DOCUMENT NO. KD72-21 REV _____

PRINCIPLES OF OPERATION

SKC-2000 (FOCUS) COMPUTER

ENGINEERING TECHNICAL REPORT

SINGER
AEROSPACE & MARINE SYSTEMS

THE SINGER COMPANY • KEARFOTT DIVISION • 1150 MCBRIDE AVENUE • LITTLE FALLS, N. J. 07424

Document No. KD-72-21
February 1972

PRINCIPLES OF OPERATION

SKC-2000 (FOCUS) COMPUTER

Prepared by:

Computation and Programming Department

February 1972

Document No. KD-72-21
February 1972

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Document No. KD-72-21
February 1972

THE SINGER COMPANY • KEARFOTT DIVISION

PRINCIPLES OF OPERATION

SKC-2000 (FOCUS) COMPUTER

ABSTRACT

This document describes the operation of the SKC-2000 Computer from a programming point of view. In conjunction with the FOCAP Language Reference Manual (Document No. KD 71-60) and the FOCAP Users' Manual (Document No. KD 72-18), it provides the necessary information for preparing SKC 2000 computer programs.

1. INTRODUCTION

A general purpose, high performance digital computer, the SKC-2000 (FOCUS) bases its modular design upon a single data and control bus interconnecting all modules with a standard interface to provide true modularity and maximum flexibility. Modular flexibility is further augmented by asynchronous module operation, a complete spectrum of input/output capabilities, and contemporary mechanical design utilizing sandwich construction which is easily expandable.

SKC-2000 modules' strict compliance with asynchronous modularity concepts permits module mixing or matching, and growth by adding modules or through replacement with modules using newer technologies.

The Principles of Operation Manual is intended to provide sufficient information on the operation of the computer to permit a qualified programmer to successfully develop an SKC-2000 program. Two companion documents on FOCUS Assembler Program (FOCAP) are also required. These are the FOCAP Language Reference Manual (KD-71-60) and the FOCAP Users' Manual (KD -72-18).

Before discussing the arithmetic capability of the SKC-2000, it is fitting to discuss briefly the principal characteristics of its various hardware components. The primary memory component is a ferrite core memory module of 9192 (32 bit) words with a 1.9 μsec cycle time and a 0.75 μsec access time. Designed as independent, asynchronous modules, one or more of them can be connected to the common bus without impacting the other units sharing the bus. The basic machine also includes a Read/Write LSI circuitry memory of 256 (32 bit) words with an effective cycle time of 0.5 μsec. Optional memory types include Read Only Memory (ROM) with a very high packing density, low power, and high speed (100 n sec access time). The latter can be used to store the Common Subroutine library for further execution speed enhancement. The computer has an addressing capacity of 131, 072 (32 bit) words, most of which can be designated as fixed or Write Protected memory. Consequently, the use of a Read Only LSI circuitry memory blends well with the SKC-2000 Computer's logical architecture.

A BITE card is included in the basic computer to provide continuous monitoring of the performance of the computer and signals and occurance of a detected failure.

A wide variety of input/output equipment has been developed by Kearfott for the SKC-2000 and other production avionic systems, not the least of which is necessary digital and analog interface to permit the SKC-2000 to operate an inertial navigation subsystem using the KT70 series of Kearfott inertial platforms. Since I/O facilities usually depend strongly on the nature of the application, they will not be discussed further here.

The packaging techniques employed in the SKC-2000 are both novel and successful. A stacked and clamped card mounting technique is employed in which the cards form an integral "box like" structure for the overall assembly. Four longitudinal members are used to clamp the cards between a front connector panel and rear support panel. Designed to be hard-mounted to eliminate the need for vibration isolators, the computer has successfully passed a series of extensive vibration tests.

Finally, the summary physical characteristics of a typical SKC-2000 configuration are listed below:

1/2 ATR cross section, variable length

Cooling via forced air (cold plate also possible)

Dimensions = 15.33 in. x 7.50 in. x 4.88 in.

Power = 241 Watts

Weight = 19.7 lbs.

2. CENTRAL PROCESSING UNIT

Parallel organization and floating point arithmetic in the Central Processing Unit provides the SKC-2000 with outstanding arithmetic capabilities.

The processor operates on 16 bit, 32 bit, and (to a limited extent) 64 bit operands. It is thus capable of exceptionally high accuracy as well as efficient memory utilization. Sixty index registers are supplied in high speed circuitry which facilitate matrix operations and the processing of tabular data. The basic CPU includes a volatile fast scratchpad memory of 256 words which assures sufficient capacity for large computation loads.

Efficient memory utilization is enhanced by a very powerful short instruction capability. Most instructions have both a long (32 bit) and short (16 bit) format. Experience shows that over 80% of the instructions in a representative program can be short. Typically, instruction use the greater part of memory. Hence, packing two instructions per (32 bit) memory word results in a greatly increased number of instructions for a given memory size. For example, 10 K short instructions and 2K words of data can be accomodated in an SKC-2000 memory of 8K (32 bit) words.

A more detailed description of CPU operation is contained in the following subsections.

2.1 REGISTERS

The Central Processing Unit contains several registers as listed below with their abbreviated designations in parentheses:

1. Upper Accumulator Register (A) - 32 bits
2. Lower Accumulator Register (B) - 32 bits
3. Instruction Register (IR) - 32 bits
4. Program Counter (PC) - 18 bits
5. Operand Address Register (OAR) - 18 bits
6. Index Registers (XR) - 18 bits
7. Status Register (SR) - 16 bits
8. Interrupt Mask Register (MR) - 16 bits
9. Concatenation of A and B Registers (A,B) - 64 bits.

2.1.1 A Register

This 32 bit register receives the result of most instructions and the most significant results of multiply instructions. The contents of the accumulator may be shifted right or left. It is accessible through the Aerospace Ground Equipment connector (AGE).

2.1.2 B Register

This 32 bit register receives the result of Add and Subtract Lower instructions, the least significant results of multiply instructions, and the least significant portion of the dividend for fixed point divide instructions. The contents of the lower accumulator may be shifted right or left in conjunction with the upper accumulator. It is accessible through AGE.

2.1.3 Instruction Register

This 32 bit register is used to hold the current instruction. It can hold two instructions if they are both short. It is accessible through AGE.

2.1.4 Program Counter

This 18 bit register contains the Address of the instruction being executed. Upon execution of a long instruction, the PC is normally incremented by 2.

2.1.5 Operand Address Register

This 18 bit register contains the effective address of the instruction operand. See Section 2.2.1 for a discussion of effective address.

2.1.6 Index Registers

A total of 64 index registers are provided. These registers are implemented in LSI circuitry and are organized in four groups of sixteen. A particular group is selected by two bits in the status register (SR0, SR1). Within a group, the sixteen registers are designated XR0, ..XR15.

Each group of sixteen consists of XR0, seven first level registers (XR1, ..XR7) and eight second level registers (XR8, ..XR15). Second level registers may only be referenced by long instructions. Except for XR0, they may all be used to modify the operand address field of arithmetic instructions, thus making 60 registers available for indexing. The XR0 registers may only be referenced by the following instructions: LDX, STX, ICN, ICL, IMP, IMN, CLS and LSC.

2.1.7 Status Register

The register of 16 bit long and contains the following status information: the group of index registers selected, short address extension bits, program flags, interrupt enable, carry and overflow bits, the halfword arithmetic indicator, etc.

The Status Register bits are defined as follows:

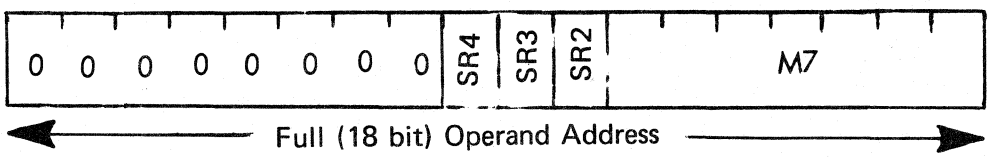
SR0	Index Register Selection
SR1	Index Register Selection
SR2	Short Address Extension
SR3	Short Address Extension
SR4	Short Address Extension
SR5	Halfword Arithmetic Mode
SR6	Disable XR7 during RTM
SR7	Unassigned
SR8	Program Flag
SR9	Program Flag
SR10	Program Flag
SR11	Program Flag
SR12	Carry or Borrow from B to A Register
SR13	Fixed Point Double Precision (one implies double precision)
SR14	Memory Interrupt (one allows interrupt)
SR15	Program Interrupts (one allows interrupt)

2.1.7.1 Index Register Selection

Interrupt housekeeping is facilitated by the use of bits SR0 and SR1 of the status register to designate one of four groups of index registers. If the standard (reentrant) subroutine calling sequence is employed, the index registers are used to control the temporary memory area. A different set of index registers is used for each of four interrupt levels.

2.1.7.2 Short Address Extension

Bits SR2, SR3, and SR4 may be used to extend the operand addressing range of short memory reference instructions that are not indexed, and short Index Register Control instructions. These three bits are effectively appended to the most significant bits of the 7 bit short address field (M7) as shown below:



This permits an addressing capability of 10 bits for this class of instructions.

Bits SR2, SR3, and SR4 are never used to modify the address field of a long instruction or of a short indexed instruction.

This provides a greater addressing range for this class of instructions.

Bits SR2, SR3, and SR4 are never used to modify the address field of a long instruction or of a short indexed memory reference instruction.

When the initial address of an SKC 2000 physical memory is offset from the origin, short unindexed instructions have a constant offset automatically added to the address field to form an operand address in the front of the physical memory. This factor must be considered in determining the effect of the address extension bits (SR2, 3, 4). Consider the baseline machine with a core memory beginning at address 16,384 and a fast scratchpad memory with addresses ranging from 15,872 to 16,383. The combined effect of the constant offset and the SR2, SR3, and SR4 bits on the addressing range of short unindexed instructions are shown in Table 2-1.

TABLE 2-1. MEMORY FIELDS DEFINED BY SR2, SR3 AND SR4

MEMORY TYPE	STATUS REGISTER BITS			ADDRESSING RANGE	
	SR4	SR3	SR2	SR5 = 0 FULLWORD MODE	SR5 = 1 HALFWORD MODE
FAST SCRATCHPAD	0	0	0	Growth	15,872 to 15,999
	0	0	1		16,000 to 16,127
	0	1	0	15,872 to 16,126	16,128 to 16,255
	0	1	1	16,128 to 16,382	16,256 to 16,383
CORE	1	0	0	16,384 to 16,638	16,384 to 16,511
	1	0	1	16,640 to 16,894	16,512 to 16,639
	1	1	0	16,896 to 17,150	16,640 to 16,767
	1	1	1	17,152 to 17,406	16,768 to 16,895

2.1.7.3 Halfword Arithmetic Mode

Bit SR5 is used to control whether short instructions are executed in the halfword or fullword mode. If SR5 = 1, then for all short arithmetic and logical instructions, except load and store index registers, the operand has a 16 bit data format. If SR5 = 0, all short instructions operate on fullword (32 bit) data. The only exception is the store halfword (STH) instruction which always operates on a short dataword (16 bits) independent of the setting of SR5.

With long instructions and the H bit (bit 15) of the instruction set to 1 the operand has a 16 bit data format independent of SR5. If the addressing mode is immediate, the 16 bit data operand is left justified with the H bit set. If this bit is not set, the immediate operand is right justified. If the addressing mode is indirect, then the H bit should be 0 until the last level is reached. The H bit (bit 15) of the instruction should not be used with the floating point instructions.

In general, in half word mode, the half words are left adjusted when transmitted to the CPU from memory. When storing words in memory in half word mode, the half word is taken from the left half of the CPU register and placed in either the left or right part of a (32 bit) memory word as directed by the effective operand address.

2.1.7.4 Disable XR7

Bit SR6 is used to disable the use of XR7 as an address modifier for instructions executed with the return to memory option. If SR6=0, the contents of XR7 are added to the address field when designated in the X1 field. If SR6=1, the contents of XR7 are not added to the address field.

2.1.7.5 Program Flags

Bits SR8, SR9, SR10, SR11 of the Status Register are settable and resettable under program control. These may also be tested by an instruction provided for the purpose.

2.1.7.6 Carry and Double Precision Indicator

Bits SR12 and SR13 permit the easy implementation of fixed point double precision (64 bit) Arithmetic. SR13 is used to indicate the occurrence of a B Register Add or Subtract (ADL, or SBL). If SR13 is on, then SR12 indicates whether or not a Carry or a borrow should be added to the result of a following A Register Add or Subtract (ADU or SBU). SR12 in this case is actually an overflow/underflow indicator.

2.1.7.7 Memory Interrupt Inhibit Bit

Bit SR14 provides the ability to enable (SR14=1) or disable (SR14=0) all memory interrupts associated with DMA transfers. If an interrupt signal occurs while all interrupts are disabled (SR14=0), its effect is delayed until interrupts are enabled (SR14=1). The interrupt is not lost.

2.1.7.8 Program Interrupt Inhibit Bit

Bit SR15 provides the ability to enable (SR15=1) or disable (SR15=0) all program interrupts. If an interrupt signal occurs while all interrupts are disabled (SR15=0), its effects are delayed until interrupts are enabled (SR15=1). The interrupt is not lost.

2.1.8 Program Interrupt Mask Register

This register is 16 bits long and may be used to mask (or inhibit) individual interrupts. Unlike the bit in the Status Register which disables all interrupts, the Interrupt Mask Register (MR) causes the interrupt to be lost if it occurs during the period it is inhibited.

2.1.9 Concatenation of A and B Registers

For certain instructions a double length (64 bit) data word is involved. These instructions include: MUL, DVD, MLF, DVF, AFD, SFD, SRD, SLD, SRCD, SLCD, and SRAD. For these operations, the A Register and the B Register are concatenated (joined together) to form a single 64 bit register (denoted A,B). Bit 0 of the A Register is the most significant bit and bit 31 of the B Register is the least significant bit of the A,B Register.

2.2 ADDRESSING

The SKC-2000 Memory is organized in 32 bit words. (See Table 2-2). Each word consists of two 16 bit halfwords which are directly addressable. A halfword may contain a short instruction (16 bits) or a short data word (16 bits). Consequently, each long instruction (32 bits) or each long data word (32 bits) must be designated by an even address. Double length (64 bits) data words may be designated at any even address, the designated 32 bit word and the following 32 bit word is joined to form the 64 bit word. In the case of a mixture of long and short instruction, the Assembler will automatically insert a NOP, where appropriate, to force each long instruction to be at an even address.

The SKC-2000 Computer contains 18 bit address registers yielding an addressing range of 0 to 262, 143 which is sufficient for a 131K machine.

2.2.1 Effective Address

The effective address is defined to be the final target address of an instruction after all levels of indirectness and indexing have been accomplished. When the mode of the instruction is normal (not immediate or indirect) the effective address is the sum of the address field and the contents of any specified index registers.

2.2.2 Indirect Addressing

Indirect addressing is a feature of long instructions. Up to sixteen levels of indirectness may be utilized. Specification of indirect addressing is achieved by setting bit 13 of the instruction. With this bit set, the instruction is executed in the following way:

An **effective** address is computed in the normal manner, by adding the contents of any specified index register to the address field. This is known as an indirect effective address. The computer then examines the contents of the 32 bit fullword specified by this indirect effective address and uses the index register and address fields (based on the long instruction format) to compute another effective address. If the indirect bit is set in this word, the process is repeated. Otherwise, the final effective address of the instruction has been reached. A maximum of 16 levels are permitted. If the immediate bit is set, the new effective address is treated as an immediate operand. (See Section 2.2.3). Setting of both the immediate and indirect bits to one is undefined and should not be specified.

TABLE 2-2. SKC-2000 MEMORY

WORDS	ADDRESSES	CONTENTS	COMMENTS
0 7935	0 15871	RAM/ROM MODULES (FUTURE)	SOLID STATE MEMORY
7936 8191	15872 16383	FAST SCRATCHPAD	
8192 12287	16384 24575	CORE SCRATCHPAD	READ/WRITE
12288	24576		READ ONLY BOUNDARY ADJUSTABLE
16336 16351	32672 32703	INTERRUPT TRAPS	PROTECTED MEMORY
		CORE SCRATCHPAD	CORE MEMORY
16368 16383	32736 32767	INTERRUPT RETURNS	
16384 131071	32768 262143	CORE MODULES (FUTURE)	

2.2.3 Immediate Operand

Setting bit 14 of a long instruction to a one specifies an immediate operand. In this case, the effective address (as computed from the sum of the operand field and the contents of the specified index registers) is taken to be the actual operand. Indirect addressing is not permitted when the immediate operand bit is set.

The immediate operand is left adjusted if bit 15 of the instruction is set to 1 and is right adjusted if bit 15 is set to 0. An exception to this is the Load Status and Load Interrupt instructions. In these cases the operand is always right adjusted for the Load Status instruction and is always left adjusted for the Load Interrupt instruction.

2.2.4 Address Modification With Index Registers

Address modification is accomplished by specifying one or two index registers. A single index modification is specified by the field designated X1 (bits 6-8) in either a long or short instruction. An index modification can also be specified by the field designated X2 (bits 9-12) in a long instruction. If both fields specify indexing, both designated registers are added to the address field. If a second level register (XR8 - XR15) is specified, or if two registers are assigned, the instruction must be long. In all cases, the effective address is computed by adding the contents of the referenced register(s) to the address field. A group of instructions are used to alter or test the contents of an index register. The number used to test or alter an index register may be contained in the operand field of the instruction (using the immediate option) or in memory.

Note that the 3 bit X1 field can designate 7 registers (XR1 to XR7) while the four bit X2 can designate 15 registers (XR1 to XR15). When X1 is used to specify XR7, it also designates the return to memory option.

2.2.5 Address Offset

In an SKC 2000 computer whose physical memory begins at address zero, short unindexed instructions may be used to reference the lower memory address. In a typical machine, however, the use of both core and fast scratchpad memory will cause the initial physical address to be offset from zero. In this case, a constant offset is automatically added to the address field of each short unindexed instruction to assure that it may still designate the data in the lowest address of the physical memory. See paragraph 2.1.7.2 for precise addressing information.

2.2.6 Halfword Addressing

Halfword data (16 bits) can be fetched by both long and short instructions. In the long instruction, setting the H bit (15) = 1 results in a 16 bit operand left adjusted in a 32 bit field with the 16 rightmost bits reset to 0. The result is the same for an immediate operand with H=1. When indirect addressing is employed, the H bit should not be set until the last indirect level. In the short instruction, setting Status Register bit 5 = 1 results in a left adjusted 16 bit operand.

2.3 ARITHMETIC

Arithmetic operations (add, subtract, multiply, and divide) and logical operations are performed in the A and B registers. The B Register provides an extension of the A Register for double precision arithmetic. The result of the Arithmetic operations appears in the A Register or in the combined AB Registers unless the Return to Memory feature is employed. In this case, the final contents of the AB Registers depends on the specific instruction and the result appears in memory.

2.4 INTERRUPTS

2.4.1 Program (CPU) Interrupts

Automatic program interrupts are used to signal, to the program, conditions requiring attention without requiring special test instructions. With interrupts, system status is constantly monitored and, when particular special conditions are detected, normal processing is interrupted and the program is transferred to an interrupt routine.

To identify the cause of the interrupt and to allow for a return to normal processing, the program is transferred to a fixed location and the address of the interrupted program (return address) is stored in another fixed location when an interrupt is initiated. The fixed locations depend upon the particular interrupt. (See Table 2-3). This fixed location should contain a JGU instruction, which will transfer control to a routine to process the interrupt. Exit from the interrupt routine is performed by an RTA instruction. See Section 2.1.7.1 on the use of several groups of index registers for interrupts.

2.4.2 Assigned Locations

Storage locations assigned for interrupts are, in order of increasing priority, as follows:

TABLE 2-3

Interrupt Priority Level	Transfer Location Hexadecimal	Decimal	Return Address Hexadecimal	Decimal
0	7FA0	32672	7FE0	32736
1	7FA2	32674	7FE2	32738
2	7FA4	32676	7FE4	32740
3	7FA6	32678	7FE6	32742
4	7FA8	32680	7FE8	32744
5	7FAA	32682	7FEA	32746
6	7FAC	32684	7FEC	32748
7	7FAE	32686	7FEE	32750
8	7FB0	32688	7FF0	32752
9	7FB2	32690	7FF2	32754
10	7FB4	32692	7FF4	32756
11	7FB6	32694	7FF6	32758
12	7FB8	32696	7FF8	32760
13	7FBA	32698	7FFA	32762
14	7FBC	32700	7FFC	32764
15	7FBE	32702	7FFE	32766

Interrupt Disable

One bit of the Status Register is used to temporarily inhibit all program interrupts. This bit may be set or reset under program control. Interrupts received when the disable is effective, will be delayed until the inhibit is turned off.

Interrupt Masking

The Program Interrupt Mask Register, which is under program control, is used to mask individual interrupts. Interrupts received when masked out are ignored.

Masking out interrupts may be used to clear out interrupts which have been received but not yet acted upon, if they are of no interest.

HALT

If the HLT instruction is interrupted, then the interrupt takes place (unless masked or disabled). The address of the HLT instruction plus one is stored in the interrupt store location. On return, the next instruction (not the HALT) is executed.

INTERRUPT ASSIGNMENT

Interrupt Level	Description
15	Power Fail
14	Control Panel (if used)
13	Built-In-Test
12-0	Unassigned

Interrupt Priority

Interrupt priority is shown in the previous table in order of increasing priority. Interrupts (except those masked out) are responded to in order of priority as follows:

a) At the conclusion of each instruction the highest priority interrupt will be serviced.

b) When an interrupt is being serviced, no interrupt of lower or equal priority can interrupt it until the higher priority interrupt is completed. Completion occurs when an RTA instruction is executed with an effective address equal to the interrupt store location for that interrupt.

c) An interrupt of higher priority can interrupt one of lower priority.

Control Panel Interrupt

When the Manual Control Panel (MCP) is connected, the CPU is able to operate with the keyboard/typewriter if this function is enabled by the MCP operator. The MCP interrupt has the second highest priority. An interrupt takes place whenever the operator types a character and also after completion of a character being typed by the CPU.

2.4.3 Memory Interrupts

Both the CPU and the I/O unit can access the memory over the same data bus. Consequently, if the CPU tries to access memory while an I/O memory transfer is taking place, its operation will be suspended until the I/O memory transfer is complete. This process is referred to as a memory interrupt.

Often the I/O memory transfer will take place during a period when the CPU is not accessing memory, for example during a long instruction (MUL, DVD, ADF, ADD, etc.). If not, the execution time of the interrupt instruction will be extended by the length of time the operation of the CPU was suspended.

2.5 INPUT-OUTPUT OPERATIONS

2.5.1 Multiplexed Input/Output Channel

The multiplexed input/output channel is used to transmit data between the CPU or memory and an external device, under program control. A common 32 bit data bus is used for all devices.

A 6-bit device code and a Command Bit are used to address each device. The device must recognize its own device code and transmit or accept data via the data bus.

An Acknowledge signal is used for those devices which require additional time to send or receive data. When an acknowledge is required, the CPU will wait until it receives an acknowledge from the device before completing execution of the Input/Output instruction. However, the CPU will not wait more than 4 μ s for the acknowledge; if no acknowledge is received within 4 μs, after the device code is issued, the CPU will complete execution in the same manner as if an acknowledge had been received.

The Input/Output instruction causing data transfer specifies the device code, command bit, acknowledge, and whether an input or output is required. The external device must take appropriate action, depending on its device code and command bit assignment.

Data may be transferred to or from the memory or the A Register in the CPU. There are thus four modes of data transfer, which are described in the following sections.

Input To A Register

When the device code is received by the device, the device must place the data on the data bus. The CPU then accepts the data and places it in the A Register. The CPU reads the data on the bus approximately 250ns after an acknowledge is received, or, if no acknowledge is required, 1000ns after it transmits the device code.

Output From A Register

The CPU places the contents of the A Register on the data bus and issues the device code. On receiving the device code, the external device must read the data. The CPU will hold the data on the bus until an acknowledge is received or, if no acknowledge is required by the instruction, for 1000ns.

Output From Memory

The CPU issues the device code and generates an initiate pulse to the memory. When the data appears on the bus, a signal, DP, is sent to the device. The device reads the data when it receives DP, which occurs 0.75 μs after the device code is issued. The data remains on the bus until the acknowledge is received or, if no acknowledge is required, for at least 250ns after DP occurs.

Input To Memory

The CPU issues the device code. Upon receipt of the device code, the external device must place the data to be stored on the data bus within 250ns. The CPU then stores the data. The use of the acknowledge delays the execution of the instruction, but does not affect the timing of data storage; therefore, the acknowledge should not be used in this mode.

It should be noted that the programmer is not free to select which of the four modes of transfer is to be used; the mode of data transfer depends only on the design of the external device.

2.5.2 MCP Input/Output

When the MCP is connected to the computer, the operator may enable the CPU to operate the keyboard/typewriter under program control. Two device codes are used: one to operate the typewriter (output), and one to read from the keyboard (input). Data is transferred between the MCP and the A Register, and the MCP will send an acknowledge back to the CPU when a transfer takes place.

Device code 16 (10) is used for typing. To type a character, the CPU places the 6 bit code for the character to be typed in the least significant bits of the A Register and executes an output instruction with a device code of 16 (10) and with an acknowledge required. When the MCP is ready for another character, the control panel interrupt is initiated. The CPU can then type another character.

When the keyboard input is used, the control panel interrupt occurs whenever a character is typed by the operator. The CPU then issues an input command with device code 17 (10), which transfers the 6-bit code of the character to the 6 least significant bits of the A Register.

Whenever an input/output instruction with device code of 16 or 17 is executed, the control panel interrupt is turned off until the next character is ready for input or output.

The keyboard/typewriter codes for each character are shown in the following table.

THE SINGER COMPANY • KEARFOTT DIVISION

7 BIT ASCII FORMAT

PRINT	HEX	DEC
A	4 1	6 5
B	4 2	
C	4 3	
D	4 4	
E	4 5	
F	4 6	7 0
G	4 7	7 1
H	4 8	7 2
I	4 9	7 3
J	4 A	
K	4 B	7 5
L	4 C	
M	4 D	
N	4 E	
O	4 F	
P	5 0	8 0
Q	5 1	
R	5 2	8 2
S	5 3	8 3
T	5 4	
U	5 5	
V	5 6	
W	5 7	8 7
X	5 8	8 8
Y	5 9	
Z	5 A	
0	3 0	4 8
1	3 1	
2	3 2	
3	3 3	
4	3 4	
5	3 5	
6	3 6	
7	3 7	
8	3 8	
9	3 9	5 7

PRINT	HEX	DEC
!	2 1	
"	2 2	
#	2 3	
$	2 4	
%	2 5	
&	2 6	
'	2 7	
(	2 8	
)	2 9	
*	2 A	
+	2 B	4 3
,	2 C	4 4
–	2 D	4 5
.	2 E	4 6
/	2 F	
:	3 A	
;	3 B	
<	3 C	
=	3 D	
>	3 E	
?	3 F	
@	4 0	
⊏	5 B	
\	5 C	
⊐	5 D	
↑	5 E	
←	5 F	
EOT	0 4	
WRU	0 5	
RU	0 6	
L/F	0 A	1 0
C/R	0 D	1 3
SP	2 0	3 2
ALT	7 D	
RUB	7 F	
TAB	0 9	
BELL	0 7	

THIS PAGE INTENTIONALLY LEFT BLANK

3. INSTRUCTION DESCRIPTION

The description of instructions is broken down into six groups, one major section to a group. The beginning of each section describes characteristics which are for the most part, common to the entire group. Exceptions are noted in the description of each instruction.

A notation used by all the following descriptions is:

X1	Refers to a 3 bit field which specifies one of seven first level index registers. Zero signifies no indexing.
X2	Refers to a 4 bit field which specifies any one of the fifteen index registers. Zero signifies no indexing.
I	Refers to a 1 bit field which specifies indirect addressing if set to a one.
M	Refers to a 1 bit field which specifies immediate operand if set to a one. Note that I and M may not both be one.
H	Refers to a 1 bit field which specifies an arithmetic operation on halfword data.
M7	Refers to a 7 bit field which specifies a displacement address of a short instruction.
M16	Refers to a 16 bit field which specifies a memory address of a data reference long instruction.
M18	Refers to an 18 bit field which specifies the destination address of long jump instruction.
(X1)	Contents of Index Register X1
(X2)	Contents of Index Register X2

SKC 2000 INSTRUCTION LIST

OP-CODE	LENGTH	MNEMONIC	OPERATION DESCRIPTION	OPERATION SUMMARY
00000 0001.....	S	EMI	Enable Memory Interrupts	SR14 Is Set To 1
00000 0010.....	S	DPI	Disable Program Interrupts	SR15 Is Set To 0
00000 0011.....	S	DMI	Disable Memory Interrupts	SR14 Is Set To 0
00000 0100.....	S	EPI	Enable Program Interrupts	SR15 Is Set To 1
00000 0101.....	S	HLT	Halt	Halts If Test Signal Is Present
00000 0110.....	S	SET	Set Selected Program Flags	Sets Indicated Flags To 1
00000 0111.....	S	RST	Reset Selected Program Flags	Resets Indicated Flags To 0
00000 1000.....	S	CFX	Convert Floating To Fixed	(A,B) → A,B
00000 1001.....	S	CXF	Convert Fixed To Floating	(A,B) → A,B
00000 1010.....	S	EAB	Exchange A and B	(A)→B, (B)→A
00000 1011.....	S	SHM	Set Halfword Mode	SR5 Bit Set To 1
00000 1100.....	S	RHM	Reset Halfword Mode	SR5 Bit Reset To 0
00000 11010.....	S	LXA	Load Index Register From A	(A) XR (18 Low Order Bits)
00000 1110.....	S	NOP	No Operation	No Operation
00001 0...00.....	S	SLLD	Shift A, B Left Logically	Shift By EA
00001 0...01.....	S	SLCD	Shift A, B Left Circularly	Shift By EA
00001 0...10.....	S	SLL	Shift A Left Logically	Shift By EA
00001 0...11.....	S	SRLD	Shift A, B Right Logically	Shift By EA
00001 1...00.....	S	SRAD	Shift A, B Right Algebraically	Shift By EA
00001 1...01.....	S	SRCD	Shift A, B Right Circularly	Shift By EA
00001 1...10.....	S	SRA	Shift A Right Algebraically	Shift By EA
00001 1...11.....	S	SRC	Shift A Right Circularly	Shift By EA
00010 0.....	S,L	LDA	Load A Register	(EA) → A
00011 0.....	S,L	STX	Store Index Register	(XR) → EA
00100 0	S,L	ICN	Test XR and Skip On Not Equal	Skip if (XR) ≠ (EA)
00100 1.....1	L	ICL	Test XR and Skip on Less Than	Skip if (XR)< (EA)
00101 0.....	S	MFM	Move Black From Fast to Main	
00101 1.....	S	MMF	Move Block From Main to Fast	
00110	S,L	LAE	Load A With EA	EA → A
00111	S,L	STA	Store A Register	(A) → EA

Abbreviations:

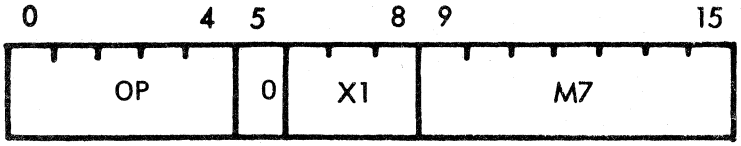
( )	Contents of	CARRY	Carry Status Bit
A	A Register	MR	Interrupt Mask Register
B	B Register	SR	Status Register
EA	Effective Address	PC	Program Counter
XR	An Index Register	→	Goes Into
		Δ	Floating Point

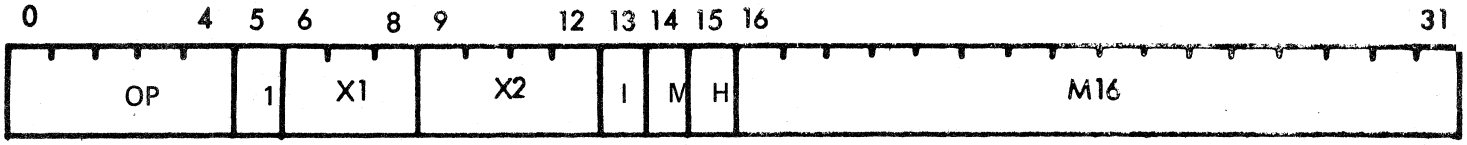
SKC 2000 INSTRUCTION LIST (Continued)

OP-CODE	LENGTH	MNEMONIC	OPERATION DESCRIPTION	OPERATION SUMMARY
01000	S,L	STH	Store Left Half of A Register	$(A)_{0-15} \rightarrow EA$
01001 0.....0.	S	DOA	Data Output From A Register	
01001 0.....1.	S	DIA	Data Input To A Register	
01001 1.....0.	L	DOM	Data Output From Memory	
01001 1.....1.	L	DIM	Data Input To Memory	
01010	S	LDB	Load B Register	(EA)→B
01011	S,L	LDX	Load XR Register	(EA)→XR
01100 000.....	S	JU,JRU	Jump Unconditional	Jump To EA
01100 001.....	S	JN,JRN	Jump If A ≠ 0	Jump To EA if (A) ≠ 0
01100 010.....	S	JG,JRG	Jump If A ≥ 0	Jump To EA If (A) ≥ 0
01100 011.....	S	JL,JRL	Jump If A < 0	Jump To EA If (A) < 0
01100 1...00001..	L	JS	Jump To Subroutine	(PC)+2→EA Indirectly, Jump To EA+2
01100 1....0010..	L	JGF	Jump On Program Flag	Jump To EA If Any Flag Tested Is On
01100 1....0100..	L	JGS	Jump On Status Bit	Jump To EA If Status Bit On
01100 1...00110..	L	JGW	Jump On Switch	Jump To EA If Switch On
01100 10000110...	L	JU,JGU	Jump Unconditional	Jump To EA
01100 10100100...	L	JN,JAN	Jump If A ≠ 0	Jump To EA If (A) ≠ 0
01100 110001000..	L	JG,JAG	Jump If A ≥ 0	Jump To EA If (A) ≥ 0
01100 111001000..	L	JL,JAL	Jump If A ≤ 0	Jump To EA If (A) ≤ 0
01101 1.....0	L	IMP	Modify Index Register Positive	(XR)+(EA)→XR
01101 1	S,L	IMN	Modify Index Register Negative	(XR)−(EA)→XR
01110	S,L	RTA	Return Address	Jump Indirect Via EA
01111	S,L	STB	Store B Register	(B)→EA
10000	S,L	AND	Logical And	(A) AND (EA)→A
10001	S,L	SAM	Skip On A Register Masked	Skip Unless (A) and (EA) = 0
10010	S,L	MLF	Multiply - Floating Point	$(A)^{\Delta} *$ (EA)→A, B
10011 0	S,L	AFD	Add Floating Double Precision	$(A,B)^{\Delta} +$ (EA,EA+2)→A,B
10011 1.....1	L	LDS	Load Status Register	(EA)→SR
10100	S,L	ADU	Add Upper - Fixed Point	(A) + (EA) + Carry→A
10101	S,L	ADL	Add Lower - Fixed Point	(B) + (EA)→A
10110 0	S,L	DVF	Divide - Floating Point	$(A,B)^{\Delta} /$ (EA)→A, Remainder→B
10111 0	S,L	ADF	Add - Floating Point	$(A)^{\Delta} +$ (EA)→A
10111 1.....1	L	STI	Store Interrupt Mask Register	(MR)→EA
11000	S,L	LOR	Logical OR	(A) OR (EA)→A
11001	S,L	EXO	Exclusive OR	(A) XOR (EA)→A
11010	S,L	MUL	Multiply - Fixed Point	(A)*(EA)→A,B
11011	S,L	SFD	Subtract - Floating Double Precision	$(A,B)^{\Delta} -$ (EA,EA+2)→A,B
11011 1.....1	L	STS	Store Status Register	(SR)→(EA)
11100	S,L	SBU	Subtract Upper - Fixed Point	(A) - (EA) - Carry→A
11101	S,L	SBL	Subtract Lower - Fixed Point	(B) - (EA)→B
11110	S,L	DVD	Divide - Fixed Point	(A,B)/(EA)→A, Remainder→B
11111 0	S,L	SBF	Subtract - Floating Point	$(A)^{\Delta}$ - (EA)→A
11111 1.....1	L	LDI	Load Interrupt Mask Register	(EA)→MR

3.1 ARITHMETIC INSTRUCTIONS

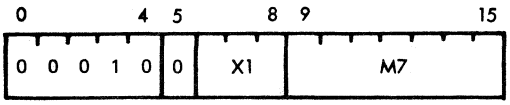
These instructions have two possible formats: Short (16 bits) or long (32bits). They all reference memory and may be indexed to the degrees specified. All of the floating point operations assume normalized operands except ADF, SBF, AFD, SFD. The assembler automatically chooses the short instruction format, if possible. This decision is based on information supplied to the Assembler by the programmer via several special pseudo- operations. The basic instruction format for the arithmetic instruction is given below:

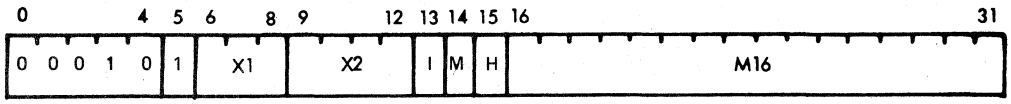




Bit 5	Short/Long Designator
X1	Index Designator (XR1-XR7)
X2	Index Designator (XR1-XR15)
1	Indirect Addressing
M	Immediate Operand Designator
H	Halfword Arithmetic Designator
M7,M16	Address Fields

LDA: Load the A Register





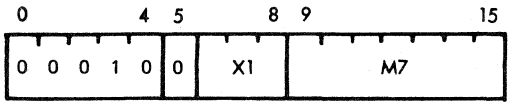
This instruction operates on fullwords; in the long format H = 0 and when a short format LDA is executed, the status register must be set to fullword mode (SR5 = 0). The fullword in memory, designated by the effective address, is placed in the A Register (Bits 0-31). The contents of the effective address are unchanged.

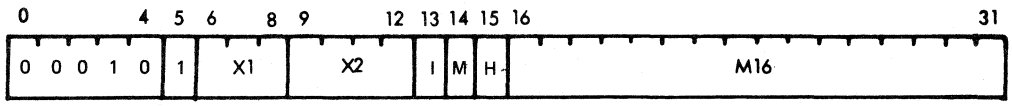
If an immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are loaded right adjusted into the A Register (Bits 16-31). The sign bit is extended to fill the most significant half.

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

This instruction does not have return-to-memory capability.

LDAH: Load the A Register with Halfword





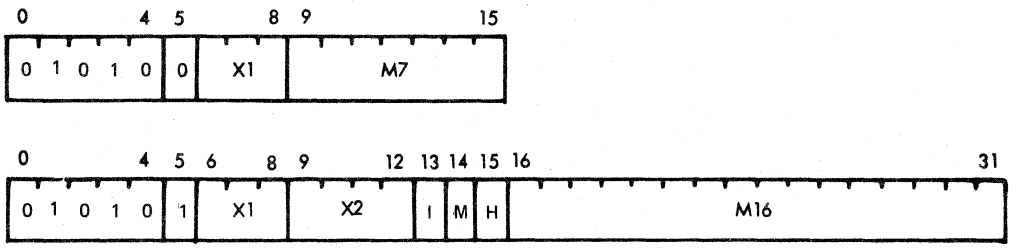
This instruction operates on halfwords; in the long format H = 1 and when a short format LDAH is executed, the status register must be set to halfword mode (SR5=1). The halfword in memory, designated by the effective address, is placed in the A Register left adjusted (Bits 0-15). The least significant half of the A Register (Bits 16-31) is set to zero. The contents of the effective address are unchanged.

If an immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are loaded left adjusted into the A Register (Bits 16-31). The least significant half of the A Register (Bits 16-31) is set to zero.

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

This instruction does not have return-to-memory capability.

LDB: Load the B Register



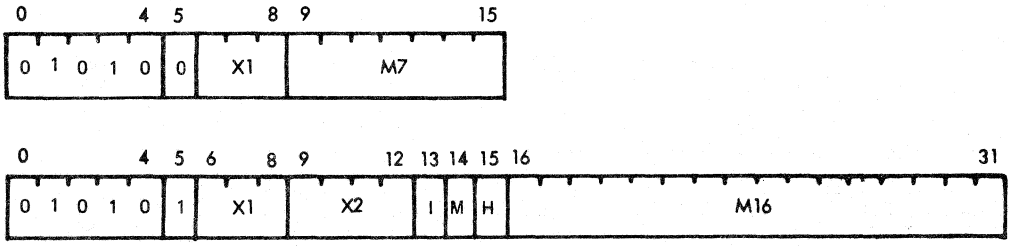
This instruction operates on fullwords; in the long format H = 0 and when a short format LDB is executed, the status register must be set to fullword mode (SR5 = 0). The fullword in memory, designated by the effective address, is placed in the B Register (Bits 0-31). The contents of the effective address are unchanged.

If an immediate operand is designated, (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are loaded right adjusted into the B Register (Bits 16-31). The sign bit is extended to fill the most significant half.

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

This instruction does not have return-to-memory capability.

LDBH: Load the B Register with Halfword



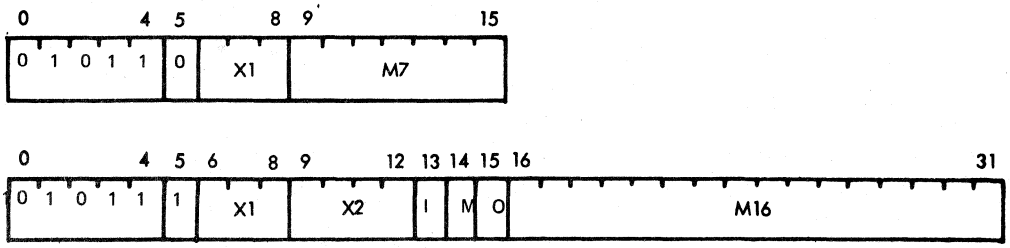
This instruction operates on halfwords; in the long format H = 1 and when a short format LDBH is executed, the status register must be set to halfword mode (SR5=1). The halfword in memory, designated by the effective address, is placed in the B Register left adjusted (Bits 0-15). The least significant half of the B Register (Bits 16-31) is set to zero. The contents of the effective address are unchanged.

If an immediate operand is designated, (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are loaded left adjusted into the B Register (Bits 16-31). The least significant half of the B Register (Bits 16-31) is set to zero.

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

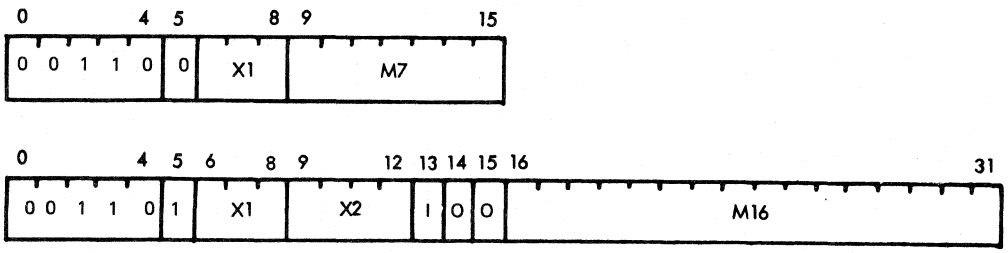
This instruction does not have return-to-memory capability.

LDX: Load Index Register



The least significant 18 bits contained in the location specified by the effective address are placed in the designated Index Register. In the short format, the X1 field designates an Index Register (XR0-XR7) as the target of the instruction and no indexing of the effective address is permitted. In the long format, the X2 field designates an Index Register (XR0-XR15) as the target of the instruction. The X1 field is used to designate modification of the effective address. If an Immediate operand is designated (M=1), the effective address is loaded in the designated under register. The halfword option (H=1) cannot be used. This instruction does not have return-to-memory capability.

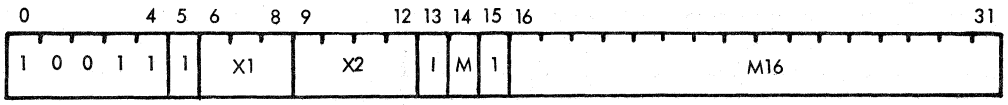
LAE: Load The A Register With The Effective Address



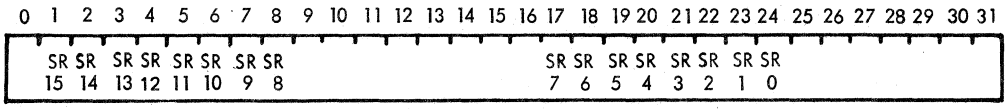
The effective address (E) is placed in the A Register. No other registers are affected. The immediate and halfword options are ignored. This instruction is most commonly used to transmit argument addresses to subroutines using the indirect option to reference the argument list in the calling program.

NOTE: If an immediate option is encountered during indirect references, the current effective address (in this case an operand not an operand address), is loaded in the A Register. Since an operand address is always desired, this situation should be avoided. This instruction does not have return-to-memory capability.

LDS: Load Status Register



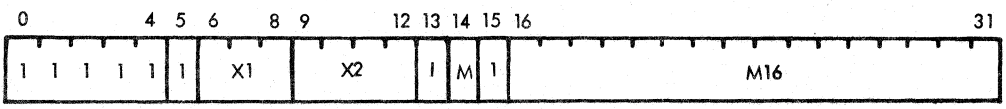
The contents of the effective address are placed in the Status Register according to the following format:



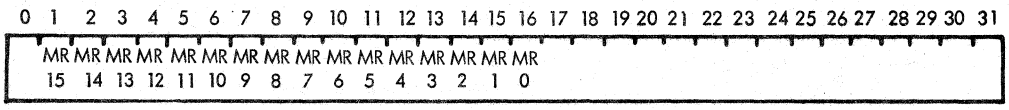
For example, if bit 24 of the contents of the effective address is a 1, then SR9 of the Status Register is set to 1. The contents of the effective address are unchanged. In the immediate mode, status register bits SR8 - SR15 are not affected. No other registers are affected. This instruction does not have return-to-memory or halfword capability. Bit 15 of the instruction must be a 1, if not this instruction becomes an AFD instruction.

NOTE: There are several non-memory reference instructions which can be used to directly set certain bits in the Status Register, viz. EMI, DPI, DMI, EPI, SET, RST, SHM, RHM.

LDI: LOAD INTERRUPT MASK REGISTER

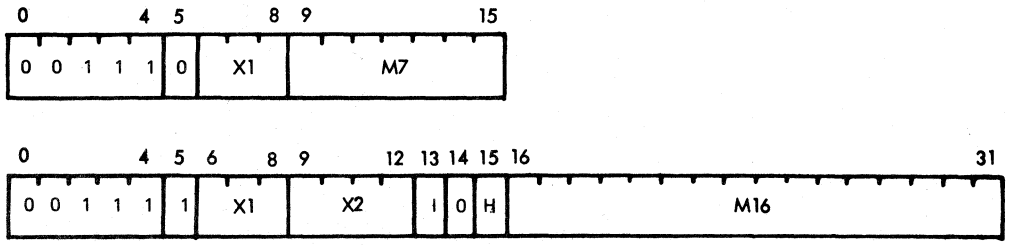


The contents of the effective address are placed in the Interrupt Mask Register according to the following format:



No other registers are affected. If bit 15 of the instruction is set to 0, this instruction becomes an SBF instruction. If the immediate operand is designated (M=1), the least significant 16 bits of the effective address are treated as the most significant half of the operand. That is the LSB of the effective address goes to MR1, the next bit to MR2, etc. Bit MR0 is set to zero. The halfword mode is not available with LDI and it does not have return-to-memory capability.

STA: Store the A Register

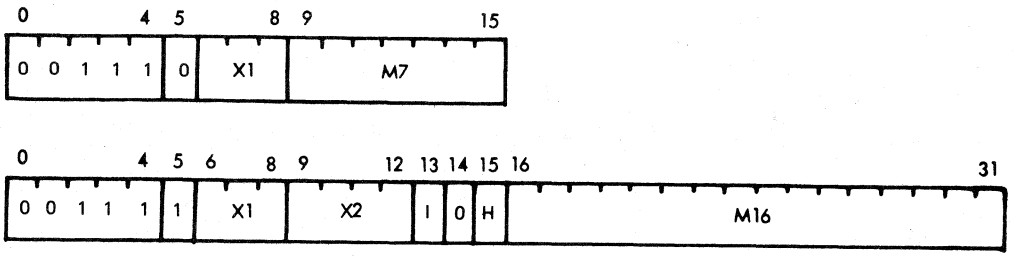


This instruction operates on fullwords; in the long format H = 0 and when a short format STA is executed, the status register must be set to fullword mode (SR5 = 0). The fullword in the A Register is stored in the memory cells designated by the effective address. The contents of the A Register are unchanged.

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

The immediate option should not be used and the instruction does not have return-to-memory capability.

STAH: Store the A Register, Halfword

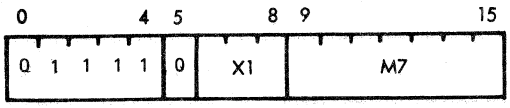


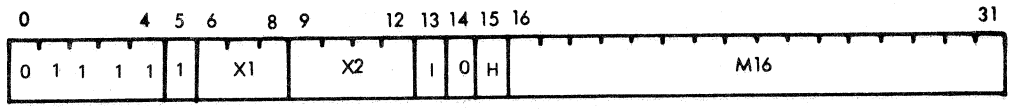
This instruction operates on halfwords; in the long format H = 1 and when a short format STAH is executed, the status register must be set to halfword mode (SR5 = 1). The most-significant half of the A Register (Bits 0-15) is stored in the memory cell designated by the effective address. The contents of the A Register are unchanged.

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

The immediate option should not be used and the instruction does not have return-to-memory capability.

STB: Store the B Register



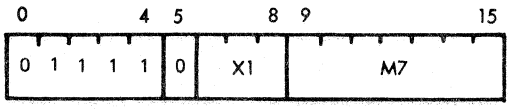


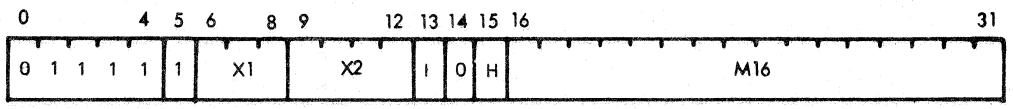
This instruction operates on fullwords; in the long format H = 0 and when a short format STB is executed, the status register must be set to fullword mode (SR5 = 0). The fullword in the B Register is stored in the memory cells designated by the effective address. The contents of the B Register are unchanged.

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

The immediate option should not be used and the instruction does not have return-to-memory capability.

STBH: Store the B Register, Halfword



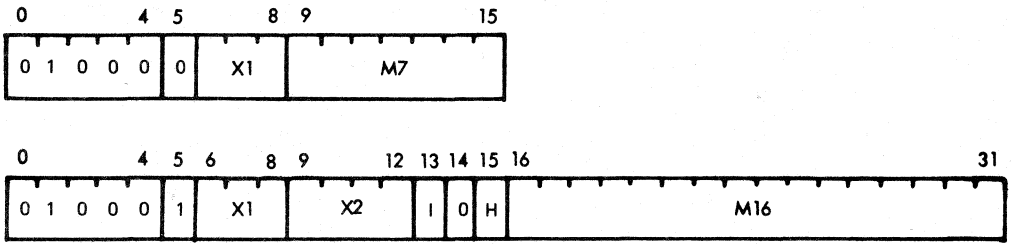


This instruction operates on halfwords; in the long format H = 1 and when a short format STBH is executed, the status register must be set to halfword mode (SR5 = 1). The most significant half of the B Register (Bits 0-15) is stored in the memory cell designated by the effective address. The contents of the B Register are unchanged.

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

The immediate option should not be used and the instruction does not have return-to-memory capability.

STH: Store A Register Halfword



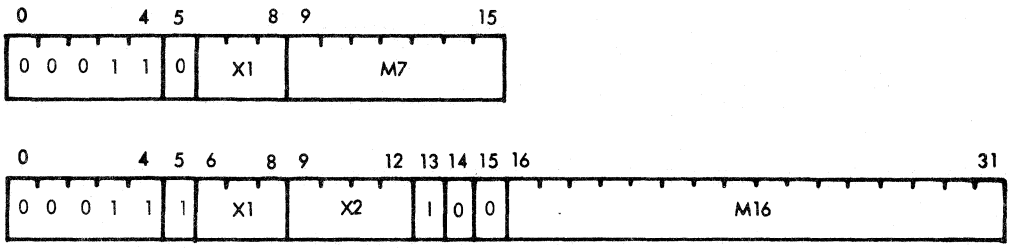
This instruction operates on halfwords independent of the setting of the H bit in the long format or the halfword mode bit in the status register (Bit SR5) for the short format. The most significant half of the A Register (Bits 0-15) is stored in the memory cell designated by the effective address. The contents of the B Register are unchanged.

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

The immediate option should not be used and the instruction does not have return-to-memory capability.

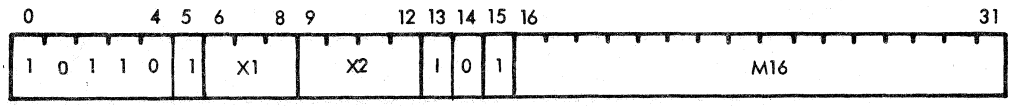
This instruction is the same as an STAH except that the STH does not examine the H or SR5 settings. It automatically executes in halfword mode even if H = 0 or SR5 = 0 (which would normally indicate fullword mode).

STX: Store Index Register

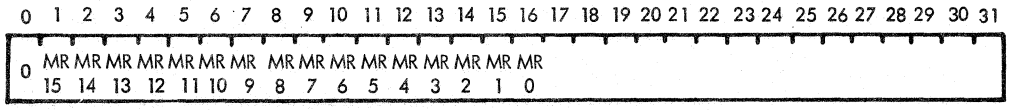


The contents of the designated Index Register are stored in the 18 least significant bit positions specified by the effective address. In the short format, the X1 field designates an Index Register (XR0-XR7) as the target of the instruction and no indexing of the effective address is permitted. In the long format, the X2 field designates an Index Register (XR0-XR15) as the target of the instruction. The X1 field is used to designate modification of the address. The immediate and halfword options should not be used. The instruction has no return-to-memory capability.

STI: Store Interrupt Mask Register

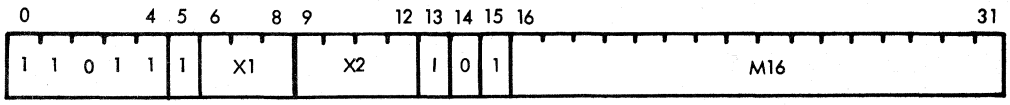


The contents of the Program Interrupt Mask Register replace the contents of the effective address according to the following format:

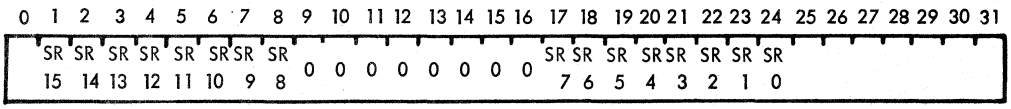


No registers are affected. Neither the immediate option nor the halfword option should be used with the STI instruction. The instruction does not have return-to-memory capability.

STS: Store Status Register

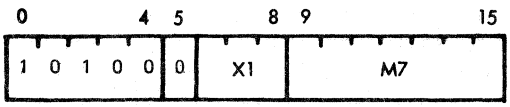


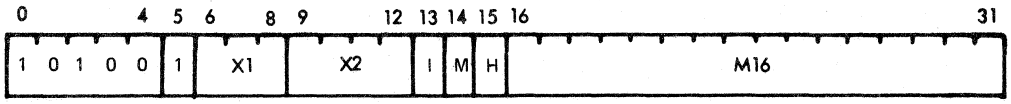
The contents of the Status Register replaces the contents of the effective address according to the following format:



For example: Bit 12 of the Status Register (SR12) is stored in bit 4 of the designated memory word. No registers are affected. The immediate option should not be used. If bit 15 of the instruction is set to 0, this instruction becomes an SFD instruction. This instruction does not have return-to-memory or halfword capability.

ADU: Add Upper Fixed Point Fullword





This instruction operates on fullwords; in the long format H = 0 and when a short format ADU is executed, the status register must be set to fullword mode (SR5 = 0). The fullword in memory, designated by the effective address, is algebraically added to the contents of the A Register. If both SR12 = 1 and SR13 = 1 (indicating a Carry from the MSB of the B Register on a preceeding ADL addition), then a bit is also added into the LSB position of the result. The final result is placed in the A Register (Bits 0-31).

Status Register bits SR12 and SR13 are reset to zero. The PC is incremented by 1 for the short format and by 2 for the long format. No other register is affected.

If an immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are added to the least significant half of the A Register (Bits 16-31). More precisely, the 16 bit operand is treated as a 32 bit word with the sign bit (Bit 16) extended during the addition.

ADUH: Add Upper Fixed Point Halfword

This instruction operates on halfwords; in the long format H = 1 and when a short format ADUH is executed, the status register must be set to halfword mode (SR5 = 1). The halfword in memory designated by the effective address is algebraically added to the most significant half of the A Register (Bits 0-15). Zeroes are added to the least significant half of the register (Bits 16-31). The contents of the effective address are unchanged. If both SR12 = 1 and SR13 = 1 (indicating a carry from the MSB of the B Register on a preceeding addition) then a bit is also added into the LSB position of the result. The final result is placed in the A Register.

Status Register bits SR12 and SR13 are reset to zero. The PC is incremented by 1 for the shoft format and by 2 for the long format. No other registers are affected.

If an immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are added to the most significant half of the A Register (Bits 0-15), as above.

ADUR: Add Upper and Return

This instruction generates the same result as the ADU instruction. However, the 32 bit result is stored in the Fast Scratchpad from which it obtained the operand. The contents of the A Register do not get modified. If the operand address is not in the Fast Scratchpad Memory, the instruction behaves like a NOP.

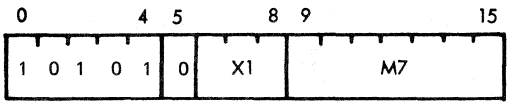
The ADUR is distinguished from the ADU in that the X1 field must contain a 7 for the ADUR and cannot contain a 7 for the ADU.

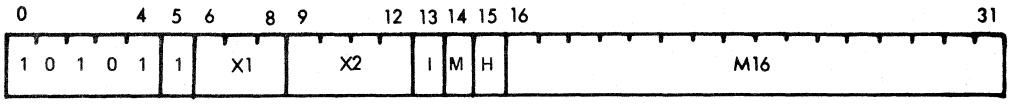
ADUHR: Add Upper Halfword and Return

This instruction generates the same result as the ADUH instruction. However, the 16 bit result is stored in the Fast Scratchpad cell from which it obtained the operand. The contents of the A Register do not get modified. If the operand address is not in the Fast Scratchpad Memory, the instruction behaves like a NOP.

The ADUHR is distinguished from the ADUH in that the X1 field must contain a 7 for the ADUHR and cannot contain a 7 for the ADUH.

ADL: Add Lower Fixed Point Fullword





This instruction operates on fullwords; in the long format H = 0 and when a short format ADL is executed, the status register must be set to fullword mode (SR5 = 0). The fullword in memory, designated by the effective address, is algebraically added to the contents of the B Register. The final result is placed in the B Register (Bits 0-31).

If a carry is generated from the MSB position (Bit 0), both SR12 and SR13 are set to one. Otherwise, SR13 is set to one and SR12 to zero. The PC is incremented by 1 for the short format and by 2 for the long format. No other register is affected.

If an Immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are added to the least significant half of the B Register (Bits 16-31). More precisely, the 16 bit operand is treated as a 32 bit word with the sign bit (Bit 16) extended during addition.

ADLH: Add Lower Fixed Point Halfword

This instruction operates on halfwords; in the long format H = 1 and when a short format ADLH is executed, the status register must be set to halfword mode (SR5 = 1). The halfword in memory, designated by the effective address, is algebraically added to the most significant half of the B Register (Bits 0-15). Zeroes are added to the least significant half of the register (Bits 16-31).

If a carry is generated from the MSB position (Bit 0), both SR12 and SR13 are set to one. Otherwise, SR13 is set to one and SR12 to zero. The PC is incremented by 1 for the short format and by 2 for the long format. No other register is affected.

If an Immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are added to the most significant half of the B Register (Bits 0-15). Zeroes are added to the least significant half of the register (Bits 16-31).

ADLR: Add Lower and Return

This instruction generates the same result as the ADL instruction. However, the 32 bit result is stored in the Fast Scratchpad from which it obtained the operand. The contents of the B Register do not get modified. If the operand address is not in the Fast Scratchpad Memory, the instruction behaves like a NOP.

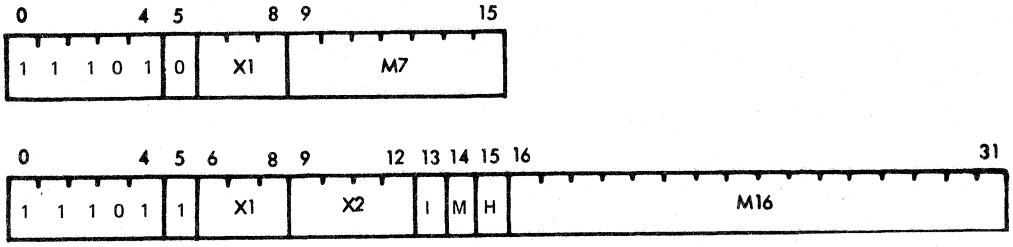
The ADLR is distinguished from the ADL in that the X1 field must contain a 7 for the ADLR and cannot contain a 7 for the ADL.

ADLHR: Add Lower Halfword and Return

This instruction generates the same result as the ADLH instruction. However, the 16 bit result is stored in the Fast Scratchpad cell from which it obtained the operand. The contents of the B Register do not get modified. If the operand address is not in the Fast Scratchpad Memory, the instruction behaves like a NOP.

The ADLHR is distinguished from the ADLH in that the X1 field must contain a 7 for the ADLHR and cannot contain a 7 for the ADLH.

SBU: Subtract Upper Fixed Point Fullword



This instruction operates on fullwords; in the long format H = 0 and when a short format SBU is executed, the status register must be set to fullword mode (SR5 = 0). The fullword in memory, designated by the effective address, is algebraically subtracted from the contents of the A Register. If both SR12 = 1 and SR13 = 1 (indicating a borrow from the MSB of the B Register on a preceeding SBL subtraction), then a bit is also subtracted from the LSB position of the result. The final result is placed in the A Register (Bits 0-31).

Status Register bits SR12 and SR13 are reset to zero. The PC is incremented by 1 for the short format and by 2 for the long format. No other register is affected.

If an immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are subtracted from the least significant half of the A Register (Bits 16-31). More precisely, the 16 bit operand is treated as a 32 bit word with the sign bit (Bit 16) extended during the addition.

SBUH: Subtract Upper Fixed Point Halfword

This instruction operates on halfwords; in the long format H = 1 and when a short format SBUH is executed, the status register must be set to halfword mode (SR5 = 1). The halfword in memory designated by the effective address is algebraically subtracted from the most significant half of the A Register (Bits 0-15). Zeroes are added to the least significant half of the register (Bits 16-31). The contents of the effective address are unchanged. If both SR12 = 1 and SR13 = 1 (indicating a borrow from the MSB of the B Register on a preceeding subtration) then a bit is also subtracted from the LSB position of the result. The final result is placed in the A Register.

Status Register bits SR12 and SR13 are reset to zero. The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

If an immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are subtracted from the most significant half of the A Register (Bits 0-15), as above.

SBUR: Subtract Upper and Return to Memory

This instruction generates the same result as the SBU instruction. However, the 32 bit result is stored in the Fast Scratchpad cell from which it obtained the operand. The contents of the A Register are not modified. If the operand address is not in the Fast Scratchpad Memory, the instruction behaves like a NOP.

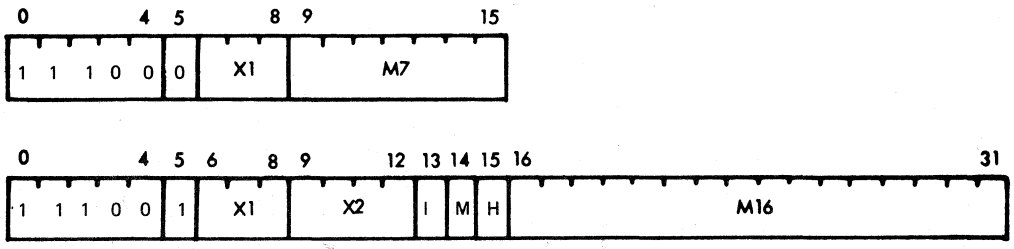
The SBUR is distinguished from the SBU in that the X1 field must contain a 7 for the SBUR and cannot contain a 7 for the SBU.

SBUHR: Subtract Upper Halfword and Return to Memory

This instruction generates the same result as the SBUH instruction. However, the 16 bit result is stored in the Fast Scratchpad cell from which it obtained the operand. The contents of the A Register are not modified. If the operand address is not in the Fast Scratchpad Memory, the instruction behaves like a NOP.

The SBUHR is distinguished from the SBUH in that the X1 field must contain a 7 for the SBUHR and cannot contain a 7 for the SBUH.

SBL: Subtract Lower Fixed Point Fullword



This instruction operates on fullwords; in the long format H = 0 and when a short format SBL is executed, the status register must be set to fullword mode (SR5 = 0). The fullword in memory, designated by the effective address, is algebraically subtracted from the contents of the B Register. The final result is placed in the B Register (Bits 0-31).

If a borrow occurs from the MSB position (Bit 0), both SR12 and SR13 are set to one. Otherwise, SR13 is set to one and SR12 to zero. The PC is incremented by 1 for the short format and by 2 for the long format. No other register is affected.

If an Immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are subtracted from the least significant half of the B Register (Bits 16-31). More precisely, the 16 bit operand is treated as a 32 bit word with the sign bit (Bit 16) extended during subtraction.

SBLH: Subtract Lower Fixed Point Halfword

This instruction operates on halfwords; in the long format H = 1 and when a short format SBLH is executed, the status register must be set to halfword mode (SR5 = 1). The halfword in memory, designated by the effective address, is algebraically subtracted from the most significant half of the B Register (Bits 0-15). Zeroes are added to the least significant half of the register (Bits 16-31).

If a borrow occurs from the MSB position (Bit 0), both SR12 and SR13 are set to one. Otherwise SR13 is set to one and SR12 to zero. The PC is incremented by 1 for the short format and by 2 for the long format. No other register is affected.

If an Immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand. They are subtracted from the most significant half of the B Register (Bits 0-15). Zeroes are added to the least significant half of the register (Bits 16-31).

SBLR: Subtract Lower and Return to Memory

This instruction generates the same result as the SBL. However, the 32 bit result is stored in the Fast Scratchpad cell from which it obtained the operand. The contents of the B Register do not get modifed. If the operand address is not in the Fast Scratchpad Memory, the instruction behaves like a NOP.

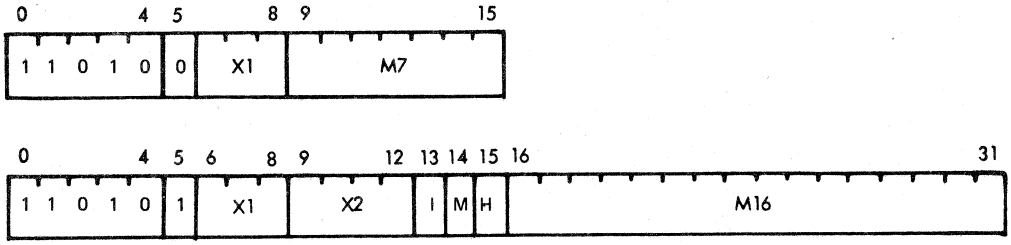
The SBLR differs from the SBL in that the X1 field must contain a 7 for the SBLR and cannot contain a 7 for the SBL.

SBLHR: Subtract Lower Halfword and Return

This instruction generates the same result as the SBLH instruction. However, the 16 bit result is stored in the Fast Scratchpad cell from which it obtained the operand. The contents of the B Register do not get modified. If the operand address is not in the Fast Scratchpad Memory, the instruction behaves like a NOP.

The SBLHR differs from the SBLH in that the X1 field must contain a 7 for SBLHR and cannot contain a 7 for the SBLH.

MUL: Multiply - Fixed Point Fullword



This instruction operates on fullwords; in the long format H = 0 and when a short format MUL is executed, the status register must be set to fullword mode (SR5 = 0). The fullword in memory, designated by the effective address, is multiplied by the contents of the A Register. Both operands are treated as fractions with binary point set between Bit 0 and 1. The 63 bit result is left in Bits 0-31 of the A Register and Bits 0-30 of the B Register. Bit 31 of the B Register is set equal to the sign of the multiplier. The contents of the effective address are unchanged.

The PC is incremented by 1 for the short format and by 2 for the long format. No other register is affected.

If an Immediate operand is designated (M=1), the least significant 16 bits of the effective address are treated as the operand. More precisely, they form the least significant half of a 32 bit operand with the sign bit (Bit 16) extended through the leading bit positions.

MULH: Multiply - Fixed Point Halfword

This instruction operates on halfwords; in the long format H = 1 and when a short format MULH is executed, the status register must be set to halfword mode (SR5=1). The halfword in memory, designated by the effective address, is multiplied by the most significant half of the A Register. Both 16 bit operands are treated as fractions. The 31 bit result is left in the A Register (Bits 0-30).

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

If an Immediate operand is designated (M=1), the least significant 16 bits of the effective address are treated as the operand.

MULR: Multiply and Return to Memory

This instruction generates the same result as the MUL instruction. However, the most significant half of the result, (Bits 0 to 31 in the A Register), are also stored in the Fast Scratchpad cell from which it obtained the operand. The A Register also retains this result. If the operand address is not in the Fast Scratchpad Memory, the result is not returned to memory but it is properly generated in the AB Register.

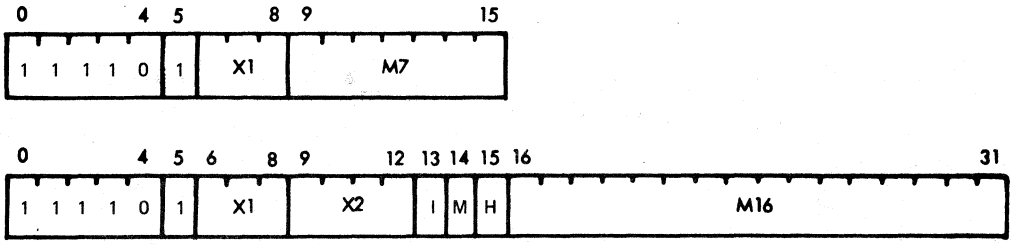
The MULR is distinguished from the MUL in that the X1 field must contain a 7 for the MULR and cannot contain a 7 for the MUL. The immediate option must not be used with MULR.

MULHR: Multiply Halfword and Return to Memory

This instruction generates the same result as the MULH instruction. However, the most significant half of the result, (Bits 0 to 15 in the A Register) are also stored in the Fast Scratchpad cell from which it obtained the operand. The A Register also retains the result. If the operand address is not in the Fast Scratchpad Memory, the result is not returned to memory but it is properly generated in the A Register.

The MULHR is distinguished from the MULH in that the X1 field must contain a 7 for the MULHR and cannot contain a 7 for MULH. The immediate option must not be used with MULHR.

DVD: Divide Fixed Point



This instruction operates on fullwords; in the long format H = 0 and when a short format DVD is executed, the status register must be set to fullword mode (SR5 = 0). The contents of the A Register is divided by the fullword in memory designated by the effective address. Both operands are treated as fractions with binary point set between Bit 0 and Bit 1. The initial contents of the A Reigster must be smaller in magnitude than the divisor in memory. The 32 bit quotient is placed in the A Register with the appropriate sign. The contents of the effective address are unchanged.

An additional 32 bits of accuracy can be obtained by transferring the B Register to the A Register, complementing the MSB, and repeating the division.

The PC is incremented by 1 for the short format and by 2 for the long format. No other register is affected.

If an Immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand. More precisely, they form the least significant half of a 32 bit operand with the sign bit (Bit 16) extended through the leading bit positions.

DVDH: Divide Fixed Point Halfword

This instruction operates on halfwords; in the long format H = 1 and when a short format DVDH is executed, the status register must be set to halfword mode (SR5 = 1). The most significant half of the A Register is divided by the halfword in memory designated by the effective address. Both 16 bit operands are treated as fractions. The 16 bit quotient appears in the A Register (Bits 0-15). The contents of the effective address are unchanged.

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

If an Immediate operand is designated (M = 1), the least significant 16 bits of the effective address are treated as the operand.

DVDR: Divide Fixed Point and Return to Memory

This instruction generates the same result as the DVD instruction. However, the quotient (Bits 0-31 in the A Register) are also stored in the Fast Scratchpad location from which it obtained the operand. The A Register also retains this result. If the operand address is not in the Fast Scratchpad Memory, the result is not returned to memory but it is properly generated in the A Register.

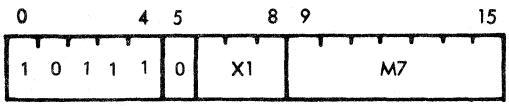
The DVDR differs from the DVD by the X1 field (Bit positions 6-8) which must contain a 7 for the DVDR and must <u>not</u> contain a 7 for the DVD. The Immediate option must not be used with DVDR.

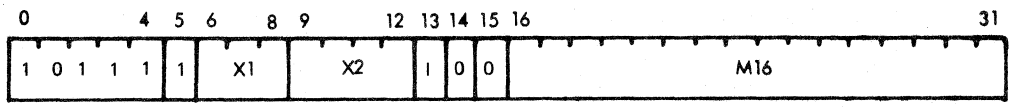
DVDHR: Divide Fixed Point Halfword and Return to Memory

This instruction generates the same result as the DVDH instruction. However, the most significant half of the A Register (the quotient) is also stored in the Fast Scratchpad location from which it obtained the operand. Bits (0-15) of the A Register also retain the result. If the operand address is not in the Fast Scratchpad Memory, the result is not returned to memory but it is properly generated in the A Register.

The DVDHR differs from the DVDH by the X1 field (Bit positions 6-8) which must contain a 7 for the DVDHR and must <u>not</u> contain a 7 for the DVDH. The Immediate option must not be used with DVDHR.

ADF: Add Floating Point





This instruction operates on fullwords only, even if the status register is set to halfword made (SR5=1) or if H=1 in the long format. It is assumed that each fullword operand is a normalized or unnormalized floating point representation. The fullword in memory designated by the effective address is, (floating), added to the fullword in the A Register. The normalized result is returned to the A Register. If the exponent underflows or overflows, an erroneous result is returned and there is no detection of the condition.

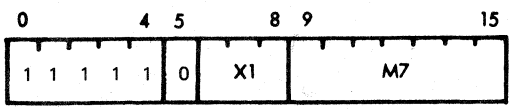
The PC is incremented by 1 for the short format and by 2 for the long format. The contents of the effective address are unchanged. No other registers are affected. The immediate and halfword designators in the long format should not be used for floating point instructions.

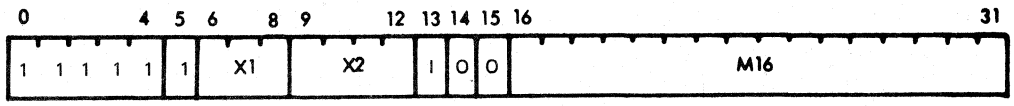
ADFR: Add Floating and Return

This instruction generates the same result as the ADF instruction. However, the result is stored in the Fast Scratchpad Memory cell from which the fullword operand was obtained. The A Register also retains the result. If the operand address is not in the Fast Scratchpad Memory, the result does not get returned to memory but it is properly generated in the A Register.

The ADFR is distinguished from the ADF in that the X1 field must contain a 7 for the ADFR and cannot contain a 7 for the ADF.

SBF: Subtract Floating Point





This instruction operates on fullwords only, even if the status register is set to halfword mode (SR5 = 1) or if H = 1 in the long format. It is assumed that each fullword operand is a normalized or unnormalized floating point representation. The fullword in memory designated by the effective address is, (floating), subtracted from the fullword in the A Register. The normalized result is returned to the A Register. If the exponent underflows or overflows, an erroneous result is returned and there is no detection of the condition.

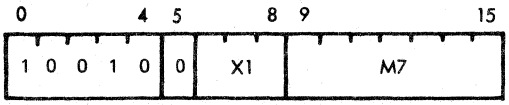
The PC is incremented by 1 for the short format and by 2 for the long format. The contents of the effective address are unchanged. No other registers are affected. The immediate and halfword designators in the long format should not be used for floating point instructions.

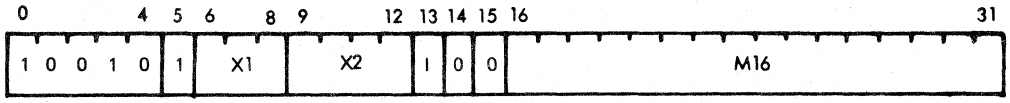
SBFR: Subtract Floating Point and Return to Memory

This instruction generates the same result as the SBF instruction. However, the result is stored in the Fast Scratchpad Memory cell from which the fullword operand was obtained. The A Register also retains the result. If the operand address is not in the Fast Scratchpad Memory, the result does not get returned to memory but it is properly generated in the A Register.

The SBFR is distinguished from the SBF in that the X1 field must contain a 7 for the SBFR and cannot contain a 7 for the SBF.

MLF: Multiply Floating Point





This instruction operates on fullwords only, even if the status register is set to halfword mode (SR5=1) or if H=1 in the long-format. It is assumed that each fullword operand is a normalized floating point representation. Any unnormalized number is treated as zero and the execution time is reduced. The fullword in memory, designated by the effective address, is multiplied by the fullword in the A Register. The normalized double length result (8 bit exponent and 47 bit mantissa), is left in the A Register, (Bits 0 to 31), and the leftmost portion of the B Register, (Bits 0-22). Note that this is the proper format for double precision accumulation using AFD, although the single precision result is also directly available in the A Register. If the exponent underflows or overflows, an erroneous result is returned and there is no detection of the condition.

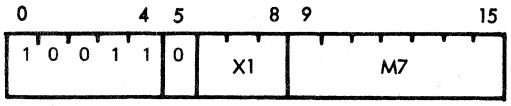
The PC is incremented by 1 for the short format and by 2 for the long format. The contents of the effective address are unchanged. No other registers are affected. The immediate and halfword designators in the long format should not be used for floating point instructions.

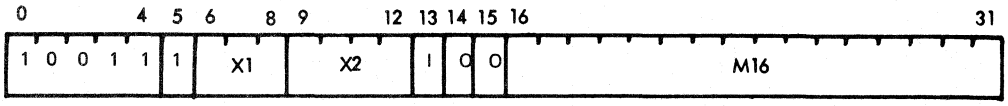
MLFR: Multiply Floating and Return

This instruction generates the same result as the MLF instruction. However, the most significant 32 bits of the result are also stored in the Fast Scratchpad Memory cell from which the fullword operand was obtained. The A Register also retains the result. If the operand address is not in the Fast Scratchpad memory, the result does not get returned to memory but is properly generated in the AB Register.

The MLFR is distinguished from the MLF in that the X1 field must contain a 7 for the MLFR and cannot contain a 7 for the MLF.

AFD: Add Floating Double-Precision





The double precision (64 bit) floating point number in the location designated by the effective address of the location containing the leftmost 32 bits is added to the double precision floating point word in the combined A,B Register. The result is placed in the A,B Registers. The contents of the effective address are unchanged. An underflow or overflow of the exponent causes return of an erroneous result and there is no detection of the condition.

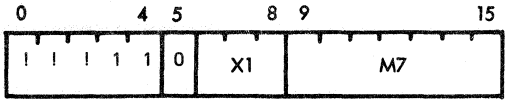
The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected. The immediate and halfword designators should not be used.

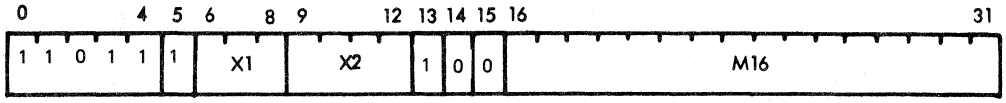
AFDR: Add Floating Double-Precision and Return to Memory

This instruction generates the same result as the ADF instruction. However, the most significant half of the result (Bits 0 to 31 in the A Register) are also stored in the Fast Scratchpad cell from which the leftmost 32 bits of the operand were obtained, i.e., the leftmost cell of the two containing the operand. The A,B Registers also retain the result. If the operand is not in the Fast Scratchpad Memory, the result is not returned to memory, but it is properly generated in the A,B Registers.

The AFDR instruction is distinguished from the AFD in that the X1 field must contain a 7 for the AFDR and cannot contain a 7 for the AFD.

SFD: Subtract Floating Double Precision





The double-precision (64 bit) floating point number in the location designated by the effective address of the location containing the leftmost 32 bits is subtracted from the double-precision floating point word in the combined A,B Registers. The result is placed in the A,B Registers. The contents of the effective address are unchanged. An underflow or overflow of the exponent causes return of an erroneous result and there is no detection of the conditions.

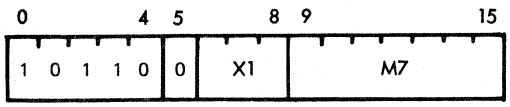
The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected. The immediate and halfword designators should not be used.

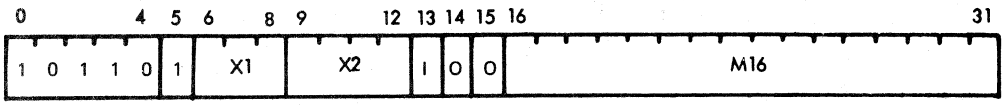
SFDR: Subtract Floating Double Precision and Return to Memory

This instruction generates the same result as the SFD instruction. However, the most significant half of the result (Bits 0 to 31 in the A Register) are also stored in the Fast SCratchpad cell from which the leftmost 32 bits of the operand were obtained, i.e., the leftmost cell of the two containing the operand. The A,B Registers also retain the result. If the operand is not in the Fast Scratchpad Memory, the result is not returned to memory, but it is properly generated in the A,B Registers.

The SFDR instruction is distinguished from the SFD in that the X1 field must contain a 7 for the SFDR and cannot contain a 7 for SFD.

DVF: Divide Floating Point





This instruction operates on fullwords only, even if the status register is set to halfword mode (SR5 = 1) or if H = 1 in the long format. It is assumed that each fullword operand is a normalized floating point representation. Any unnormalized number is treated as zero and the execution time is reduced. The fullword in memory, designated by the effective address, is divided into the fullword in the A Register. The normalized quotient (8 bit exponent and 23 bit mantissa), is left in the A Register, (Bits 0 to 31). If the exponent underflows or overflows, an erroneous result is returned and there is no detection of the condition.

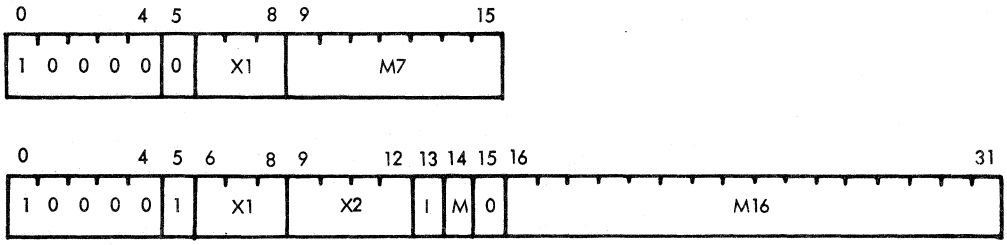
The PC is incremented by 1 for the short format and by 2 for the long format. The contents of the effective address are unchanged. No other registers are affected. The immediate and halfword designators in the long format should not be used for floating point instructions.

DVFR: Divide Floating Point and Return

This instruction generates the same result as the DVF instruction. However, the most significant 32 bits of the result are also stored in the Fast Scratchpad Memory cell from which the fullword operand was obtained. The A Register also retains the result. If the operand address is not in the Fast Scratchpad Memory, the result is not returned to memory but is properly generated in the AB Register.

The DVFR is distinguished from the DVF in that the X1 field must contain a 7 for the DVFR and cannot contain a 7 for the DVF.

AND: Logical AND



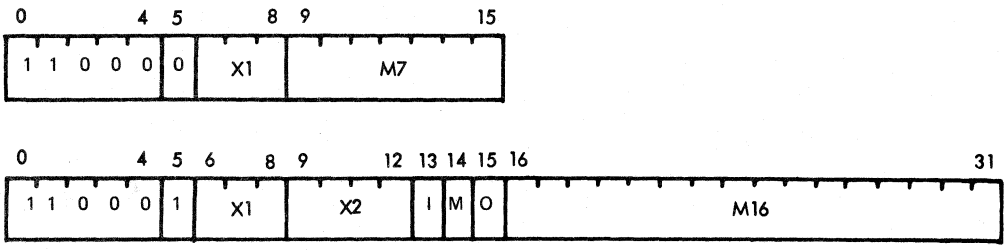
The logical product of the contents of the effective address and the A Register is placed in the A Register. For each zero in the contents of the effective address a zero is placed in the corresponding Bit position of the A Register. For each one in the contents of the effective address, the corresponding Bit position in the A Register remains unchanged. The contents of the Effective Address remain unchanged.

The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

ANDR: Logical AND and Return to Memory

This instruction generates the same result as the AND instruction. However, the result is stored in the Fast Scratchpad location from which the operand was fetched. If the operand is not in Fast Scratchpad memory, the instruction behaves like a NOP.

LOR: **Logical OR**



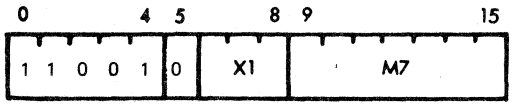
The logical sum of the contents of the effective address and the A Register is placed in the A Register. For each 1 in the contents of the effective address a 1 is placed in the corresponding Bit of the A Register. Otherwise the Bit is unchanged. The contents of the effective address are unchanged.

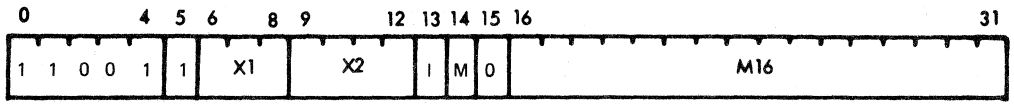
The PC is incremented by 1 for the short format and by 2 for the long format. No other registers are affected.

LORR: Logical OR and Return to Memory

This instruction generates the same result as the LOR instruction. However, the result is stored in the Fast Scratchpad location from which the operand was fetched. If the operand is not in Fast Scratchpad memory, the instruction behaves like a NOP.

EXO: Exclusive OR





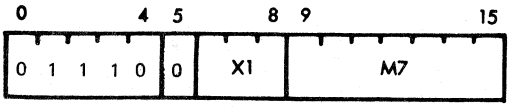
The logical difference of the contents of the effective address and the A Register is placed in the A Register. For each 1 in the contents of the effective address, the corresponding Bit position of the A Register is complemented. For each 0 in the contents of the effective address, the corresponding Bit position of the A Register is unchanged. The contents of the effective address are unchanged. No other registers are affected.

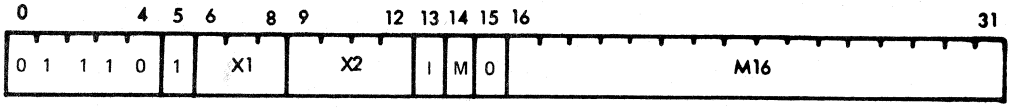
The PC is incremented by 1 for the short format and by 2 for the long format.

EXOR: Exclusive OR and Return to Memory

This instruction generates the same result as the EXO instruction. However, the result is stored in the Fast Scratchpad location from which the operand was fetched. If the operand is not in Fast Scratchpad memory, the instruction vehaves like a NOP.

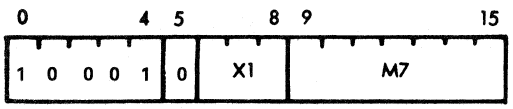
RTA: Return Address

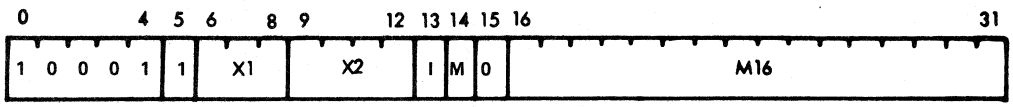




Bits 14 through 31 of the contents of the effective address are placed in the program counter and the next instruction is taken from the specified location. No other registers are affected. The return address instruction is used to return to the main program after execution of a subroutine or after processing a program interrupt request. Execution of an RTA instruction to return from a program interrupt enables the lower priority interrupts. Interrupt returns are automatically distinguished by testing the designated effective address against the location used for storing interrupt return addresses.

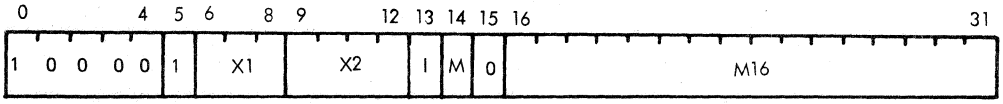
SAM: Skip On A Register Masked





For each one-bit in the contents of the effective address, the corresponding bit position of the A Register is examined. If one or more of the examined bits is a one, the program counter is incremented by 3 or 4. If the SAM instruction is short, the program counter is incremented by 3. If the SAM instruction is long the program counter is incremented by 4. If all the examined bits are 0, the next instruction is executed. The contents of the effective address and the A Registers are unchanged.

PTR: Pointer



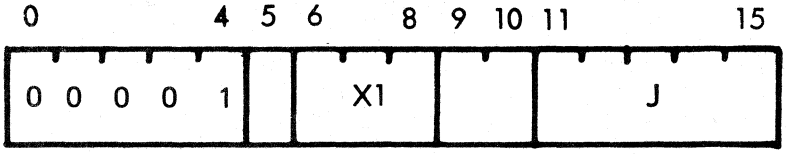
This is not an executable instruction. It is assembled in the format of a basic instruction to serve as an address pointer for indirect operations. It has all the addressing attributes of a long instruction including double indexing, indirect, and immediate modes.

3.2 SHIFT INSTRUCTIONS

All shift instructions are 16 bits long. They do not reference memory and may be indexed by a first level register. The shift count E is computed by

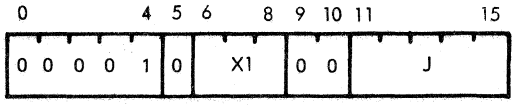
$$E = [(X1) + J] \text{ modulo } 256, \text{ ie, } 0 \leq E \leq 255.$$

The basic format of the shift instructions is shown below:



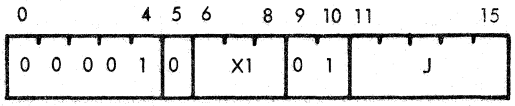
Bits 0 - 4	Basic Operation Code (Fixed)
Bits 5,9, 10	Secondary Operation Code
X1	Index Designator (XR1-XR7)
J	Basic Shift Count

SLLD: Shift A, B Left Logically



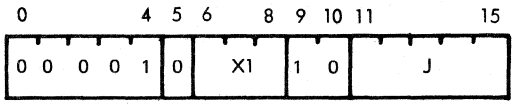
The contents of the A Register (bits 0-31) and the B Register (bits 0-31) are treated as a single 64 bit register and are shifted to the left the number of positions specified by the sum of the contents of the specified index register and J (modulo 256). Bits shifted out of the A Register are lost. Bits shifted out of the MSB of the B Register (bit 0) enter the low-order position of the A Register (bit 31). Zeroes are shifted into the vacated low-order positions of the B Register (bit 31).

SLCD: Shift A, B Left Circularly



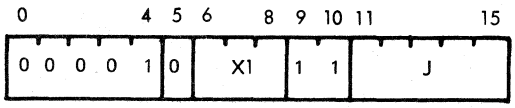
The contents of the A Register (bits 0-31) and the B Register (bits 0-31) are treated as a single 64 bit register and are rotated to the left the number of positions specified by the sum of the contents of the specified index register and J (modulo 256). Bits shifted out of the sign of the A Register (bit 0) enter the low-order position of the B Register (bit 31). Bits shifted out of the sign position of the B Register (bit 0) enter the low-order position of the B Register bit 31.

SLL: Shift A Left Logically



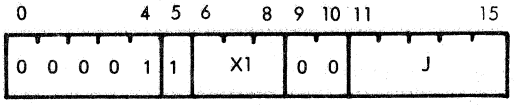
The contents of the A Register (bits 0-31) are shifted left the number of positions specified by the sum of the contents of the specified index register and J (modulo 256). Bits shifted out of the A Register (bit 0) are lost. Zeroes are shifted into the vacated low-order positions of the A Register (bit 31). A shift of 32 or more places will fill the A Register with zeroes.

SRLD: Shift A, B Right Logically



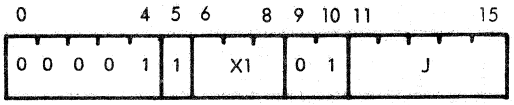
The contents of the A Register (bits 0-31) and the B Register (bits 0-31) are treated as a single 64 bit register, and are shifted right the number of positions specified by the sum of the contents of the specified index register and J (modulo 256). Zeroes are shifted into the high-order positions of the A Register (bit 0). Bits shifted out of the low-order position of the A Register (bit 31) enter the high-order position of the B Register (bit 0). Bits shifted out of the low-order position of the B Register (bit 31) are lost.

SRAD: Shift A, B Right Algebraically



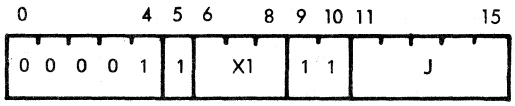
The contents of the A Register (bits 0-31) and the B Register (bits 0-31) are treated as a single 64 bit register and are shifted to the right the number of positions specified by the sum of the contents of the specified index register and J (modulo 256). The sign of the A Register is shifted into the vacated positions as shifting takes place. Bits shifted out of position 31 of the A Register enter the sign of the B Register (bit 0). Bits shifted out of the low-order position of the B Register (bit 31) are lost.

SRCD: Shift A, B Right Circularly



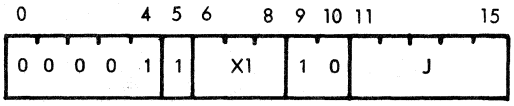
The contents of the A Register (bits 0-31) and the B Register (bits 0-31) are treated as a single 64 bit register and are rotated to the right the number of positions specified by the sum of the contents of the specified index register and J (modulo 256). Bits shifted out of the low-order position of the A Register (bit 31) enter the high-order position of the B Register (bit 0). Bits shifted out of the low-order position of the B Register (bit 31) enter the high-order position of the A Register (bit 0).

SRC: Shift A Right Circularly



The contents of the A Register (bits 0-31) are rotated to the right the number of positions specified by the sum of the contents of the specified index register and J (modulo 256). Bits shifted out of the low-order position of the A Register (bit 31) enter the high-order position of the A Register (bit 0). A shift of 32 leaves the A Register unchanged.

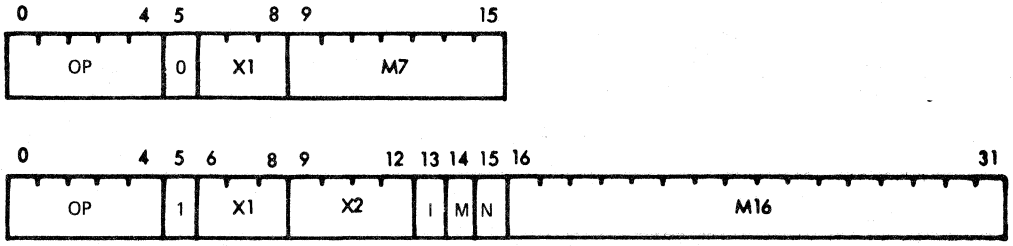
SRA: Shift A Register Right Algebraically



The contents of the A Register are shifted to the right the number of positions specified by the sum of the contents of the specified index register and J (modulo 256). The sign of the A Register remains unchanged and is shifted to the vacated positions. Bits shifted out of the low-order position of the A Register are lost. A shift greater than 32 will replace the contents of the A Reglster with all ones or zeroes depending on the sign of A.

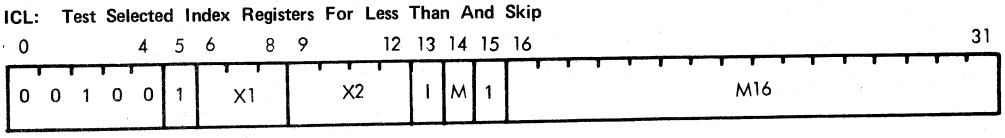
3.3 INDEX REGISTER INSTRUCTIONS

These instructions may be used to modify or test Index Registers. In the long format, the four bit field X2 is used to designate any one of the 15 index registers as the object of the operation. The Effective Address thus consists of the M16 field plus the Index Register designated in field X1. If the X1 and X2 designate the same Index Register, the Effective Address is computed before the instruction operates on the contents of the Index Register. Halfword from memory is not permitted. If the indirect option is chosen, the initially designated object register is retained, the subsequent pointer words are decoded normally to yield the effective address.



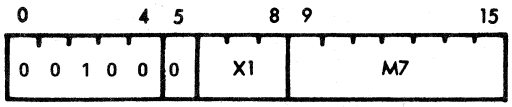
OP	Primary Operation Code
Bit 5	Long/Short Designator
X1	Index Register Selection
X2	Index Register Selection for Operation
1	Indirect Addressing Option
M	Immediate Operand Option
N	Secondary Operation Code
M7, M16	Address Fields

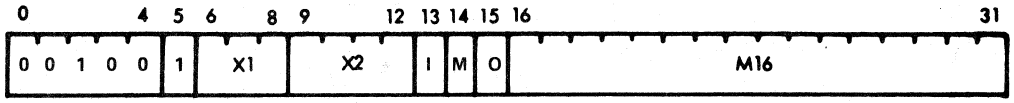
ICL: Test Selected Index Registers For Less Than And Skip



The index register designated by field X2 is compared with the contents of the effective address. The effective address is determined by the sum of M16 and the Index Register designated by X1, modified by I and M. If the contents of the index register are less than that of the effective address, the program counter is incremented by 4. If the contents of the Index Register is not less that that of the effective address, the next instruction is executed. No registers are affected. Note that this instruction must have a long format and that bit 15 must be 1.

ICN: Test Selected Index Register For Non-Equal And Skip

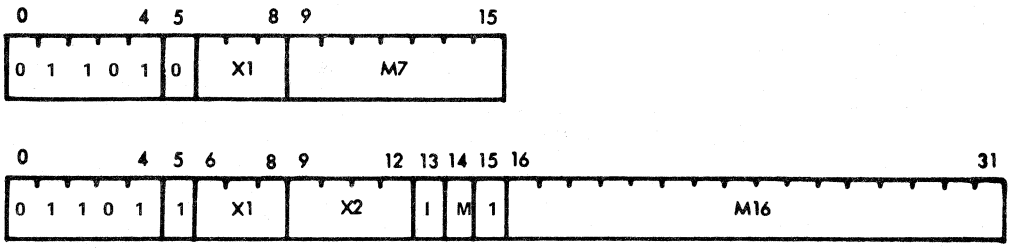




For the short format (bit 5=0), the Index Register designated in field X1 is compared to the contents of the effective address specified by M7 and Status Register bits 2, 3, and 4. For the long format (bit 5 = 1), the Index Register designated in field X2 is compared to the contents of the effective address specified by M16 plus the X1 Index Register modified by I and M. Note that bit 15 must be zero in the long format.

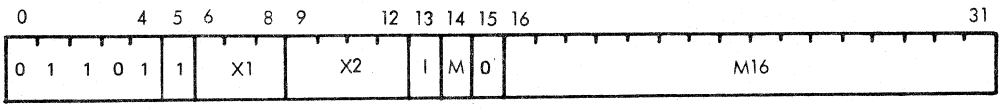
If the contents of the index register are not equal to the content of the effective address, the program counter is incremented by 3 or 4. If the ICN instruction is short, the program counter is incremented by 3. If the ICN instruction is long, the program counter is incremented by 4. If the contents of the Index Register are equal to the content of the effective address, the next instruction is executed. No registers are affected.

IMN: Modify Index Register By Negative Increments And Skip



In the short format, the contents of the Index Register designated by field X1 are decremented by the contents of the location designated by M7 and Status Register bits 2, 3, and 4. In the long format, the contents of the Index Register designated by field X2 are decremented by the contents of the effective address. For both the short and long formats, the result replaces the original contents of the Index Register designated by X2. The effective address is determined by the sum of M16 and the Index Register designated by X1, modified by I and M. No other registers are affected. Note that bit 15 must be 1 in the long format.

IMP: Modify Index Register By Positive Increment

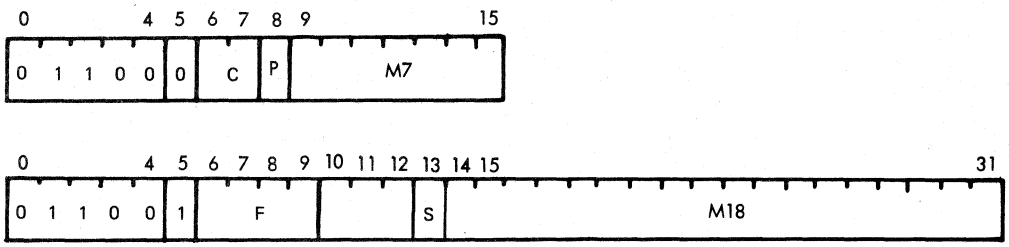


The contents of the index register designated by field X2 are incremented by the contents of the effective address. The effective address is determined by the sum of M16 and the Index Register designated by X1, modified by I and M. Note that this instruction must have a long format and that bit 15 must be zero.

3.4 JUMP INSTRUCTIONS

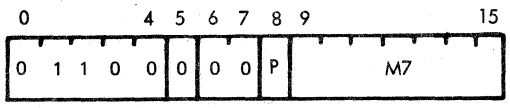
Four of these instructions have a short (16 bit) format. They may be used to perform a relative jump to locations as far as 127 short instructions away from the location of the jump instruction. Each short jump has a corresponding long jump instruction which, like all long jumps, will perform an absolute jump rather than a jump relative to the current program counter. The Assembler automatically chooses the short instruction format, if possible.

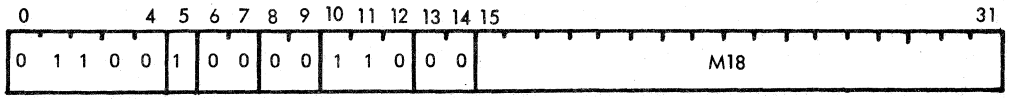
The basic jump instruction format is given below:



Bits 0-4	Primary Operation Code (Fixed)
Bit 5	Short/Long Designator
C	Jump Criteria
M7	Address Field, Relative to PC
P	Sign of M7
Bits 6-9	Flags to be tested
OP2	Secondary Operation Code
S	Designator for Storage of Return Address
M18	Address Field, Absolute Global Address

JU: Jump Unconditional

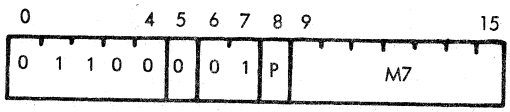


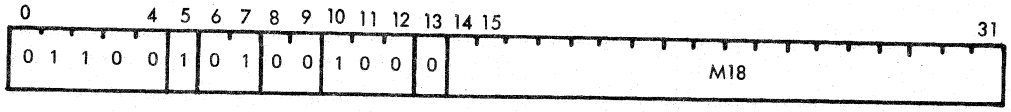


In the short format, a relative jump to within ±127 of the jump instruction location is implemented. The PC is incremented or decremented by M7. The P field (1 bit) indicates the addition of M7 (if P=0) or the subtraction of M7 (if P=1). Normal incrementing of the PC does not take place. (Incrementing PC by 1 is equivalent to not branching.) In the long format, an absolute jump to any location in memory is implemented. The CPU takes its next instruction from the memory location specified by M18. No other registers are affected.

The assembler automatically chooses the short instruction format, if possible. This decision may be overridden by using the mnemonic JRU for a short (relative) jump and the mnemonic JGU for a long (global) jump.

JN: Jump If A Not Equal Zero

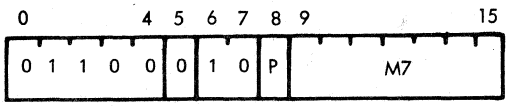


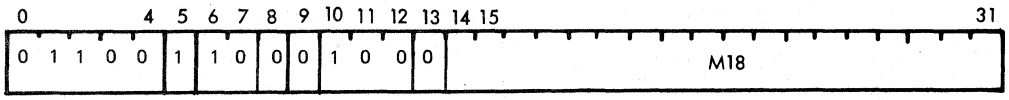


If the contents of the A Register does not equal zero, the jump is taken. Otherwise, the next instruction in sequence is executed. If the short jump is taken, the PC is incremented or decremented by M7. The P field (1 bit) indicates the addition of M7 (if P=0) or the subtraction of M7 (if P=1). Normal incrementing of the PC does not take place. (Incrementing PC by 1 is equivalent to not branching.) If the long jump is taken, an absolute jump to any location in memory is implemented. The CPU takes its next instruction from the memory location specified by M18. No other registers are affected.

The Assembler automatically chooses the short instruction format, if possible. The decision may be overridden by using the mnemonic JRN for the short (relative) jump or the mnemonic JAN for the long (global) jump.

JG: Jump If A Greater Than Or Equal To Zero

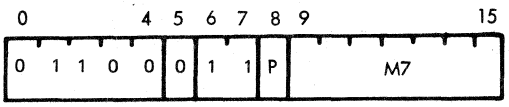


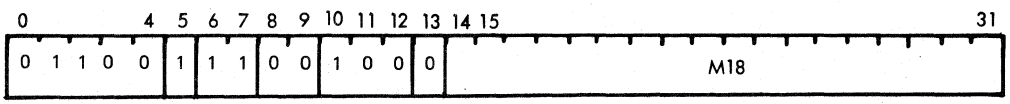


If the contents of the A Register are greater than or equal to zero, the jump is taken, otherwise, the next instruction in sequence is executed. If the short jump is taken, the PC is incremented or decremented by M7. The P field (1 bit) indicates the addition of M7 (if P=0) or the subtraction of M7 (if P=1). Normal incrementing of the PC does not take place. (Incrementing PC by 1 is equivalent to not branching.) If the long jump is taken, an absolute jump to any location in memory is implemented. The CPU takes its next instruction from the memory location specified by M18. No other registers are affected.

The Assembler automatically chooses the short instruction format, if possible. The decision may be overridden by using the mnemonic JRG for the short (relative) jump or the mnemonic JAG for the long (global) jump.

JL: Jump If A Less Than Zero

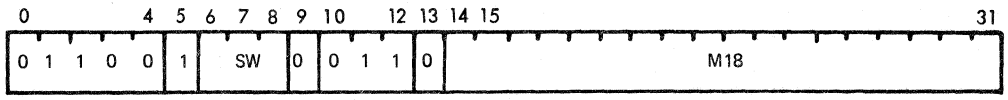




If the contents of the A Register are negative, the jump is taken otherwise, the next instruction in sequence is executed. If the short jump is taken, the PC is incremented or decremented by M7. The P field (1 bit) indicates the addition of M7 (if P=0) or the subtraction of M7 (if P=1). Normal incrementing of the PC does not take place. (Incrementing PC by 1 is equivalent to not branching.) If the long jump is taken, an absolute jump to any location in memory is implemented. The CPU takes its next instruction from the memory location specified by M18. No other registers are affected.

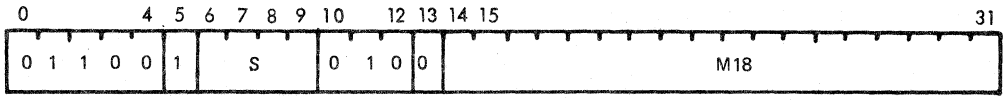
The Assembler automatically chooses the short instruction format, if possible. This decision may be overridden by using the mnemonic JRL for a short (relative) jump and the mnemonic JAL for a long (global) jump.

JGW : Jump On Switch



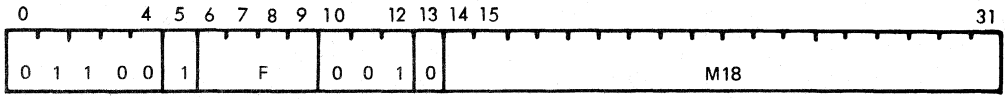
If the switch selected by the SW field of the instruction equals a one, the CPU takes its next instruction from the memory location specified by M18 otherwise, the next instruction in the sequence is executed. No registers are affected.

JGS : Jump On Status Bit



This instruction may be used to test any one of the bits in the 16 bit status register. If the status bit selected by the S field of this instruction equals a one, the CPU takes its next instruction from the memory location specified by M18 otherwise, the next instruction in sequence is executed. No registers are affected.

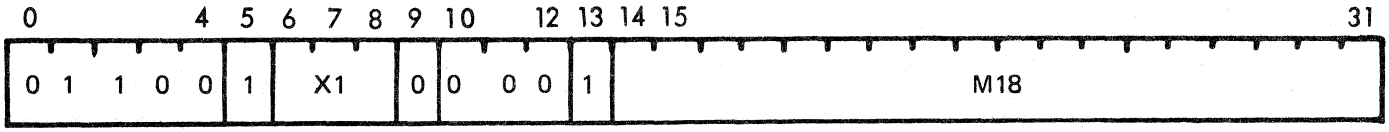
JGF: Jump On Program Flag



The four program flag selectors of the F field of this instruction are compared with the corresponding flag positions of the Status Register. If one or more of the selected flags are set, the CPU takes its next instruction from the memory location specified by M18. If no match occurs the CPU takes the next instruction in sequence. No registers are affected. The correspondence of instruction bits to status registers bits is given below.

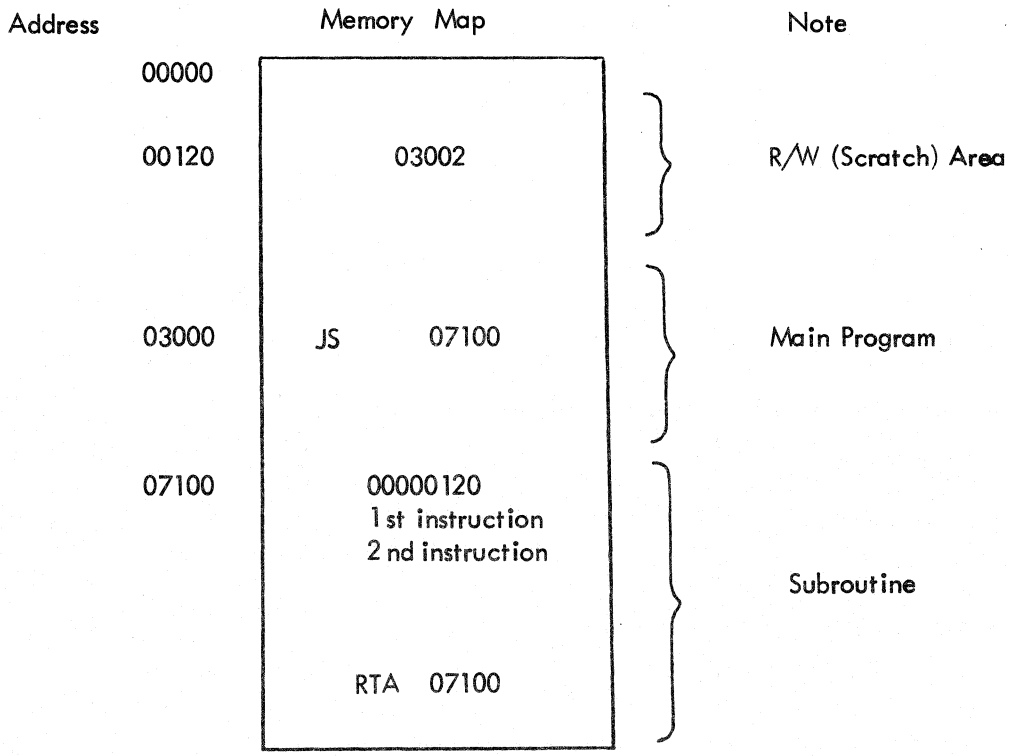
Instruction Bit	6	Tests	SR11
Instruction Bit	7	Tests	SR10
Instruction Bit	8	Tests	SR9
Instruction Bit	9	Tests	SR8

JS: Jump Unconditional To Subroutine



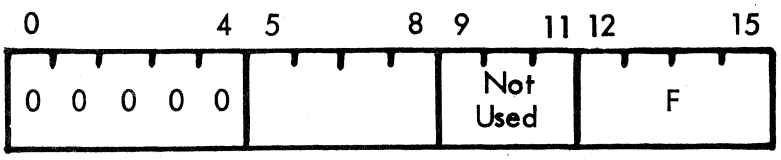
The content of the program counter is incremented by 2, and stored in the memory location specified by the pointer at the Effective Address. The CPU takes its next instruction from the memory location specified by the Effective Address +2. The Effective Address is either M18 or M18+X1, depending on the X1 bits in the instruction. Note that the Effective Address must be even. Second level indexing cannot be used. Up to 16 levels of indirect addressing are permitted in the designation of the location for storage of the return address.

This instruction is used for general subroutine entry. As an example of its use, consider the figure below. In the main (calling) program, the JS instruction at location 3000 is used to enter the subroutine which begins at location 7100. The Effective Address is M18 or Location 7100 in the example. The content of 7100 is 120, so the program counter is incremented to 3002 and is stored in location 120. The next instruction is taken from location 7102, which is the start of the subroutine. Exit from the subroutine is performed by an Indirectly Addressed RTA instruction which transfers the return address (3002) from location 120 (via 7100) to the PC, thereby returning to the main program at the instruction following the JS (i.e. location 3002).



3.5 NON-MEMORY REFERENCE INSTRUCTIONS

The non-memory reference instructions are all short and have the following basic format:



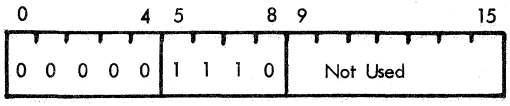
Bits 0-4 Primary Operation Code (Fixed)

Bits 5-8 Secondary Operation Code

Bits 9-11 Not Used

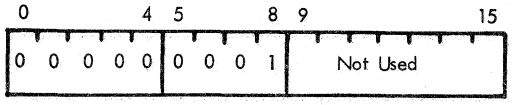
F Address Field

NOP: No Operation



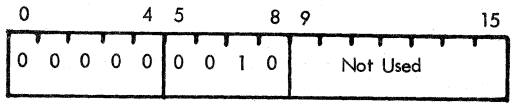
No operation is performed by this instruction. The program will take the next instruction in sequence and continue. No registers are affected. Bits 9 through 15 are not used.

EMI: Enable Memory Interrupts



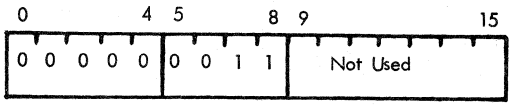
SR14 is set to 1 to allow use of the memory by units other than the CPU. Bits 9 through 15 of this instruction are not used.

DPI: Disable Program Interrupts



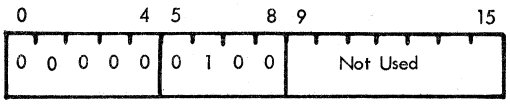
SR15 is reset to zero to disable all interrupts. Bits 9 through 15 of this instruction are not used. To insure proper operation, this instruction should be the first instruction in the interrupt routine which desires to temporarily disable all interrupts; otherwise, another interrupt may occur before this instruction is executed. The interrupt hardware will not respond to a second interrupt prior to executing the first instruction of the first interrupt's processing routine.

DMI: Disable Memory Interrupts



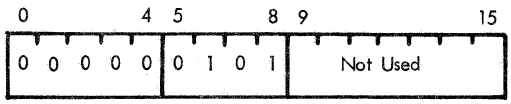
SR14 is reset to zero. This inhibits use of the memory by all units except the CPU. Bits 9 through 15 of this instruction are not used.

EPI: Enable Program Interrupts



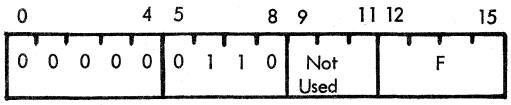
SR15 is set to 1 to enable program interrupts. Bits 9 through 15 of this instruction are not used.

HLT: Halt



If a test equipment signal is not present, the Halt becomes a NOP. No register is affected. Bits 9 through 15 are not used.

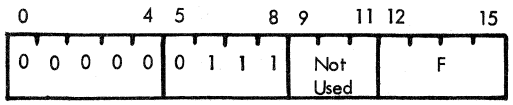
SET: Set Selected Program Flags



This instruction will set any combination of the program flags to one. Each bit of the F field controls one of the four program flags. Where the F field bit is a one, the corresponding flag is set; where the F field bit contains a zero, the corresponding flag is unchanged. The flags are bits 8 through 11 of the Status Register. No other registers are affected. Program flags might be considered indicators.

F field bit	corresponds to Status Register
15	8
14	9
13	10
12	11

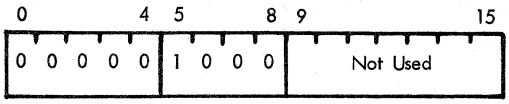
RST: Reset Selected Program Flag



This instruction will reset any combination of the program flags to zero. Each bit of the F Field controls one of the four programs flags. Where the F field bit is one, the corresponding program flag is reset; where the field bit contains a zero, the corresponding program flag is unchanged. The flags are bits 8 through 11 of the status register. Program flags might be considered indicators.

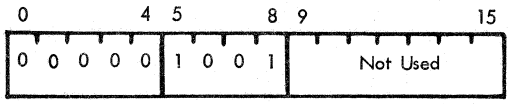
Similarly, the F field bits correspond to the Status Register bits as in SET above.

CFX: Convert Floating Point To Fixed Point



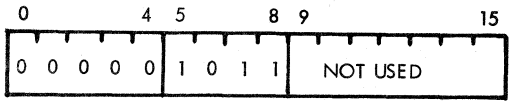
The Double Precision Floating Point number in the A,B Register is converted to an equivalent Fixed Point Number. The input number must be greater than 2^{-56} and less than 2^{+31} in magnitude and normalized. The signed integer portion of the result is left in the A Register and the fractional portion is left in the B Register. No other registers are affected.

CXF: Convert Fixed Point To Floating Point



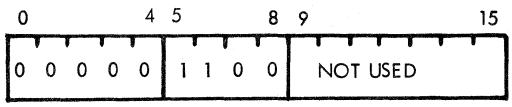
A double length fixed point number with the integer portion in the A Register and the fraction part in the B Register is converted to a double precision floating point number in the combined A,B Registers. The result is normalized. No other registers are affected.

SHM: Set Halfword Mode



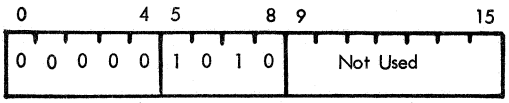
The halfword mode bit of the Status Register (SR 5) is set to one. This causes all short instructions for fixed point arithmetic to be interpreted to operate upon 16 bit halfwords. Bits 9 through 15 of this instruction are not used.

RHM: Reset Halfword Mode



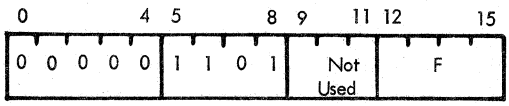
The halfword mode bit of the Status Register (SR5) is reset to zero. This causes all short instructions for fixed point arithmetic to be interpreted to operate upon 32 bit fullwords. Bits 9 through 15 of this instruction are not used.

EAB: Exchange A And B



The contents of the A and B Registers are exchanged. Bits 9 through 15 of this instruction are not used.

LXA: Load Index Register From A Register

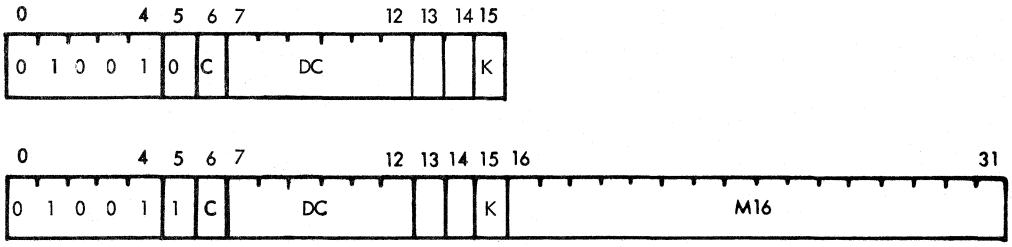


The least significant 18 bits of the A Register are placed in the Index Register designated by F(XR1-XR15). Bits 9-11 of this instruction are not used.

3.6 INPUT-OUTPUT INSTRUCTIONS

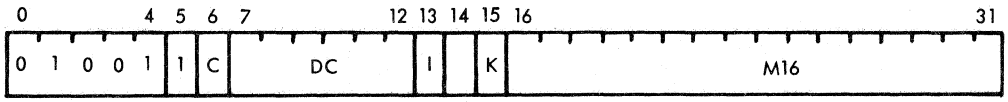
This group of instructions has two short format instructions and two long format instructions. The two short instructions cause the transfer of data from/to the A Register. The two long instructions cause the transfer of data from/to memory. The computer normally does not wait for the I/O device to respond to the command. If the wait for acknowledge bit is set (K) then the CPU may pause for a maximum of 3.75 μ sec for an acknowledge signal to be returned to the CPU. Execution of instructions is resumed upon receipt of the signal or upon the elapse of the maximum wait time. The command bit (C) is transmitted to the external device.

The basic format for I/O instruction is illustrated below:



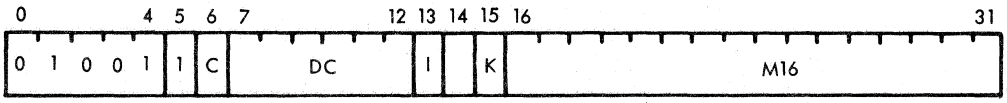
Bits 0-4	Primary Operation Code (Fixed)
Bit 5	Short/Long Designator
C	Command Bit
DC	I/O Device Designator
Bit 13	
Bit 14	
K	Acknowledge Bit
M16	Absolute Address

DIM: Data Input To Memory



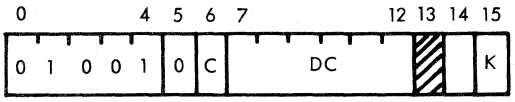
The enabled device designated by the DC Field transmits 32 bits of data into the memory at the location designated by the effective address. Indexing may be used on any indirect reference.

DOM: Data Output From Memory



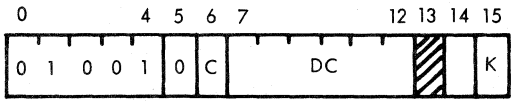
The content of the effective address in transmitted to the device designated by the DC Field. Indexing may be used on any indirect reference. No registers are affected.

DIA: Data Input To A Register



The enabled device designated by the DC field transmits 32 bits of data into the A Register. No other registers are affected.

DOA: Data Output From A Register



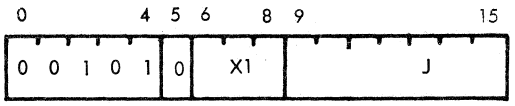
The contents of the A Register is transmitted to the device designated by the DC field. No registers are affected.

3.7 BLOCK TRANSFER INSTRUCTIONS

The two block transfer instructions are 16 bits long. They make use of the A Register, B Register, XR0, and may be indexed by a first level register. The XR group selection bits (SR0 and SR1) specify one of four index registers (XR0) used with these instructions.

These XR0 registers are store, loaded, tested, incremented, and decremented in the same way as the normal index registers with the LDX, STX, ICN, ICL, IMP, and IMN instructions. They are not, however, used for the modification of addresses when executing the Arithmetic, Shift, Jump or Index register instructions.

The basic format of the block transfer instructions is shown below:

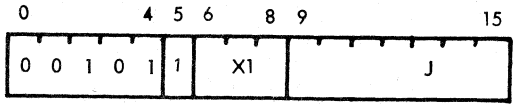


Bits 0-4	Basic Operation Code (Fixed)
Bit 5	Secondary Operation Code
X1	Index Designator (XR1-XR7)
J	Transfer Word Count

The total number (N) of fullwords to be transferred is computed by

$$N = (X1) + J$$

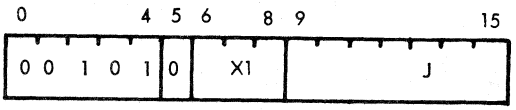
MMF: Move Main To Fast



The number of words to transferred to Fast Scratchpad Memory are specified by the sum of the contents of the specified index register and J. The beginning register in core for the transfer is specified by the initial content of the B Register +2. The beginning register in Fast Scratch Pad memory to be loaded is specified by the initial content of XR0 +2. As each word from contiguous core memory locations is transferred, it is masked with the contents of the A Register and then loaded into Scratch Pad memory.

Upon completion of the instruction execution the contents of the B Register and XR0 are (initial content) +2N. The contents of the A Register are unchanged.

MFM: Move Fast To Main



The number of words to be transferred to core are specified by the sum of the contents of the specified index register and J. The beginning register in core to be loaded is specified by the initial content of the B Register +2. The beginning register in Scratch Pad memory (Fast) for the transfer is specified by the initial content of XR0+2. As each word is transferred from (Fast), it is masked with the contents of the A Register and then loaded into core memory.

Upon completion of the instruction execution the contents of the B Register and XR0 are (initial content) +2N. The contents of the A Register are unchanged.

THIS PAGE INTENTIONALLY LEFT BLANK

4. DATA FORMATS

The SKC-2000 Computer is a word oriented binary two's complement arithmetic machine.

The arithmetic unit is designed to operate on a 32 bit fixed or floating point data words. In memory, the 32 bit word is called a fullword and must have an even address. A halfword (16 bit) arithmetic capability is also provided. Each 16 bit halfword in the memory is directly addressable. Finally, a limited double precision (64 bit) arithmetic capability is provided. The formats of these data words are defined in this section.

4.1 HEXADECIMAL NOTATION

When describing the binary representation of data in a 16 or 32 bit computer, it is common to use the Hexadecimal (powers of 16) notation. In this regard we shall follow the notation employed by IBM in their literature on the 360 Computer. Simply, it means that each binary word can be broken into 4 bit units, each of which is designated by a single alphanumeric character in accordance with the following mapping.

Binary	Hex	Binary	Hex
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

In this fashion, the 16 bit word "0111110100111011" can be represented in Hex notation by the following four digits: "7D3B", a much more abreviated form. A 32 bit word would be represented by eight Hex digits (e.g. F4A290EE).

4.2 FIXED POINT DATA FORMATS

4.2.1 SINGLE PRECISION

All fixed point data items are treated by the machine as binary fractions. The binary point is assumed between bit 0 and bit 1. Bit 0 is the sign bit. The fullword and halfword data ranges are shown in Figure 4-1.

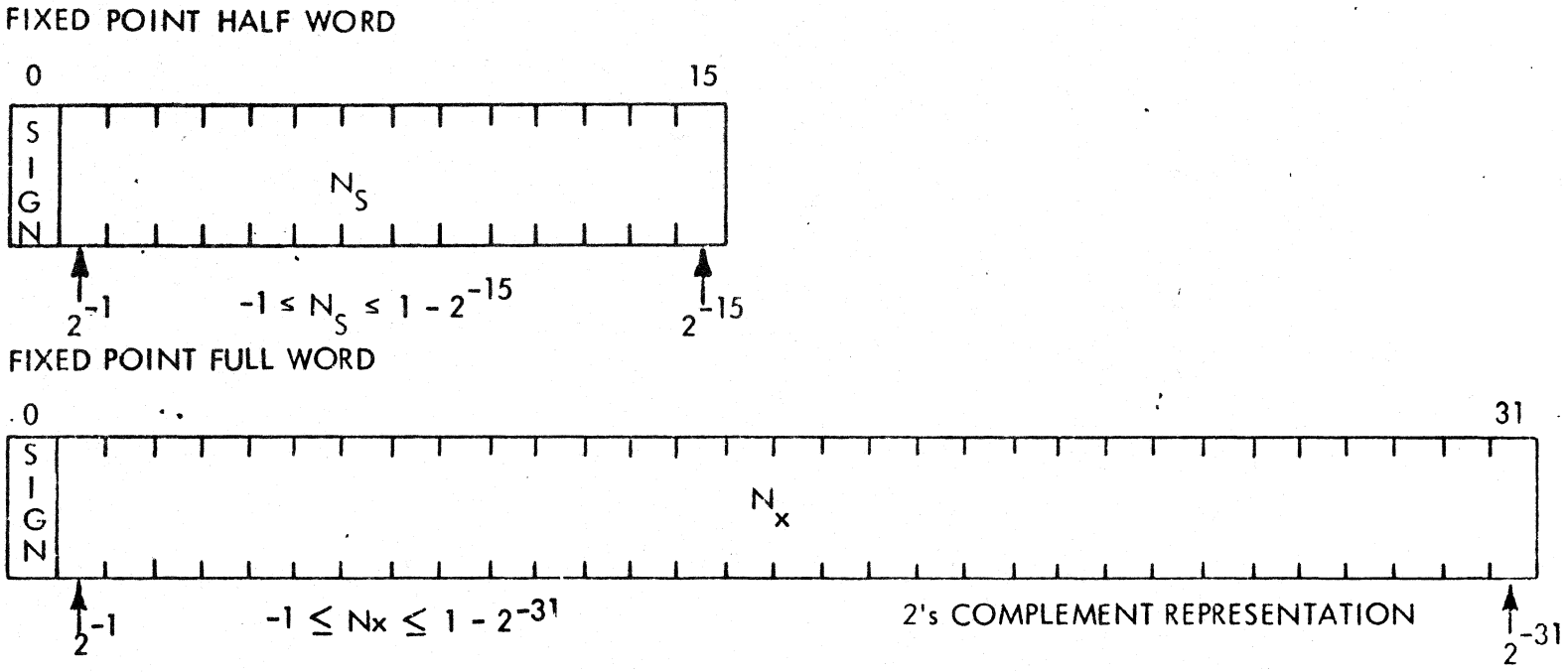


FIGURE 4-1. BASIC FIXED POINT DATA FORMAT

To illustrate, the following tabulation presents the binary equivalent (in Hex notation) of several decimal numbers. Both the fullword (32 bit) and the halfword (16 bits) form is listed:

Decimal	Fullword (Hex)	Halfword (Hex)
+0.75	60000000	6000
-0.75	A0000000	A000
$+2^{-10}$	00200000	0020
-2^{-10}	FFE00000	FFE0
$+2^{-20}$	00000800	(out of range)

4.2.2 Double Precision

A limited double precision (64 bit) arithmetic capability is provided by the SKC-2000 CPU. The A and B Registers can be treated as a single 64 bit register for the purpose of double precision summation. In addition, the multiply instruction yields a double precision product in the most significant 63 bits of the A, B Register. The divide instruction also treats the full A,B Register as the dividend. In each of these cases the following format is presumed (Figure 4-2).

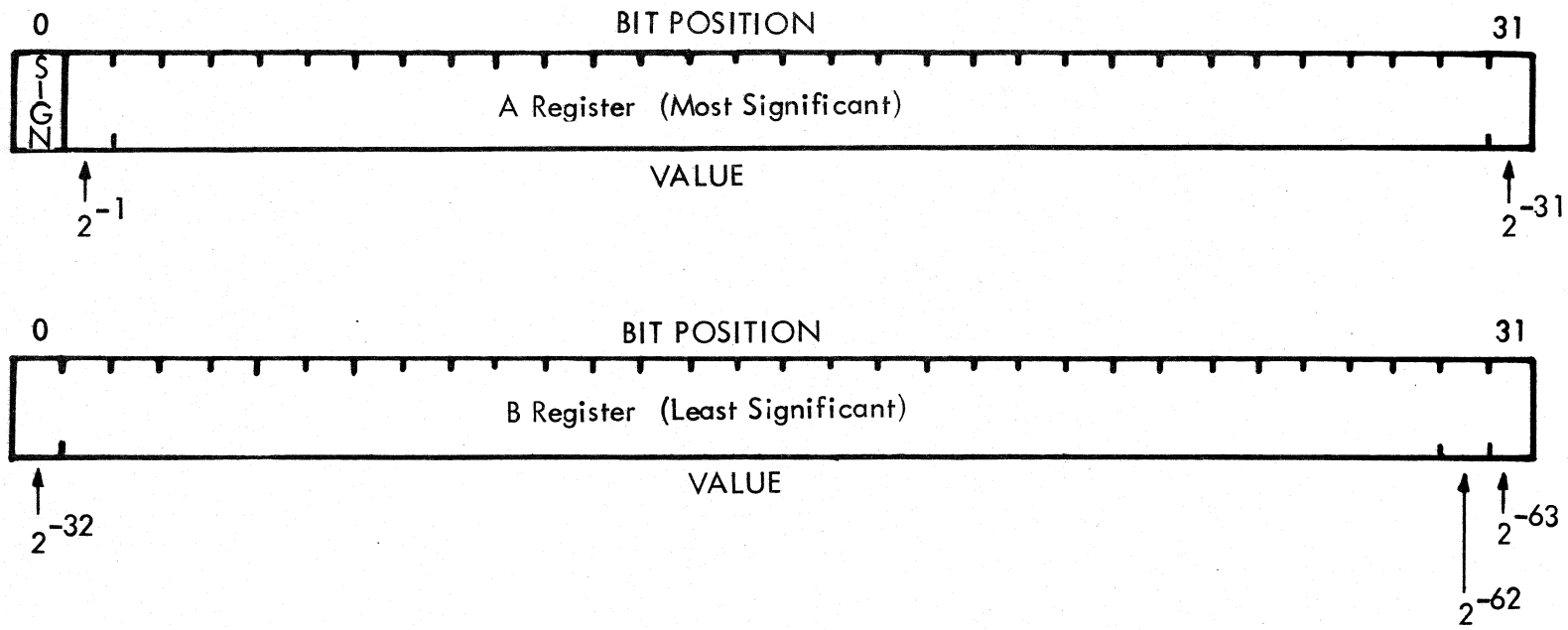


FIGURE 4-2. EXTENDED (FIXED POINT) DATA FORMAT

4.3 FLOATING POINT DATA REPRESENTATION

4.3.1 Single Precision

A single precision floating point quantity is 32 bits long and has three parts: sign, exponent, and mantissa. The sign of the data items appears in bit 0, the exponent in bits 1 to 8 and the mantissa in bits 9 to 31. The exponent is biased by 128 (decimal) so that an exponent of 128 would be used for all quantities less than 1.0 but greater than or equal to 0.5. The mantissa is always a two's complement quantity and normalized so that bit 9 is always different from bit 0. Examples of floating point words are given below.

Decimal	Floating Binary (Hex)
+1.0	40C00000
+0.5	4040000
-1.0	C000000
+0.375	3FE00000
-0.375	BFA00000

The basic floating point data format is shown in Figure 4-3.

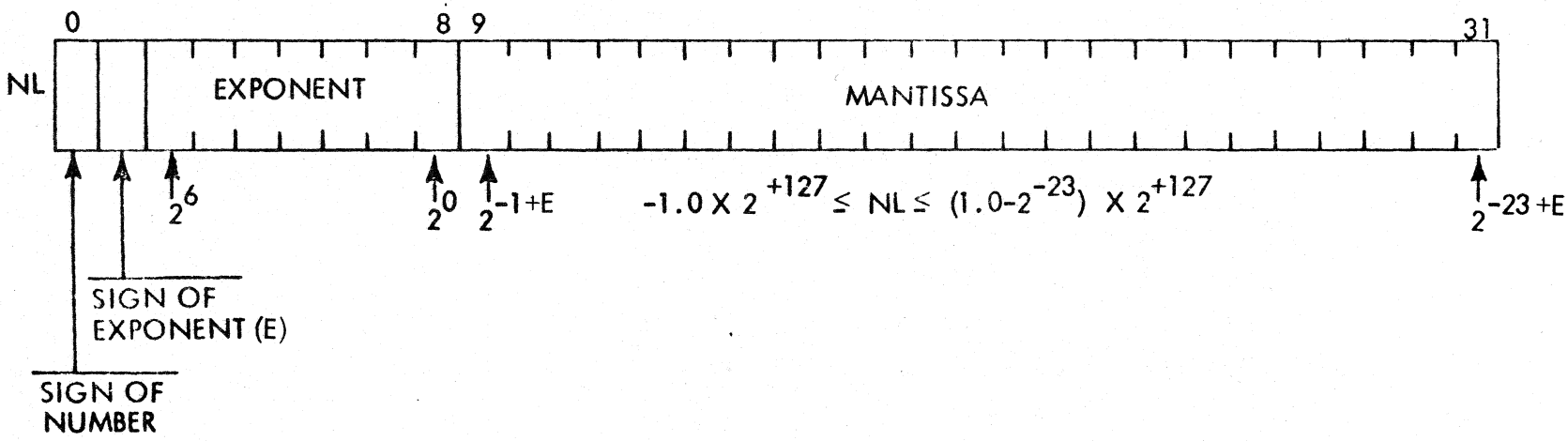


FIGURE 4-3. BASIC FLOATING POINT DATA FORMAT

4.3.2 Double Precision

The result of a floating point multiply is a double precision floating point word in the A, B Register. The leading 24 bits of the B Register contain the extension of the mantissa in the A Register and the two words may be added to another double precision quantity through the use of a double precision floating add instruction. Several other instructions also involve a double precision (64 bit) floating point quantity (i.e. DVF, CFX, CXF, SFB). The general format of the double precision floating point data word is shown in Figure 4-4.

For example, $1.0+2^{-46}$ is represented as: 40C000000000100

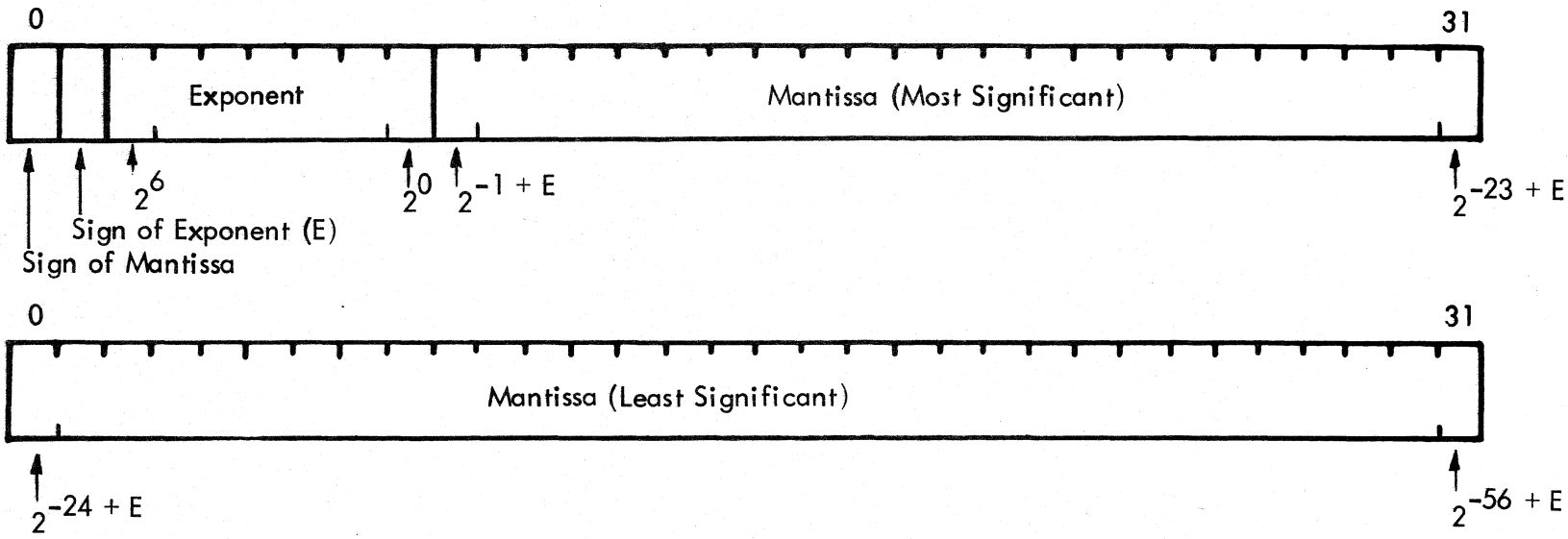


FIGURE 4-4. DOUBLE PRECISION FLOATING POINT DATA FORMAT

THIS PAGE INTENTIONALLY LEFT BLANK

THE SINGER COMPANY • KEARFOTT DIVISION

COMMENTS AND EVALUATIONS

Your evaluation of this document is welcomed by The Singer Company.

Any errors, suggested additions or general comments may be made and continued on the reverse side. Please include page number and reference paragraph and forward to:

The Singer Company
Aerospace and Marine Systems
Kearfott Division
150 Totowa Road
Wayne, New Jersey 07470
Attention: Department 5760

Name_____

Company Affiliation_____

Address_____

_____

Comments: