuration.

The word formats for the tape and disk CCCs are identical:



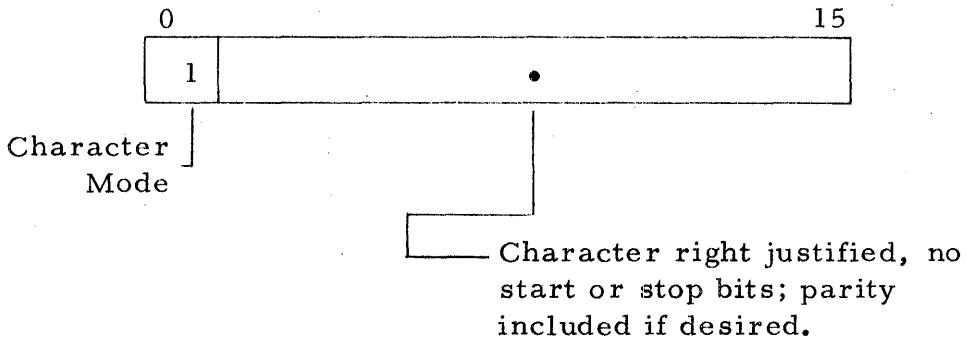
TAPE
or
DISK

Communications channels have two CCC formats, since communications may occur in either the buffer mode or character mode. In the buffer mode, input and output CCCs are identical



COMM

In the character mode, the format:

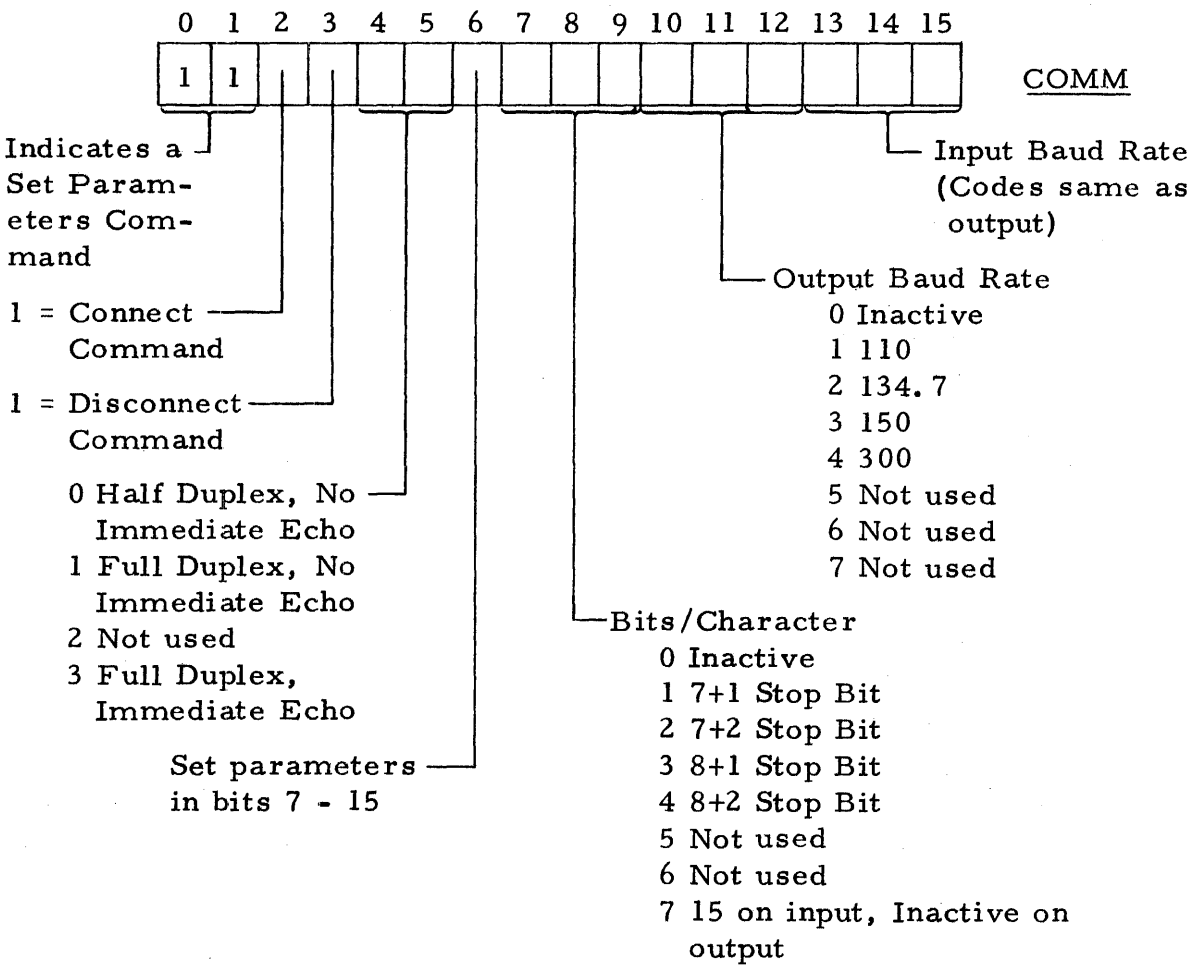


COMM

The above buffer and character mode CCCs are used to start input or output. To halt communications, CP software will set the appropriate CCCs to "all zeros."

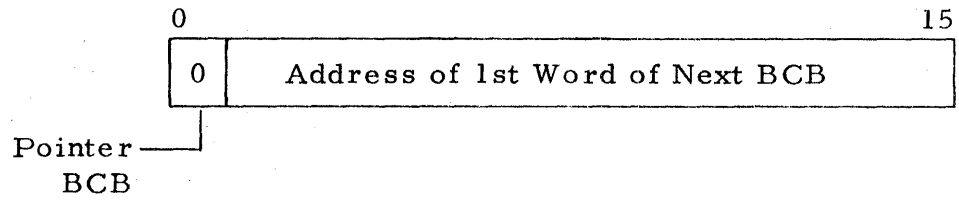
*Set initially by CP to denote 1st BCB to process; reset and maintained by the PP thereafter to denote BCB currently being processed.

In addition, when communications with a given communications channel are first being established and the terminal type and BAUD rate are in the process of being determined, CP software uses the output CCC for that channel to convey the setting of parameters to the PP. The set parameters CCC format is as follows:



Buffer Control Blocks (BCB). Associated with each tape and disk CCC, and with each communications CCC specifying buffered input or output, is one or more Buffer Control Blocks (BCB). BCBs may be placed anywhere in core memory by CP software; successive BCBs may or may not be located contiguously.

When BCBs are not contiguous, a special one-word BCB is used to point to the next BCB to be processed. This pointer BCB is used for tape, disk and communications I/O and has the format



The normal or "working" BCBs for tape, disk and communications are generally dissimilar in format, content and number of words per block. Figures 5-1, 5-2, and 5-3 give the normal BCB configuration for communications, tape and disk, respectively.

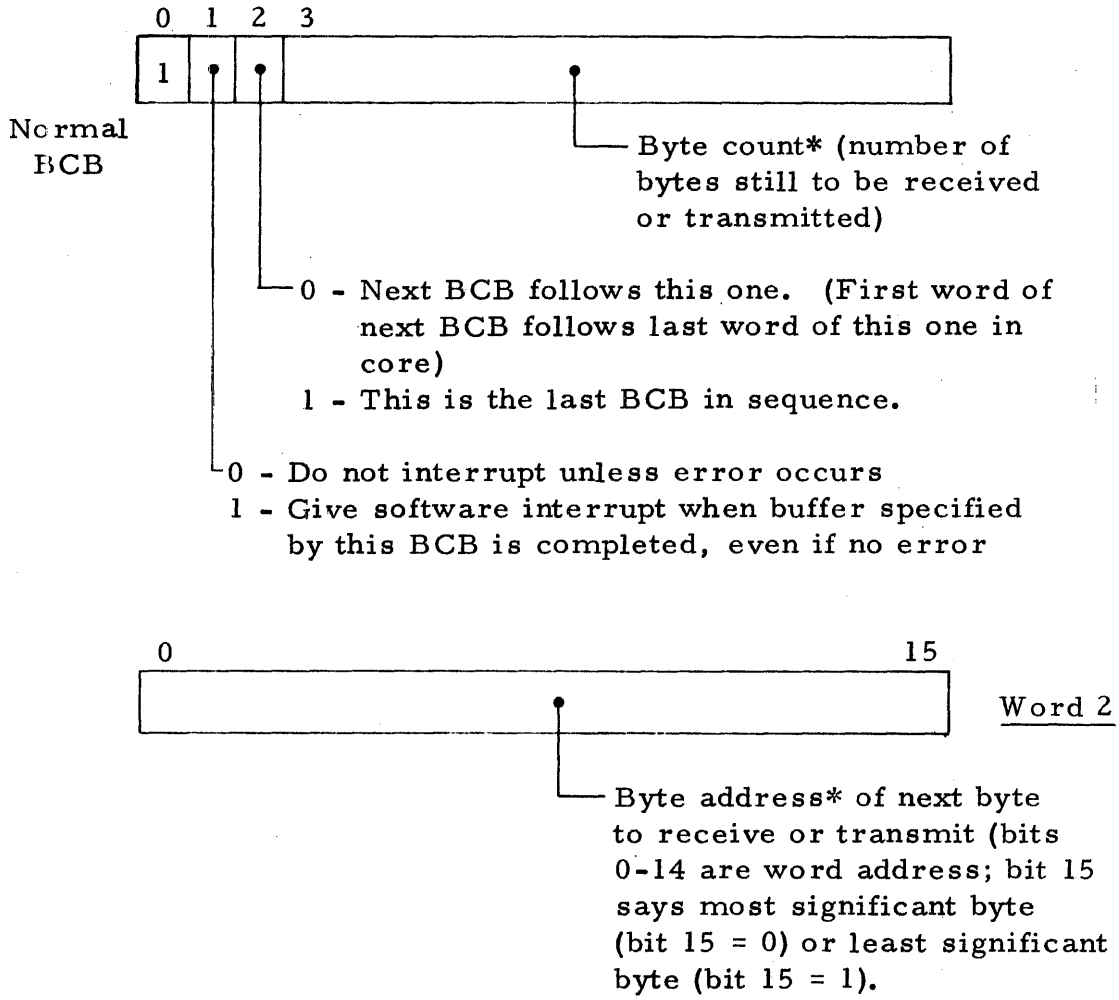
NOTE

The first three bits in word 1 of the BCB are defined in the same way for all three input/output mediums handled by the PP.

The disk interface utilizes a "Code Word" (CW) and an "Integrity Word" (IW). Both of these special-purpose words are associated with each 512-word page, of data transferred to and from main core memory and are used in near-identical fashion by the PP (disk) and DP (drum) for error detection and validity checking. Briefly, they have the following characteristics:

- Code Word — a software-defined "label" that immediately precedes every data page written out; it can be used for page identification or linkage or as a file key or check word and may be read or written independently from a data page. It is used by the 2+2 Monitor as an independent check on the access protection system.

- Integrity Word — stored with every page, it represents a computed check-sum over the 1024 8-bit bytes of data; in read operations it is compared to an internally-generated (PP or DP) check sum; in write operations it may be either generated on-line or optionally pre-specified.



*The byte count is initially set by the CP to the length of the input or output buffer; similarly, the CP sets the byte address to that of the first byte in the input or output buffer. Thereafter, the PP adjusts these parameters to reflect its progress in prosecuting the BCB.

Figure 5-1. Normal Communications BCB Format

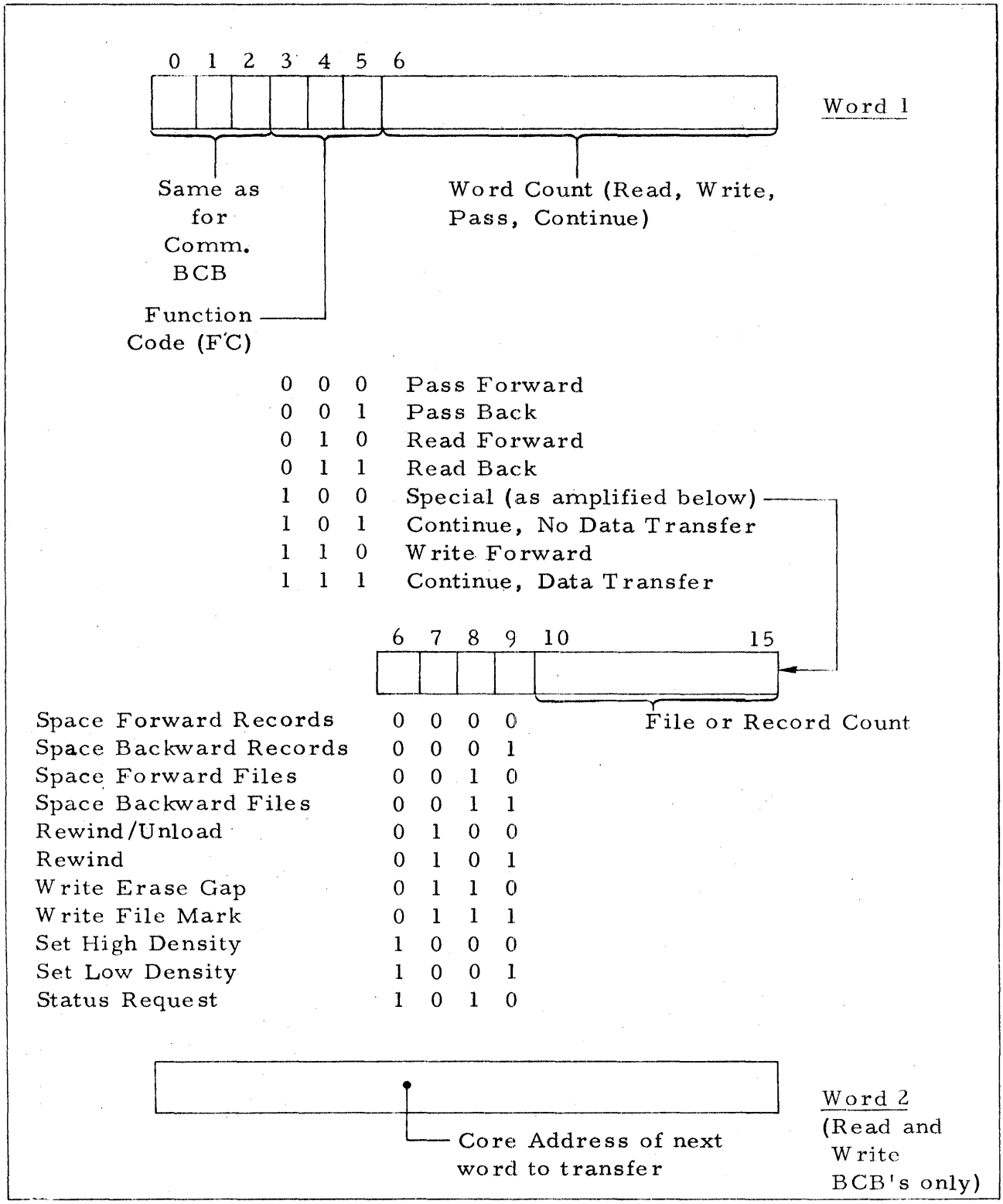
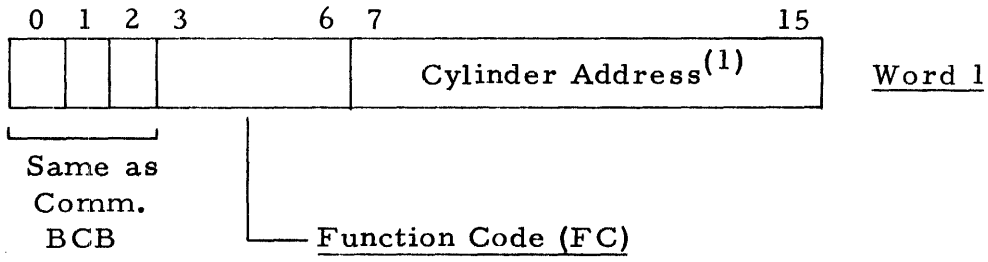


Figure 5-2. Normal Tape BCB Format



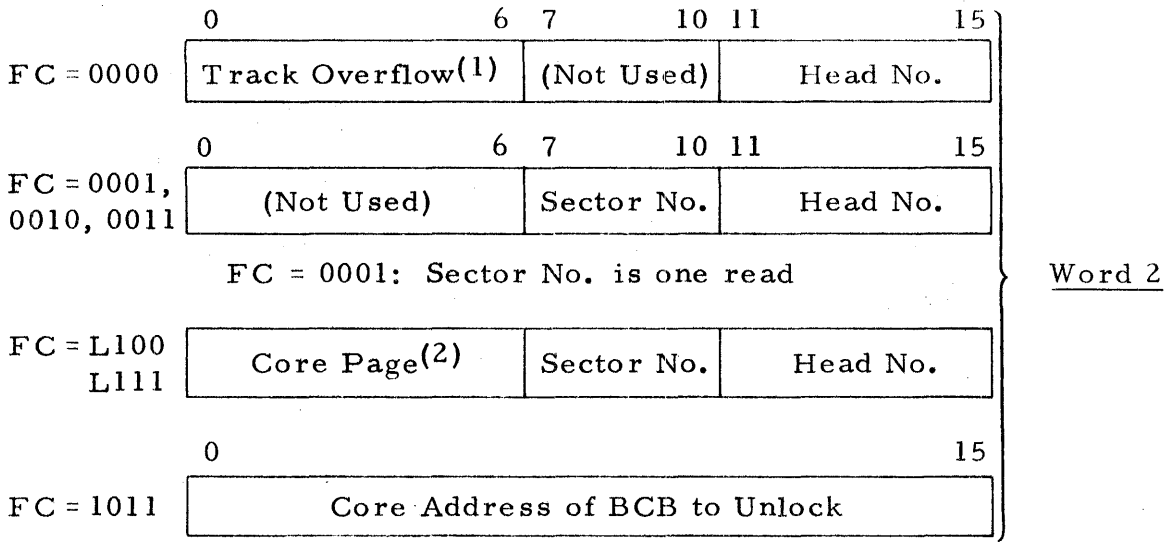
0	0	0	0	-	Initialize Track (Address Sectoring)
0	0	0	1	-	Read Sector
0	0	1	0	-	Write CW, Zero Data
0	0	1	1	-	Read CW; Compare CW
L	1	0	0	-	Read Data; Compare CW and IW
					(L = 1 means BCB is locked)(2)
L	1	0	1	-	Read Data and IW; Compare CW and IW
L	1	1	0	-	Write Data and CW
L	1	1	1	-	Write Data, CW and IW; Compare IW
1	0	0	0	-	(Not Used)
1	0	0	1	-	Seek
1	0	1	0	-	(Not Used)
1	0	1	1	-	Unlock (BCB(2)

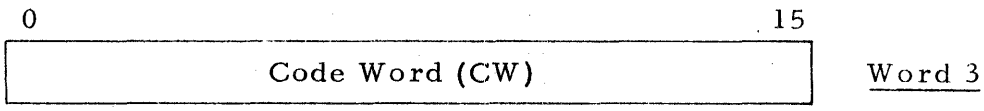
NOTES

(1) FC = 1001: This field not used.

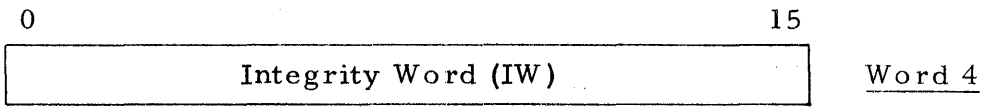
(2) In scheduling disk I/O transfers (e.g., reads/writes of same data involving different disk drives), a given BCB can be "locked", i.e., blocked in the BCB processing sequence. When unlocked by another BCB (FC = 1011), it will be processed in the normal sequence.

Figure 5-3 (sheet 1 of 2). Normal Disk BCB Format





FC = 0011, L100, L101: This CW compared to one read.
FC = 0011: After comparison, this CW is replaced by CW read.
FC = 0010, L110, L111: CW to be written out.



FC = L101: IW as read.
FC = L111: IW to be written out; after comparison, this IW is replaced by IW calculated.

NOTES

(1) During track initialization, the sector word count is given in this field. The number of words that fit in a sector varies with disk rotation rate, from 546 to 569. The CP creates the BCD with this field containing the sector word count less 546 (0-23). After execution, the PP stores the number of words of overflow (positive if all words did not fit, negative if room left over).

(2) The seven most significant bits of memory address; the low-order nine bits correspond sequentially to each of the 512 words in the page to be transferred.

Figure 5-3 (sheet 2 of 2). Normal Disk BCB Format

Interrupts Generated by PP

As the PP prosecutes a BCB list, it generates interrupts to the CP as requested by an individual BCB and/or when error conditions occur. Separate interrupt lists are maintained in CP core memory for the tape, disk and communications I/O mediums. Each cell in a list corresponds to an I/O channel, and PP firmware mechanizes the equivalent of a LINK instruction to link list entries in FIFO order. Three hardware interrupt lines are utilized between the PP and CP for disk, communications and tape I/O (CP interrupt bits 4, 5 and 6, respectively). When the CP receives a hardware interrupt on one of these lines, it goes to the corresponding software interrupt list and uses a DLINK instruction to access the oldest interrupt.

The tape, disk, and communications interrupts provided by the PP, and their respective word formats are shown in Figure 5-4. In the actual software interrupt lists, a cell entry will have one or more interrupts activated in the upper byte and either 377_8 or a pointer to the next waiting channel interrupt in the low order byte. The meaning and use of these is obvious in some cases and in others will become more apparent when the interface between the PP and the tape and disk controllers and the communications adapters is discussed.

Drum Processor (DP)

Swapping operations between the drum and main AP core memory are initiated by the CP, via an interrupt to the DP; once started, it is anticipated that the CP will maintain the drum in the active state at virtually all times thereafter. Receipt of the interrupt causes the DP to access a two-word Current State Cell (CSC) in AP core. The CSC occupies cells 26_8 and 27_8 in AP memory.

The format and content of the CSC are shown in Figure 5-5. It will be noticed that the CSC is more comprehensive than are the CCCs used by the PP; in addition to the necessary Drum Control Block (DCB) pointer, the CP uses the CSC to dictate the current operating state of the DP, and the DP, in turn, reflects its own operating status and progress in this cell.

One or more four-word DCBs will be associated with a given CP setting of the CSC; each 512-word page of data to be swapped will have a unique DCB specifying the address, integrity word and instructional information

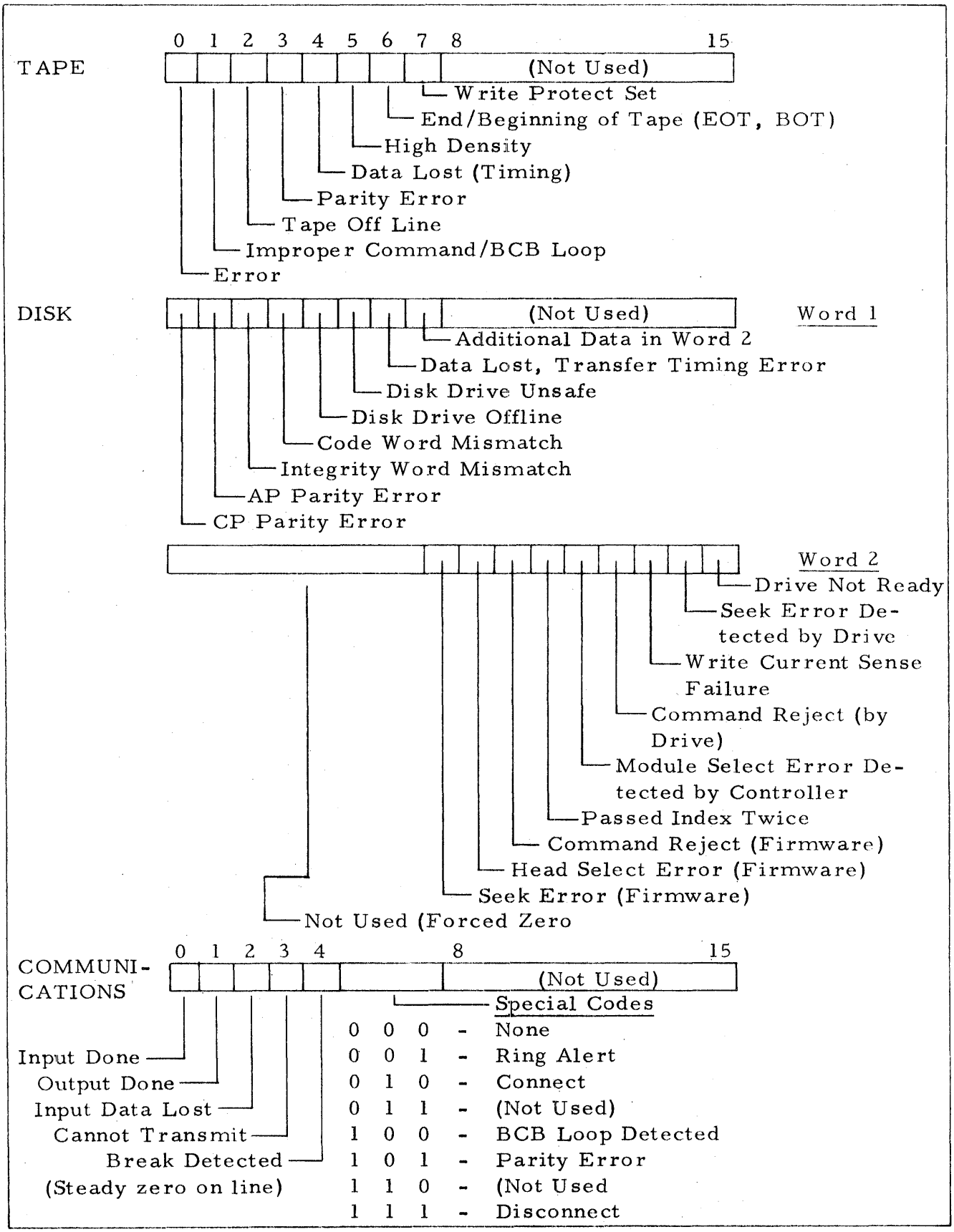


Figure 5-4. PP Generated Interrupt Formats

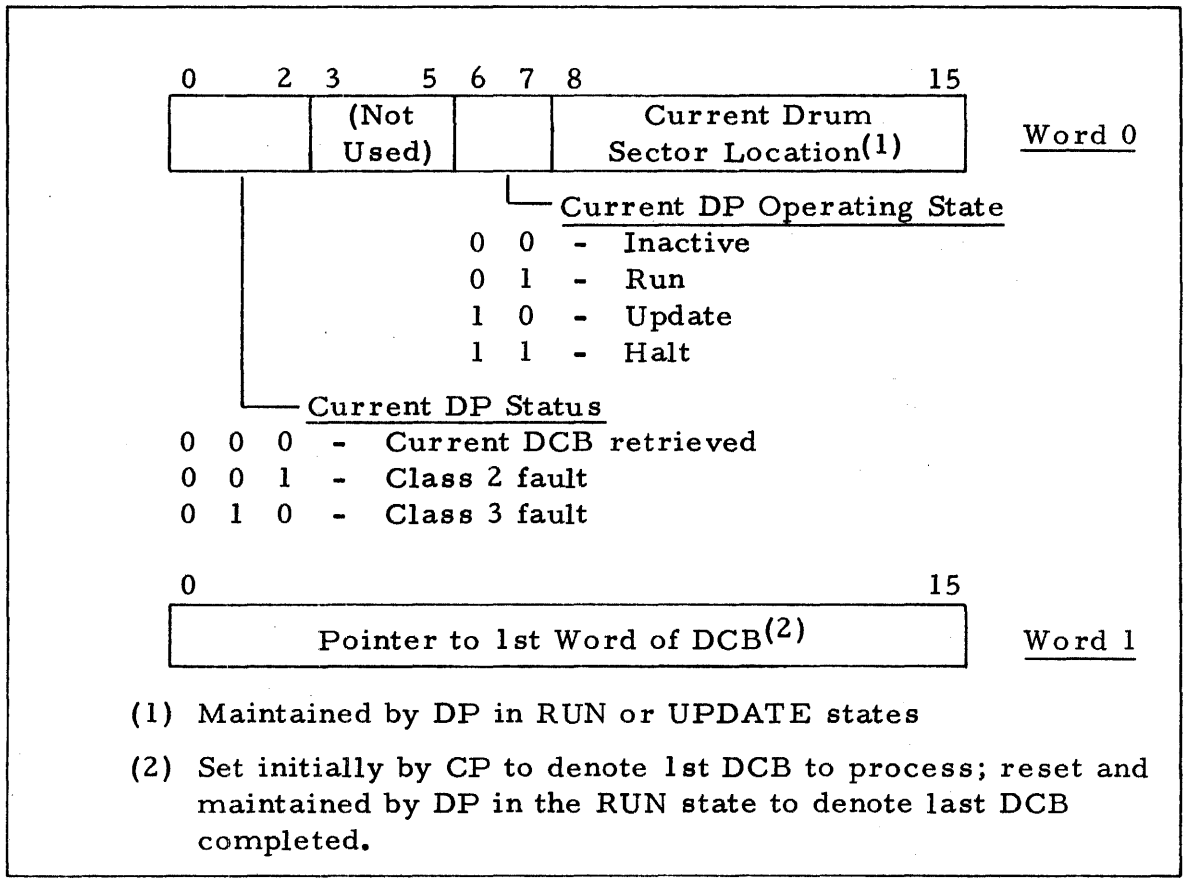


Figure 5-5. Current State Cell (CSC) Format

necessary to accomplish the transfer. The DCBs may be placed anywhere in AP core memory by the CP and may be listed sequentially (up to 128 maximum) or linked by pointers one to the next. Figure 5-6 gives the DCB format and content.

As is true for the CSC, the DCB is more extensively utilized as a software interface between the CP and CP than are the previously discussed BCBs relative to the CP/PP interface. There is, rather naturally, a marked similarity between the read/write function code repertoires and Code/Integrity Word mechanication of the DP-drum and PP-disk. However, the most important difference exhibited by the DCB is the multipurpose nature of the last word, especially in regard to its specification of error status used in conjunction with the interrupt enabler, this compares with the software interrupt list implemented between the CP and PP for each of the I/O device types handled by the PP.

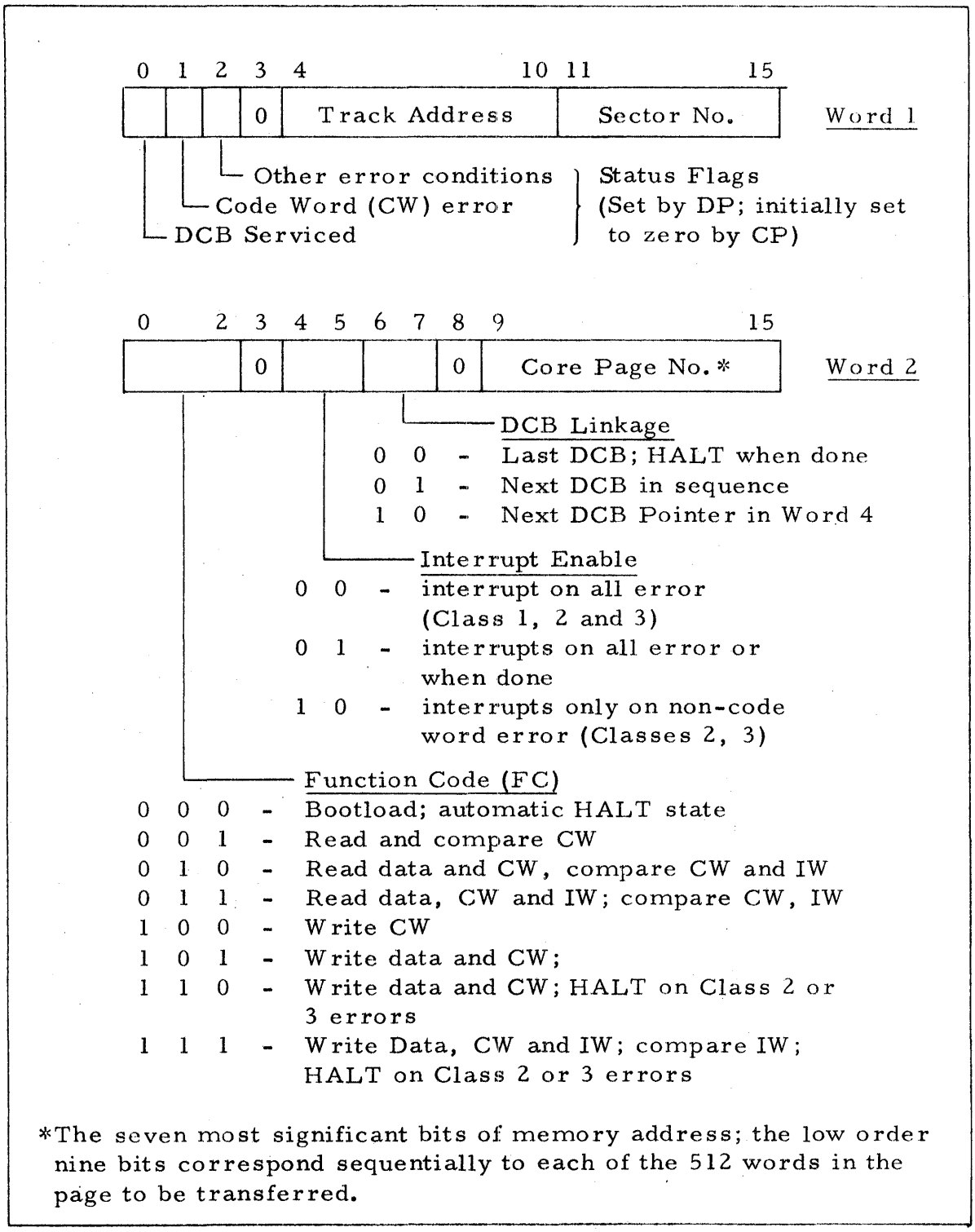


Figure 5-6 (sheet 1 of 2). Drum Control Block (DCB) Format

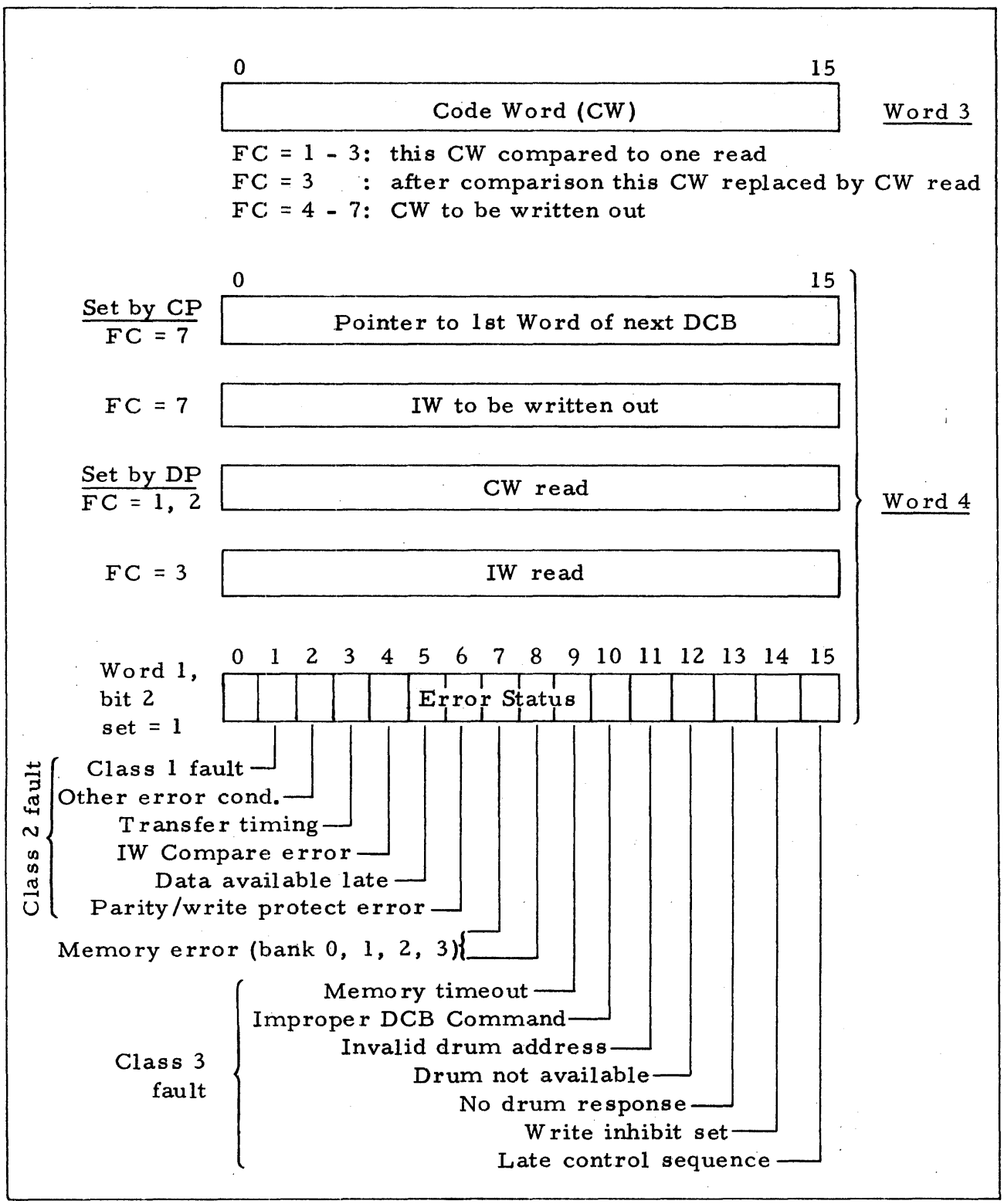


Figure 5-6 (sheet 2 of 2). Drum Control Block (DCB) Format

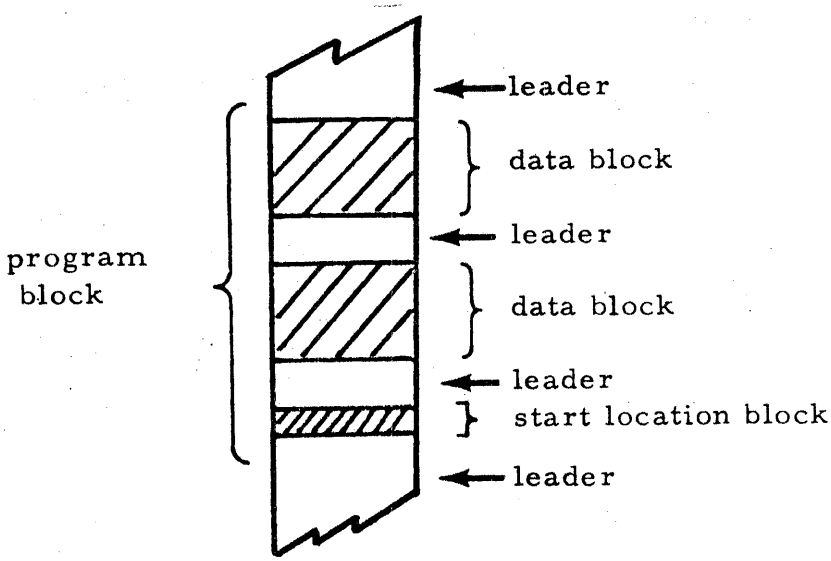
It has already been mentioned that the CP interrupts the DP to initiate swapping operation. The CP may also interrupt the DP while it is in the RUN state, in which case the DP goes into UPDATE state. The DP will generate an interrupt to the CP (input interrupt bit 3) when it has completed processing the last DCB in a given sequence or when an error condition is encountered for which the associated DCB directed that an interrupt be enabled. In the former case, the DP will go automatically into the UPDATE state; in the latter case the DP will always remain in the RUN state for class 1 (CW compare) errors, will always go into the HALT state for class 3 errors, and will remain in the RUN state for class 2 errors unless FC = 6 or 7 in DCB (word 2).

Appendix **A...**

Bootstrap Formats

PAPER TAPE BOOTSTRAP FORMAT

A formatted paper tape for a bootstrap load consists of one or more data blocks followed by a start location block:



A data block consists of a word count ($1\text{-}376_8$ in one frame), an initial address (two frames), a number of words (two frames per word) to be loaded at (initial address) to (initial address + count -1), and a checksum (two frames) that makes the whole block sum to zero (including the word count). Thus, a data block to load 025677 in cell 016254 would be:

001	count
034	address byte 1
254	address byte 2
053	data byte 1
277	data byte 2
267	checksum byte 1
224	checksum byte 2

(000001 + 025677 + 016254 + 133624 = 000000)

A start location block consists of a word count of 377 followed by a start location (two bytes). Thus, a start location block to cause a start at cell 1000 would be:

377	word count (flag)
002	address byte 1
000	address byte 2

MAGNETIC TAPE BOOTSTRAP FORMAT

A magnetic tape record for bootstrap load consists of a 1001_8 word record (1000 data words plus a checksum that makes the record sum to zero).

A magnetic tape bootstrap start causes one such record to be loaded in locations 00000 to 777 of CP memory, and the CP P register to be set to the resulting contacts of cell Ø.