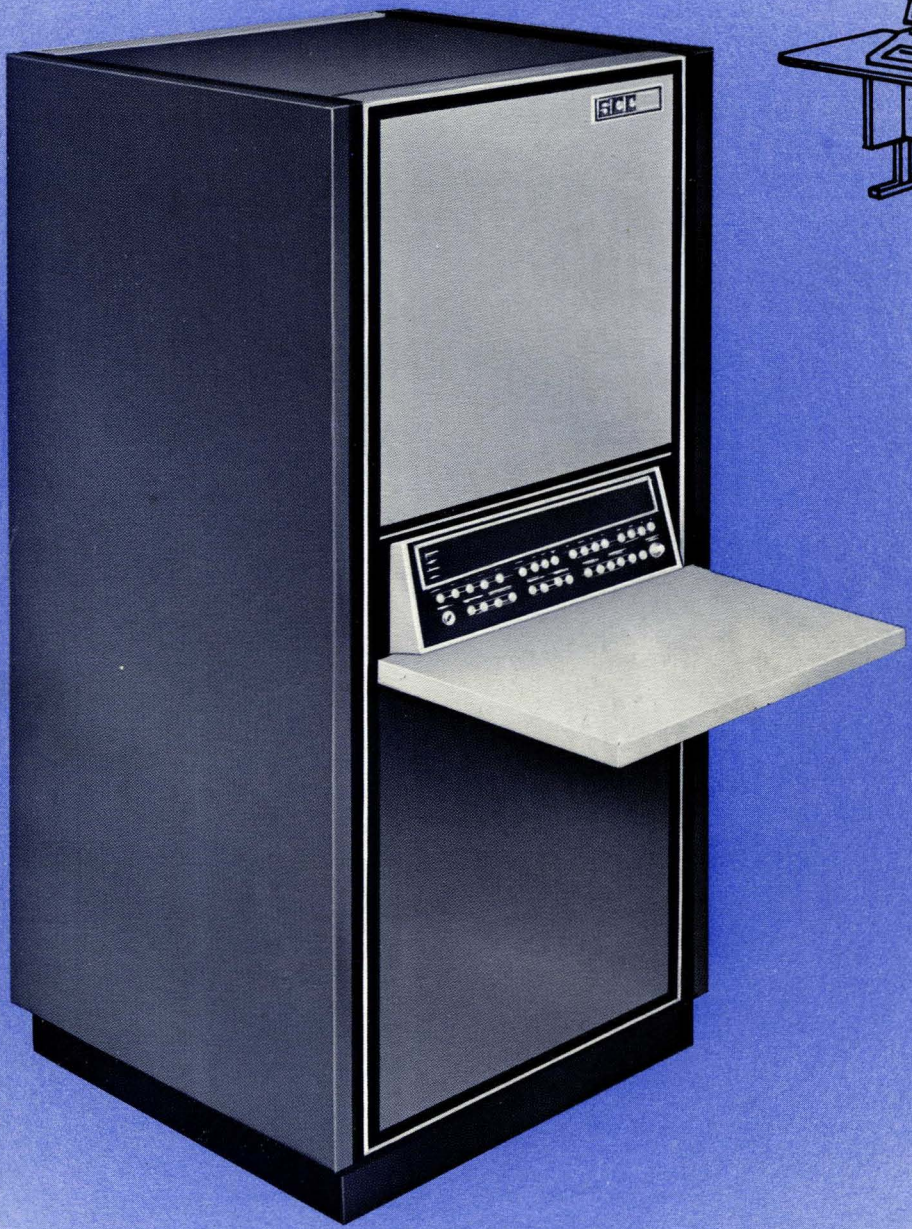
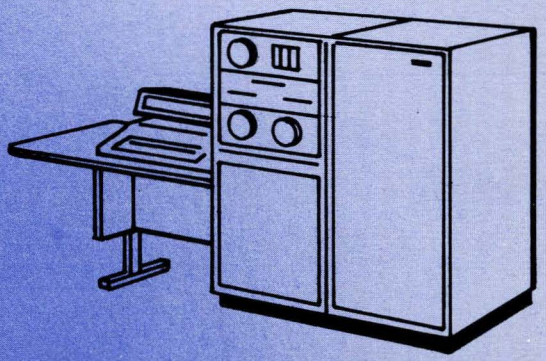
Scientific Control Corporation





SCC 4700 COMPUTER

reference manual

Price: $3.50

SCC 4700

REFERENCE MANUAL

Scientific Control Corporation

DALLAS, TEXAS 75234

P.O. BOX 34529

SCC 52-11-26-81

INTRODUCTION

The SCC 4700 represents a new design approach to the small computer field. Advanced hardware design techniques and memory mapping concepts provide a cost performance ratio that cannot be matched in any other small computer.

High speed logic forms and advanced construction techniques have been used throughout to create a "state of the art" machine which will have a high degree of reliability and avoid premature obsolescence.

The 4700 is a general purpose, high-speed, binary computer with a single address type of instruction. It features a high-speed magnetic core memory module consisting of 4096 sixteen bit words, with a 920 nanosecond cycle time which permits a wide variety of real time applications.

The 4700 has such outstanding design features as:

a. A microprogrammed read-only memory for flexible internal logic.

b. Fully integrated circuitry using the most advanced TTL integrated circuits.

c. An etched circuit back-plane board eliminating "bird nest" wiring.

d. "Register slice" internal organization for easy maintainability.

e. Programmable memory protection (optional) which provides flexible read-only, write-only, or execute-only protection.

f. Memory mapping (optional) for implementation of multiprogramming techniques.

g. Byte addressable for efficient processing of character strings, particularly those in ASCII or EBCDIC code.

h. Real time I/O structure utilizing multiplexor and high-speed selector channels for data transfer.

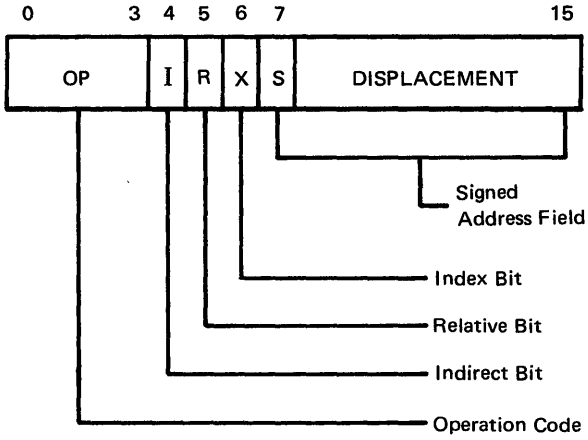
These and other advanced concepts make the 4700 faster and more flexible than any other 16-bit machine.

FORMATS

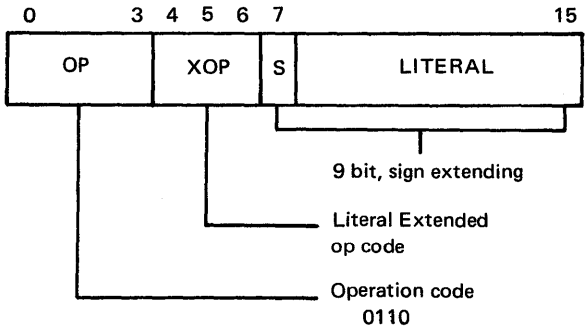
INSTRUCTION

The format for each type of instruction is given in this section. The format of each instruction is given with its description in Section IV with the operation code filled in.

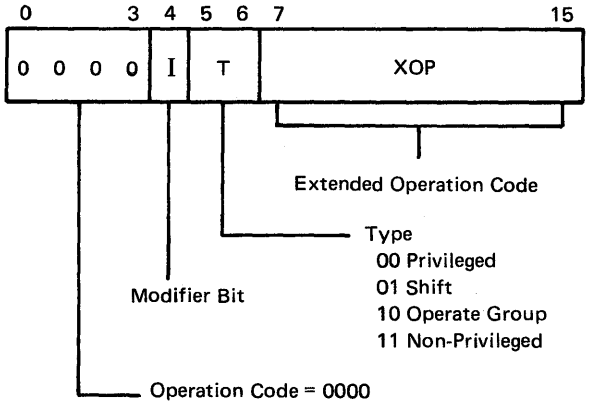
BASIC

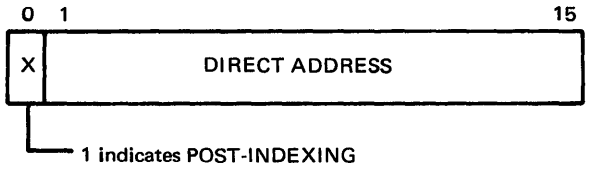


LITERAL



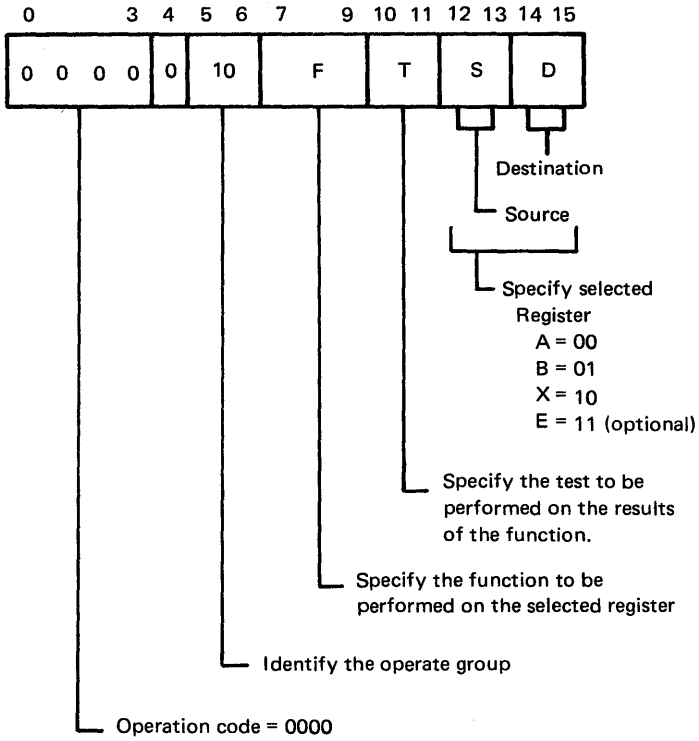
EXTENDED OP CODE



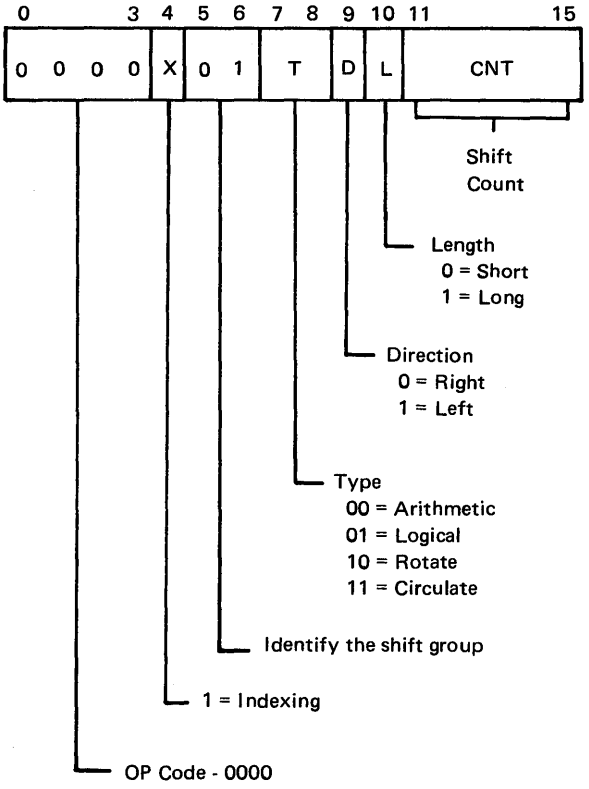


In using the indirect address word format, if the primary address of a basic instruction is the current location plus one, the location counter is incremented to skip (L+2) the next word in the instruction sequence. In this way, both operands and full addresses may be included "in-line". This indirect address technique makes possible addressing up to 32K ($7FFF_{16}$).

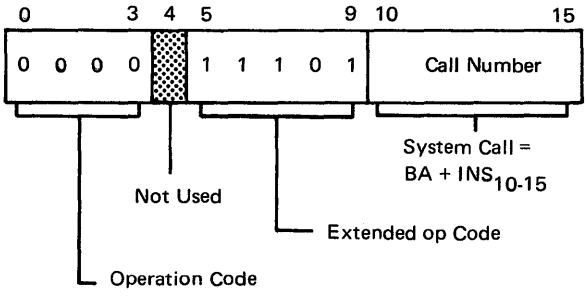
OPERATE GROUP



SHIFT



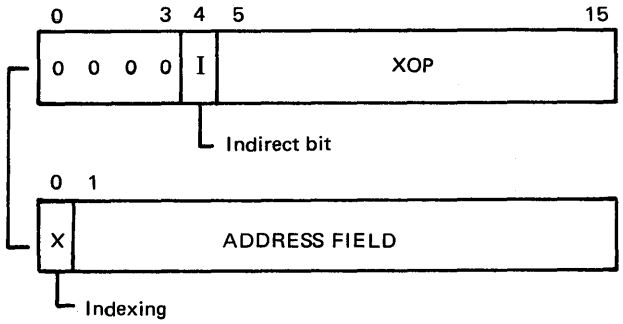
SYSTEM CALL



SYSTEM CALL

All system calls are executed from a base address (BA). A table of pointers to monitor entry points must follow the base address. As many as 64 entry points may be designated by INS_{10-15}. When encountered, a trap to $BA+INS_{10-15}$ will occur and the instruction will be executed as defined. Locations 284-347 are reserved for the table of monitor entry points used by the system call. BA is equal to 284.

DOUBLE WORD



Certain instructions require two words. The first word is in the extended op code format and the second has the format of an indirect address. There is no relative bit, because the fifteen bit address of the second word makes relative addressing unnecessary.

The instructions may be indexed and indirectly addressed, the same as other instructions. Setting bit zero of the second word specifies primary indexing. If indirect addressing (bit 4) is specified, then the second word becomes a pointer (which may be indexed) and the indirect address will be post-indexed if bit zero of the indirectly addressed location is also set.

Depending on the type of instruction, therefore, the second word will contain an address, a pointer, or data. For instance:

1. The optional arithmetic instructions require address information in the second word.
2. Input/output control commands will contain control information for the channels and devices unless indirect addressing is specified.

The address portion of this type of instruction and the word to which it points, are assembled in PAR statements. One level of indirect addressing and two levels of indexing may be specified, except in ACT and IOC instructions. These two instructions allow only one level of indexing.

ADDRESSING MODES

STANDARD

Address modification in the 4700 is based on two concepts:

PRIMARY ADDRESS

The intermediate address which is determined before indirect addressing and post-indexing are applied. It becomes the effective address if indirect addressing is not applied.

EFFECTIVE ADDRESS

The final address which is formed after all address modification and indexing have been performed.

There are five possible modes available for instructions of basic format, each of which results in a different effective address when implemented, either singly or in combination. They are:

Direct (Bit 5 = 0)

The primary address is determined by the address field of the instruction. In the direct mode (without memory mapping), the primary address always refers to the first 512 locations of memory, unless indexed.

Relative (Bit 5 = 1)

The primary address is the sum of the address field with sign extended of the instruction and the contents of the location counter.

Primary Indexed (Bit 6 = 1)

Indexing may be applied to the primary address to form the primary indexed address.

Direct indexed—the index register becomes a base register and the effective address is (X) + $(INS)_{7-15}$.

Relative indexed—the index register is added to the relative address; i.e., $[(L) \pm (INS)_{8-15}] + (X)$.

Indirect (Bit 4 = 1)

The indirect address bit is always applied after the contents of the primary address have been obtained. If the indirect address bit is zero, the contents of the location specified by the primary address is used as the operand of the instruction. If the indirect address bit is a one, the contents of the location specified by the primary address is interpreted not as an operand, but as a 15 bit operand address. (Bit 0 of the indirectly addressed location is tested for post-indexing). Indirect addressing requires one additional memory cycle (920 nanoseconds) on all instructions.

Post Indexed

In this mode, after the contents of the indirectly addressed location specified by the basic instruction are obtained, bit 0 is checked. If it is equal to one, the contents of the index register are added to the other 15 bits to form the effective operand address.

BYTE ADDRESSING

Instructions that are byte addressable, such as Load halfword, or Store Halfword, operate exactly the same as far as address modification is concerned. However, it should be remembered that the range (in number of words) will only be half that of word oriented instructions.

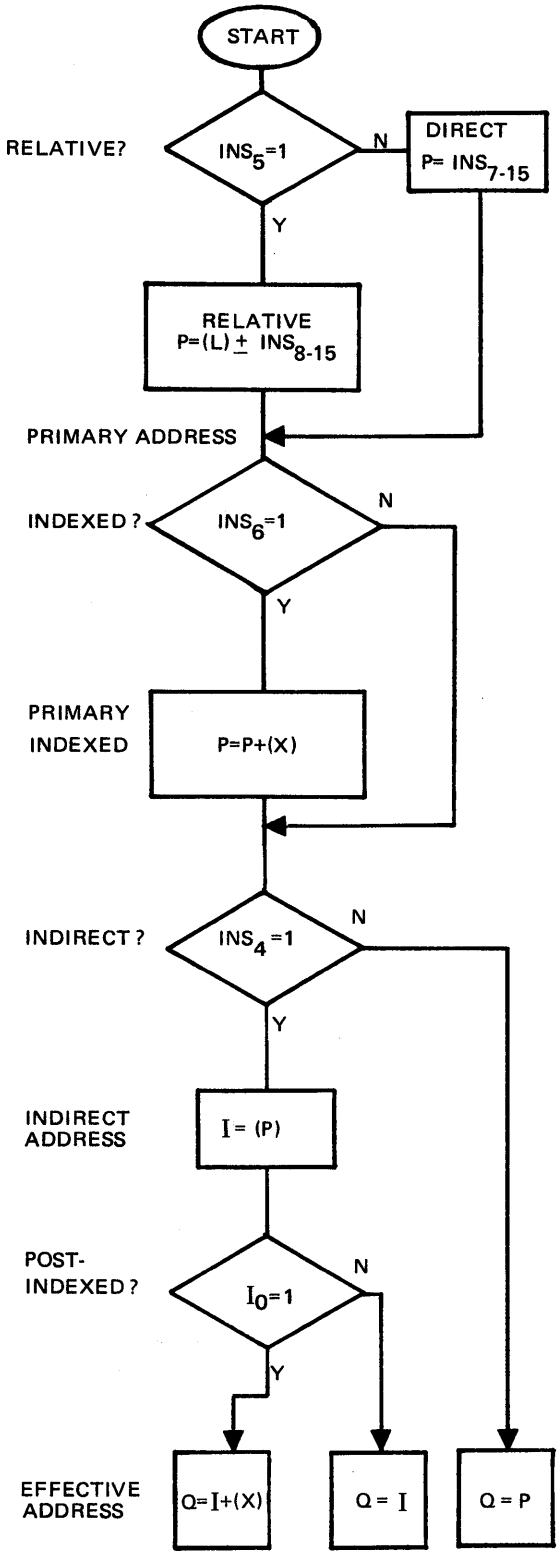
All conversion of the byte address to align the data in the proper half of the word is done automatically by the CPU.

The even numbered byte locations will be contained in the most significant eight bits of each word. For example:

Location	Byte Address	
0	0	1
1	2	3
2	4	5

BASIC ADDRESSING MODE

(Figure 1)



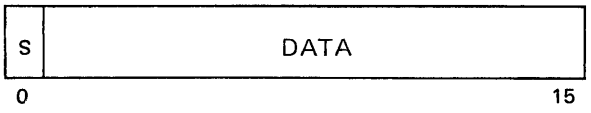
DATA

Numerical data is represented in the SCC 4700 in four ways:

1. Integer
2. Double Precision
3. Floating Point
   a. Short
   b. Long (Double Floating Point)

Integer

The basic data format is a 16-bit binary integer. The sign is located in bit 0, as follows:



S = 0 = positive, 1 = negative

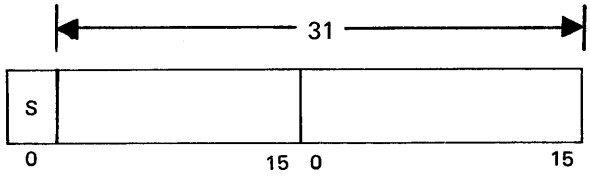
The number represented is defined as a binary or a Hexadecimal integer with bit one being the most significant position and bit fifteen the least significant.

The maximum range of signed integers which may be represented by a single word is $8000_{16} \leq i \leq 7FFF_{16}$ or in decimal, $-32{,}768 \leq i \leq +32{,}767$.

Negative quantities are expressed in two's complement form. The two's complement of a number is obtained by inverting each bit of the binary number and adding one.

Double Precision

Double precision arithmetic utilizes two machine words to represent a 31 bit, signed, binary integer with the following format:



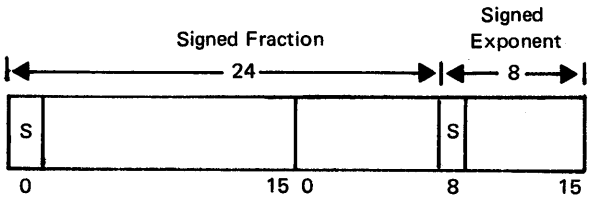
S = 0 = positive, 1 = negative

Numbers up to 2^{31} may be represented in this form. Minimum range is $80000000_{16} \leq i \leq 7FFFFFFF_{16}$ or decimal $2{,}147{,}483{,}648 \leq i \leq +2{,}147{,}483{,}647$. Negative double precision numbers are represented in two's complement form.

FLOATING POINT

Either of two formats may be used for floating point numbers in the 4700. There are two types of optional arithmetic instructions to handle the different floating point formats.

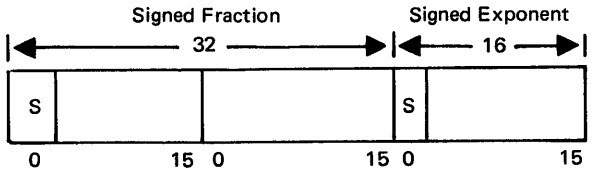
Short Format



S = 0 = positive
1 = negative

The short format uses two words to represent the floating point number. The fraction occupies bits 1-23 and the exponent, 25-31. The radix point of the fraction is assumed to be immediately to the left of the high-order fraction digit.

Long Format (Double Floating Point)



S = 0 = positive
1 = negative

The long format uses three words to represent a floating point number. The fraction occupies bits 1-32 and the exponent, 33-48. The radix point of the fraction is assumed to be immediately to the right of the sign bit (B0).

INSTRUCTION DESCRIPTIONS

The instruction repertoire of the 4700 is separated into standard and optional instruction sets. The optional instructions include four optional sets:

(1) Multiply-Divide
(2) Double Precision
(3) Floating Point
(4) Double Floating Point

The following conventions are used in describing the functions of the instructions:

(1) A register name or address enclosed in parentheses denotes the contents of the register or address.

(2) A location enclosed within two sets of parentheses indicates an indirect address.

(3) Subscription is used to denote bit positions within a register or instruction.

(4) I = Indirect
X = Indexing
S = Sign
R = Relative
Y = Address Field

In this section the function of each instruction is described and the format is given.

The location counter will normally be incremented by one for basic and extended operation code instructions. But, if the primary address of a basic instruction is L+1, the location counter will then be incremented by two, skipping the location containing the indirect address.

Double word instructions will increment the location counter by two to skip the address field unless otherwise noted.

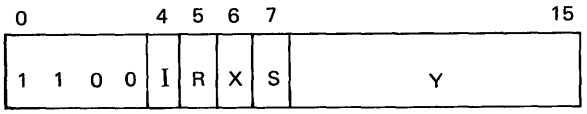
Skip instructions increment the location counter as indicated, depending on the result of the test of the contents of the location or indicator.

Indirect addressing will be allowed only where indicated by "I" in bit position 4.

Pre-indexing is allowed when indicated by an "X" in the instruction format. In double-word instructions, this will be bit 0 of the second word. Post-indexing is available only when indirect addressing is specified. Then, it will be indicated by setting (=1) bit 0 of the indirect address.

Example:

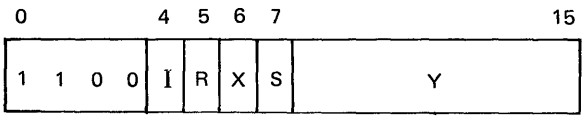
LDA LOAD ACCUMULATOR 1.84 μsec



The contents of the effective address are copied into the A register. The contents of the location is unchanged.

The address mode of the instruction is determined by the setting of the I, R, X bits in the following manner:

LDA LOAD ACCUMULATOR 1.84 μsec

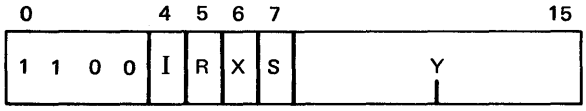


Hexadecimal	Binary		Address Mode
C0XX	1100	0000	Load A Direct
C2XX	1100	0010	Load A Direct Indexed
C4XX	1100	0100	Load A Relative
C6XX	1100	0110	Load A Relative Indexed
C8XX	1100	10000	Load A Indirect from Direct Address
CAXX	1100	1010	Load A Indirect from Indexed Direct Address
CCXX	1100	1100	Load A Indirect from Relative Address
CEXX	1100	1110	Load A Indirect from Relative Indexed Address

The effective address is formed in this way, when the basic instruction format is implemented. Post-indexing is implemented when bit 0 of the indirectly addressed word is set.

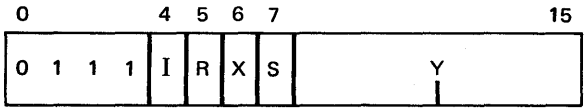
STANDARD INSTRUCTION SET

LDA LOAD ACCUMULATOR 1.84 μsec



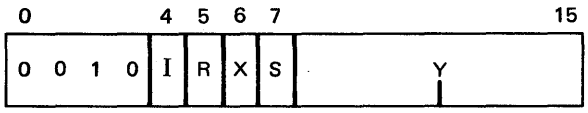
The contents of the effective address are copied into the A Register. The contents of the location are unchanged.

STA STORE ACCUMULATOR 1.84 μsec



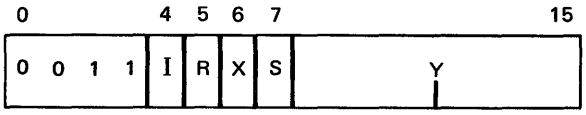
The contents of the A Register are copied into the memory location specified by the effective address. The contents of the A register are unchanged.

LDB LOAD B REGISTER 1.84 μsec



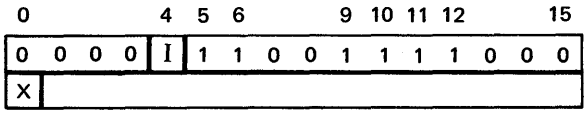
The contents of the effective address are copied into the B register. The contents of the location are unchanged.

STB STORE B REGISTER 1.84 μsec



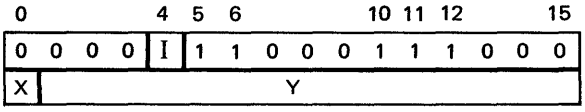
The contents of the B register are copied into the memory location specified by the effective address. The contents of the register are unchanged.

LDX LOAD INDEX 2.76 μsec



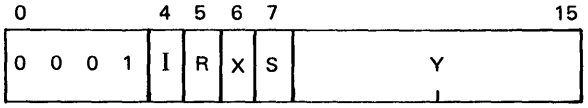
The contents of the effective address are copied into the X Register. The contents of the memory location are unchanged.

STX STORE INDEX 2.76 μsec



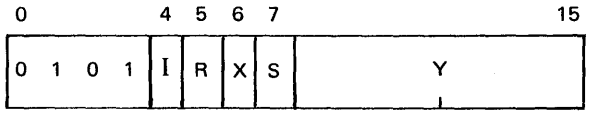
The contents of the X Register are copied into the memory location specified by the effective address. The contents of the register are unchanged.

LDH **LOAD HALFWORD** **1.84 μsec**



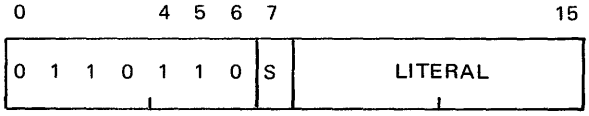
Bits 0-7 of the accumulator are set to zero and the contents of the byte address specified by the effective address are loaded into bits 8-15 of the accumulator. The contents of the effective address are unchanged.

STH **STORE HALFWORD** **2.09 μsec**



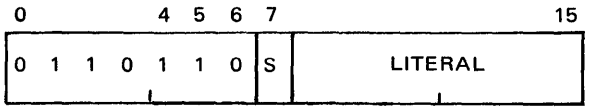
The contents of bits 8-15 of the accumulator are copied into the byte address specified by the effective address. The contents of the accumulator and the other half of the effective address word are unchanged.

LDL **LOAD A, LITERAL** **.92 μsec**



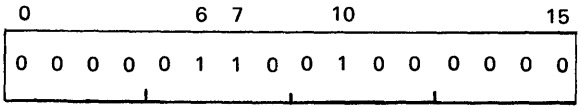
The sign (bit 7) is copied into bits 0-7 of the accumulator. Bits 8-15 of the instruction are copied into bits 8-15 of the accumulator.

LDLB **LOAD B, LITERAL** **.92 μsec**



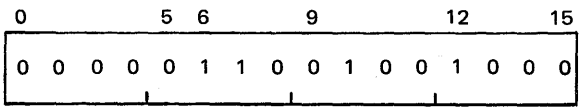
The sign bit (bit 7) is copied and extended into bits 0-7 of the B register. Bits 8-15 of the instruction are copied into bits 8-15 of the B register.

LAS **LOAD ACCUMULATOR FROM SWITCHES** **.95 μsec**



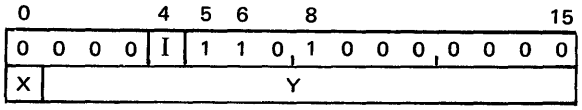
The contents of the switch register are placed into the A register. Data may be entered in the switch register by the operator using the control panel. The contents of the switch register are unchanged.

LDS **LOAD STATUS** **.95 μsec**



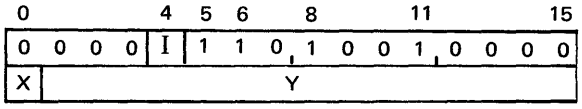
The contents of the status register are copied into the accumulator. The status register is a composite register which includes the carryout, overflow, halt, mode, and interrupts disabled indicators. The contents of the register are not changed.

LDD **LOAD DOUBLE** **3.56 μsec**



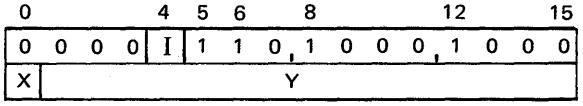
The contents of the effective address are copied into the A register. The contents of the effective address plus one are copied into the B register. The contents of memory are unchanged.

STD **STORE DOUBLE** **3.68 μsec**



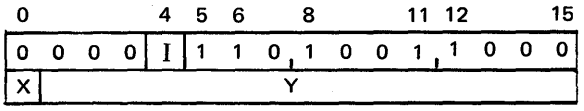
The contents of the A register are copied into the effective address. The contents of the B register are copied into the effective address plus one. The contents of the registers are unchanged.

LDF **LOAD FLOATING** **4.48 μsec**



The contents of the effective address are copied into the A register. The contents of the effective address plus one are placed in the B register. The contents of the effective address plus two are transferred to the E register. The contents of memory are unchanged.

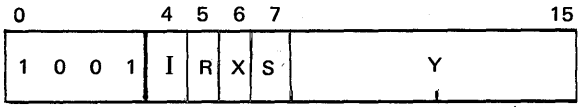
STF **STORE FLOATING** **4.60 μsec**



The contents of the A register are placed into the effective address. The contents of the B register are copied into the effective address plus one. The contents of the E register go into the effective address plus two. The contents of the registers are unchanged.

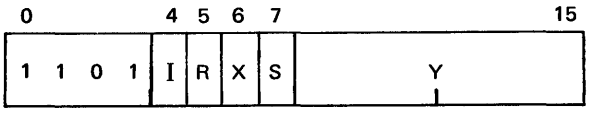
ARITHMETIC INSTRUCTIONS

ADD **ADD TO ACCUMULATOR** **1.84 μsec**



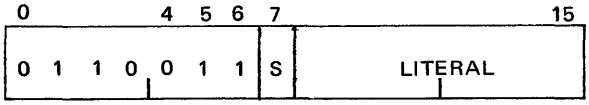
Then the contents of the effective address (16 bit operand) are added to the current value of the accumulator and the result is placed in the accumulator. The CRO is set by the carry from bit position 0. If the result is greater than the maximum size of the register, the overflow indicator is set. If overflow does not occur the overflow indicator is not altered.

SUB **SUBTRACT FROM ACCUMULATOR** **1.84 μsec**



The contents of the effective address (16 bit operand) are subtracted from the current value of the accumulator and the result is placed in the accumulator. The CRO is set by the carry from bit position 0. If the result is greater than the maximum size of the register, the overflow indicator is set. If overflow does not occur, the overflow indicator is not altered. If both numbers have the same sign, but the sign of the result is different, an overflow has occurred. The location counter is incremented by one. (By two, if the primary address is L+1.)

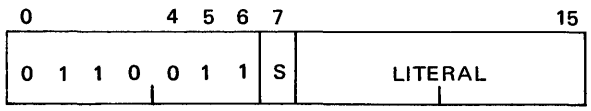
ADL **ADD TO A, LITERAL** **1.10 μsec**



This instruction provides a convenient way to add or subtract quantities in the range $-256 \leq X \leq +255$. The last nine bits (7-15) of this instruction are interpreted as a nine bit, two's complement number.

Bit 7 of the instruction is extended through bits 0-6, and the nine bit operand is converted to a 16 bit, two's complement operand and is added to the contents of the accumulator. The carryout indicator is set by the carryout of bit position zero. If overflow does not occur, the overflow indicator is not altered.

ADLB **ADD TO B, LITERAL** **1.10 μsec**

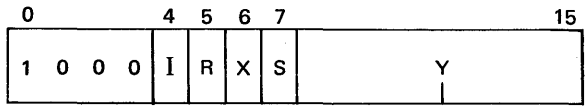


This instruction provides a convenient way to add or subtract quantities in the range $-256 \leq X \leq +255$. The last nine bits (7-15) of this instruction are interpreted as a nine bit, two's complement number.

Bit 7 of the instruction is extended through 0-6, and the nine bit operand is connected to a 16 bit, two's complement operand, which is added to the contents of the B register. The carryout indicator is set by the carryout of bit position zero. If overflow does not occur, the overflow indicator is not altered.

The location counter is incremented by one. Indirect addressing and indexing are not allowed.

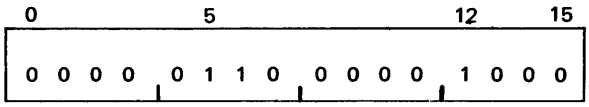
MIN **MEMORY INCREMENT; SKIP ON ZERO** **2.14 μsec**



One is added to the contents of the memory location at the effective address. The result is tested, and if the value equals zero, the location counter is incremented by two. (By three, if the primary address is L+1.) If the value is not zero, the location counter is incremented by one. The carryout and overflow indicators are not affected.

If the primary address is L + 1, the location counter will be incremented by two ($Q \neq 0$); or by three ($Q = 0$).

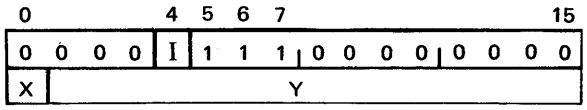
ADC **ADD CARRY** **1.20 μsec**



The carryout indicator is added to the accumulator and the result is placed in the accumulator.

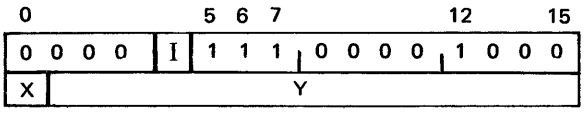
If the results exceed the size of the accumulator, the OVF indicator is set. The CRO indicator is set by the carryout of bit zero. If overflow does not occur, the OVF indicator is not altered. Indirect addressing and indexing are not allowed.

MPY **MULTIPLY** **6.84 μsec (min)** **8.44 μsec (max)**



This operation is a single-precision integer multiplication, which leaves the most significant half of the 32-bit product in the B register and the least significant half in the A register. The multiplicand is placed in A, before the instruction is executed. The multiplier is the contents of the effective address of the double word multiply instruction. The multiplier and multiplicand must be right-adjusted. The product will be right adjusted in the B and A registers after execution. Overflow is not affected; but the carryout indicator may be set if carryout occurs on the last operation performed. The contents of the effective address remain unchanged.

DIV **DIVIDE** **8.14 μsec (min)** **8.99 μsec (max)**

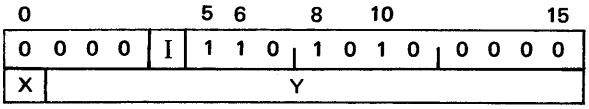


This is a single-precision integer divide, with the dividend right-adjusted in the B and A registers. (The most significant part in the B register.)

The contents of the B register plus A_0 are shifted one place to the left and compared to the contents of the effective address. If equal to or greater than the contents of the effective address, the overflow indicator is set; and the divide is terminated.

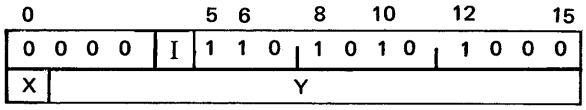
If the division is completed, the quotient will be placed in A and the remainder in B. The sign of the quotient is determined by the rules of division and the sign of the remainder will be the sign of the dividend.

DAD **DOUBLE ADD** **3.91 μsec**



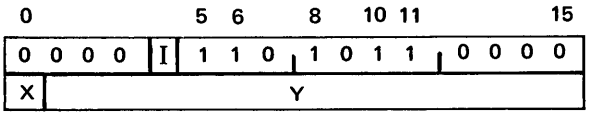
This is a 32-bit double-precision add, where the contents of the effective address (Q) and (Q) + 1 are added to the contents of the A and B register. The sum is placed in the A and B registers, with the most significant part in A. If the signs of the two operands are equal, but the sign of the result is different, an overflow has occurred and the overflow indicator is set. The carryout indicator is set if a carry occurs from the sign bit of the adder.

DSB **DOUBLE SUBTRACT** **3.91 μsec**



DSB is a 32-bit double-precision subtract, where the contents of the effective address (Q) and (Q) + 1 are subtracted from the A and B registers. (The instruction takes the one's complement of the quantity to be subtracted and adds it to the A and B registers with a forced carry-in. The result is placed in the A and B registers with the most significant part in the A register. If the signs are different and the sign of A changes, then overflow is set. The carryout indicator set if a carry occurs from the high order position of the adder.

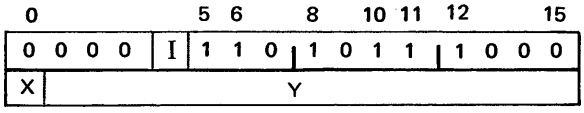
DMP **DOUBLE MULTIPLY** **25.30 μsec**



This instruction provides a fixed point multiplication of 32 binary bits times 32 binary bits. It is a fractional multiply which will yield a 32-bit product. Both the overflow and carryout may be set. The result of 80000000_{16} x 80000000_{16} will cause overflow.

The multiplier is contained in the effective address (Q) and (Q) + 1. The product will be placed in the A and B registers, with the most significant half in A.

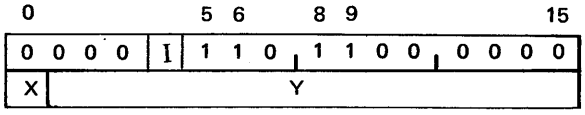
DDV **DOUBLE DIVIDE** **36.39 μsec (min)**
48.39 μsec (max)



This is a fixed point, fractional divide of 32 bits into 64 binary bits. Registers A and B comprise the high order 32 bit positions of the dividend. The low order bits are assumed to be zero. The effective address (Q) and (Q)+1 contains the divisor. The quotient will be placed in A and B with the most significant part in A.

The overflow is set if $|A,B| > |Q,Q+1|$. If overflow occurs, the contents of the A and B register cannot be predicted.

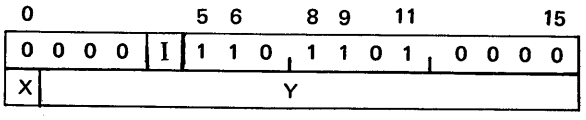
FAD **FLOATING POINT ADD** **9.11 μsec (min)**
12.01 μsec (max)



This instruction is to be used with the short (or two word) floating point format. The contents of the effective address (Q) and (Q) + 1 are added to the A and B registers after alignment of the binary point. (Alignment means that the exponents of both numbers are set equal.)

If exponent overflow or underflow occurs, the system trap indicator will be set.

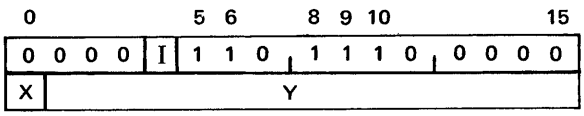
FSB **FLOATING POINT SUBTRACT** **9.81 μsec (min)**
13.01 μsec (max)



This instruction is to be used with the short (or two word) floating point format. The contents of the effective address (Q) and (Q) + 1 are subtracted from the A and B registers after alignment of the binary point. (Alignment means that the exponents of both numbers are set equal.)

If exponent overflow or underflow occurs, the system trap indicator will be set.

FMP **FLOATING POINT MULTIPLY** **31.99 μsec (min)**
33.49 μsec (max)

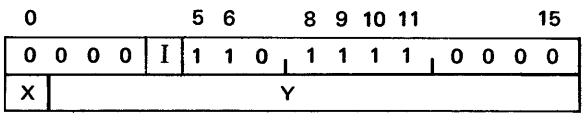


This instruction is to be used with the short (or two word) floating point format. The contents of the effective address are the multiplier; the contents of the A and B registers are the multiplicand. The product is placed in the A and B registers.

Both numbers must be normalized ($A_0 \neq A_1$) before multiplication; the answer will be normalized when returned.

If exponent overflow or underflow occurs, the system trap indicator will be set.

FDV FLOATING POINT DIVIDE 40.74 μsec (min) 47.99 μsec (max)

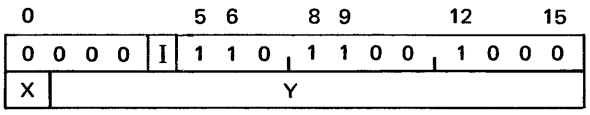


This instruction is to be used with the short (or two word) floating point format. The contents of the effective address (Q) and (Q) + 1 is the divisor and the contents of the A and B registers are the dividend. The quotient is placed in the A and B registers.

Both numbers must be normalized ($A_0 \neq A_1$) before division; the answer will be normalized when returned. Rounding will be done on the 25th bit of the answer, before combining with the exponent.

If exponent overflow or underflow occurs, the system trap indicator will be set. The carryout is not significant.

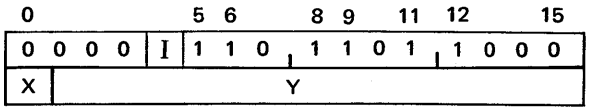
DFA DOUBLE FLOATING ADD 7.83 μsec (min) 8.98 μsec (max) ‡



This instruction is to be used with the long (or three word) floating point format. The floating point quantity at the effective address (Q), (Q) + 1, and (Q) + 2 is added to the floating point quantity in the A,B, and E registers after alignment of the binary point. (Alignment means that the exponents of both numbers are set equal.)

If exponent overflow or underflow occurs, the system trap indicator will be set.

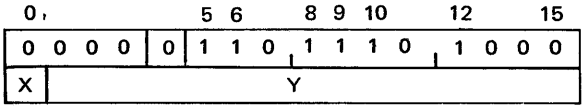
DFS DOUBLE FLOATING SUBTRACT 7.83 μsec (min) 8.98 μsec (max) ‡



This instruction is to be used with the long (or three word) floating point format. The floating point quantity at the effective address (Q), (Q) + 1, and (Q) + 2 is subtracted from the floating point quantity in the A,B, and E registers after alignment of the binary point. (Alignment means that the exponents of both numbers are set equal.)

If exponent overflow or underflow occurs, the system trap indicator will be set.

DFM DOUBLE FLOATING MULTIPLY 29.91 μsec (min) 31.26 μsec (max)



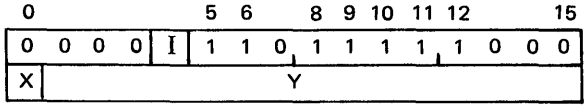
This instruction is to be used with the long (or three word) floating point format. The floating point quantity at the effective address (Q), (Q) + 1 and (Q) + 2 is multiplied times the floating point quantity in the A,B, and E registers. The product replaces the multiplicand in the A, B, and E registers.

Both numbers must be normalized ($A_0 \neq A_1$) before multiplication; the normalized answer will be returned.

If exponent overflow or underflow occurs, the system trap indicator will be set.

‡ +.21N for alignment

DFD **DOUBLE FLOATING DIVIDE** **38.21 μsec (min)**
43.11 μsec (max)



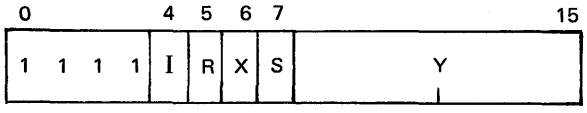
This instruction is to be used with the long (or three word) floating point format. The floating point quantity in the A,B, and E registers is divided by the floating point quantity at the effective address (Q), (Q) + 1, and (Q) + 2. The quotient replaces the dividend in the A,B, and E registers.

Both numbers must be normalized ($A_0 \neq A_1$) before division; the answer will be normalized when returned.

If exponent overflow or underflow occurs, the system trap indicator will be set.

LOGICAL INSTRUCTIONS

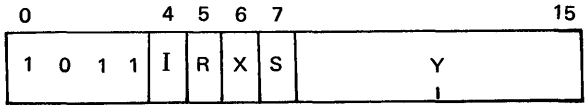
AND **AND MEMORY WITH ACCUMULATOR** **1.84 μsec**



The content of the effective address is AND'd with the contents of the accumulator. The contents of the memory location remains unchanged. The location counter is incremented by one. (By two, if the primary address is L+1.) Logical AND is defined as:

		Q	
		0	1
A	0	0	0
	1	0	1

XOR **EXCLUSIVE OR WITH ACCUMULATOR** **1.84 μsec**

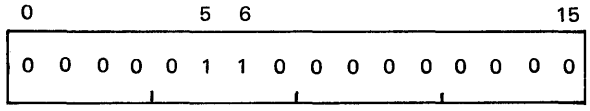


The contents of the effective address are exclusive OR'd with the contents of the accumulator. The result is placed in the accumulator. The memory location is unchanged. The following table defines the exclusive OR operation:

		Q	
		0	1
A	0	0	1
	1	1	0

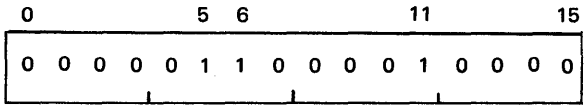
The location counter is incremented by one. (By two, if the primary address is L+1.)

AAB **AND ACCUMULATOR WITH B REGISTER** **1.10 μsec**



The contents of the accumulator is AND'd with the contents of the B register. The result is placed in the accumulator. The contents of the B Register are unchanged, and the location counter is incremented by one.

AOB **OR ACCUMULATOR WITH B REGISTER** **1.10 μsec**

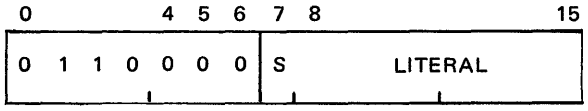


The contents of the accumulator are OR'ed with the contents of the B Register. The result is placed in the accumulator. The content of the B Register is unchanged, and the location counter is incremented by one. Indirect addressing and indexing are not allowed.

The following table defines the logical OR operation:

		X	
		0	1
A	0	0	1
	1	1	1

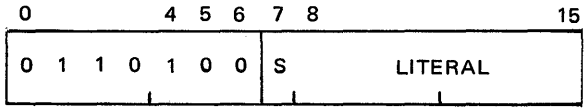
ANL **AND THE ACCUMULATOR LITERAL** **1.10 μsec**



Bits 7-15 of this instruction are interpreted as a nine bit, two's complement number. Bit 7 of the instruction is extended through bits 0-6, and the 16 bit operand is AND'd with the contents of the accumulator. The result is placed in the accumulator.

The AND operation is defined in the description of the AND instruction. The location counter is incremented by one. Indirect addressing and indexing are not allowed.

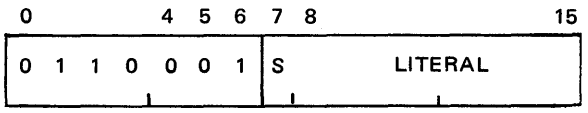
ANLB **AND THE B REGISTER LITERAL** **1.10 μsec**



Bits 7-15 of this instruction are interpreted as a nine bit, two's complement number. Bit 7 of the instruction is extended through bits 0-6, and the 16 bit operand is AND'd with the contents of the B register. The result is placed in the B register.

The AND operation is defined in the description of the AND instruction. The location counter is incremented by one. Indirect addressing and indexing are not allowed.

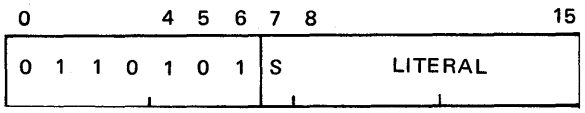
XOL **EXCLUSIVE OR THE ACCUMULATOR** **1.10 μsec**



Bits 7-15 of this instruction are interpreted as a nine bit, two's complement number. Bit 7 of the instruction is extended through bits 0-6, and the 16 bit operand is exclusive OR'd with the contents of the accumulator. The result is placed in the accumulator.

The exclusive OR operation is defined in the description of the XOR instruction. The location counter is incremented by one. Indirect addressing and indexing are not allowed.

XOLB **EXCLUSIVE OR THE B REGISTER, LITERAL** **1.10 μsec**

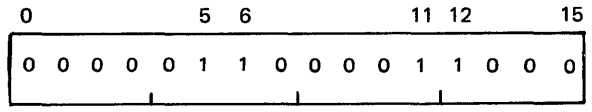


Bits 7-15 of this instruction are interpreted as a nine bit, sign, extending, two's complement number. Bit 7 of the instruction is extended through bits 0-6, and the 16 bit operand is exclusive OR'd with the contents of the B register. The result is placed in the B register.

The exclusive OR operation is defined in the description of the XOR instruction. The location counter is incremented by one. Indirect addressing and indexing are not allowed.

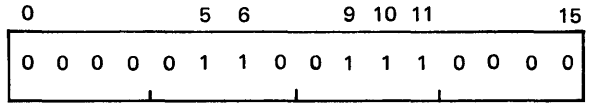
REGISTER MANIPULATION

XAX EXCHANGE ACCUMULATOR AND INDEX 1.10 μsec



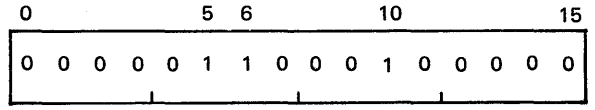
The contents of the accumulator are switched into the index and the contents of the index are placed in the accumulator in a single operation. The location counter is incremented by one. Indirect addressing and indexing are not allowed.

XBX EXCHANGE B AND X 1.10 μsec



The contents of the B Register are placed in the index and the contents of the index are placed in the B Register in one operation. The location counter is incremented by one. Indirect addressing and indexing are not allowed.

ESA EXTEND SIGN OF ACCUMULATOR 1.00 μsec



The sign (bit 0) of the accumulator is extended through the B Register ($B_{0\text{-}15}$). The accumulator is unchanged. The location counter is incremented by one. Indirect addressing and indexing are not allowed.

OPERATE GROUP

This is a special group of instructions which perform inter-register manipulations. With them, arithmetic and logical functions can be performed on a register, or between two registers, asynchronously of memory. Testing of the contents of the destination register takes place after the function is complete. The test, if specified, does not require any additional time.

FUNCTIONS

BITS 7-9	DESCRIPTION	FUNCTION	MICRO-SECONDS
000	Copy Contents	S → D	1.05
001	Increment Register	S + 1 → D	1.10
010	Add Source to Destination	S + D → D	1.40
011	Exclusive OR	S⊕D → D	1.40
100	One's Complement	$\overline{S}$ → D	1.25
101	Two's Complement	$\overline{S}$ + 1 → D	1.25
110	Decrement Register	S − 1 → D	1.25
111	Subtract Destination from Source	S − D → D	1.40

TESTS

BITS 10 & 11	
00	No Skip
01	Skip if positive (> 0)
10	Skip if negative
11	Skip if Zero

REGISTERS (Source or Destination)

CODE	
00	A Register
01	B Register
10	X Register
11	E Register (Optional)*

The assembler will implement the following mnemonics for operate instructions

RCPY – Register Copy
RINC – Register Increment
RADD – Register ADD
RXOR – Register Exclusive OR
RCMP – Register Complement
RNEG – Register Negate
RDEC – Register Decrement
RSUB – Register Subtract

The form of the operate instruction in the assembler is as follows:

OP S,D,T

where OP is one of the eight mnemonics above, S which represents the source field is A,B,X or E and D which represents the destination register is A,B,X or E. T can be G,N or Z or can be null. Testing is performed on the destination register after the indicated function has been performed.

Examples:

RINC	A,A	Increment A
RXOR	A,B	$A \oplus B \rightarrow B$
RINC	A,B	$A + 1 \rightarrow B$
RCPY	A,X,Z	$A \rightarrow X$ Skip if $X = 0$

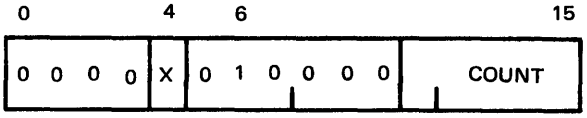
* This option is available only when floating point hardware has been implemented.

SHIFT INSTRUCTIONS

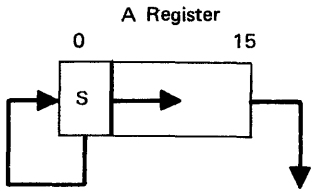
Shifts may be indexed by setting bit 4 (=1) of the instruction. Indexing will not be allowed to change the type of shift. However, it may modify the direction, length, and count. For logical shifts, if bit 11 of the instruction is set (count >16) the direction may not be changed. If it is, the result of the shift cannot be predicted.

For logical and rotate instructions, bit 11 cannot be changed by indexing. The instruction will execute the original setting regardless of indexing.

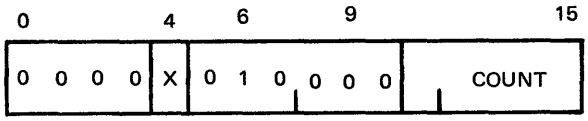
SAR **SHORT ARITHMETIC RIGHT SHIFT** **1.38 + .23N M** **1.38 + .23N μsec**



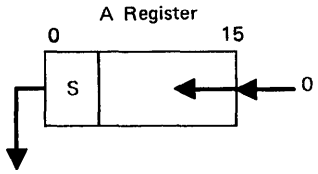
The number of binary positions which the instruction will cause the data to be shifted is determined by the number placed in the "count" field. ($0 < i < 32$). The contents of the accumulator will be shifted to the right, the desired number of binary places. The data being shifted out of A_{15} is lost. The sign bit is propagated from one position to the next, the length of the shift. The location counter is incremented by one after the required number of shifts have been performed. The index register will be added to the instruction, if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$.



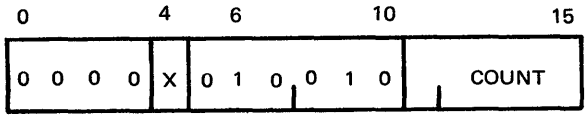
SAL **SHORT ARITHMETIC LEFT SHIFT** **1.38 + .23N μsec**



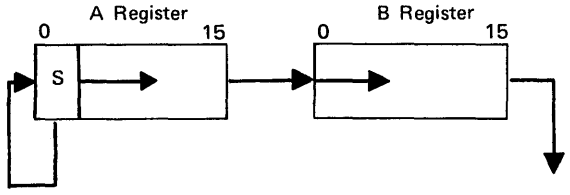
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator will be shifted to the left, the desired number of binary places. Zeros are inserted into A_{15}, and data is lost from A_0. However, the overflow indicator is set if the sign changes. The location counter is incremented by one after completion of the command. The index register will be added to the instruction if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$.



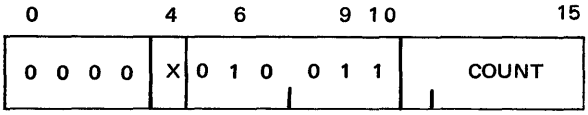
LAR **LONG ARITHMETIC** **1.38 + .23N μsec**



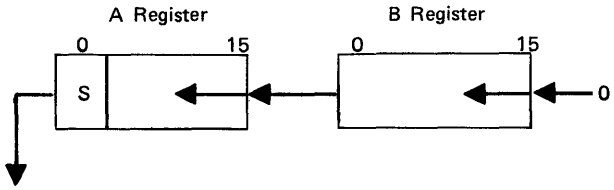
The long shift links both the A and B register and shifts them as one 32 bit register. The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of both registers will be shifted to the right, the desired number of places. The sign bit is propagated from A_0 to A_1 and data is lost from B_{15}. The location counter is incremented by one. The index register will be added to the instruction if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$.



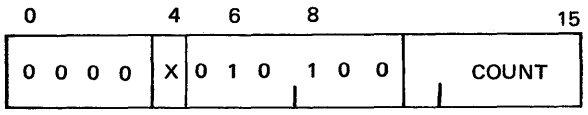
LAL **LONG ARITHMETIC LEFT SHIFT** **1.38 + .23N μsec**



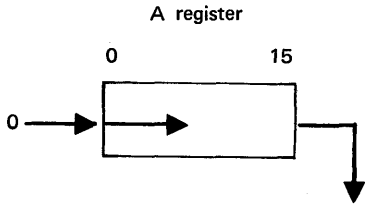
The long shift links both the A and B register and shifts them as one 32 bit register. The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the A and B registers will be shifted to the left, the desired number of binary places. Zeros are inserted into B_{15}, and data is lost from A_0. The overflow indicator is set if the sign changes. The location counter is incremented by one. The index register will be added to the instruction, if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$.



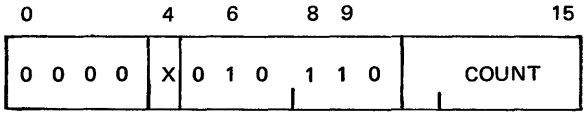
SLR **SHORT LOGICAL RIGHT SHIFT** **1.38 + .23N μsec**



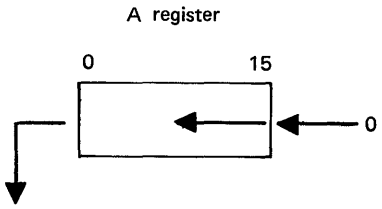
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator will be shifted to the right. The desired number of places. Zeros are inserted into A_0 and data is lost from A_{15}. The location counter is incremented by one. The index register will be added to the instruction if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$.



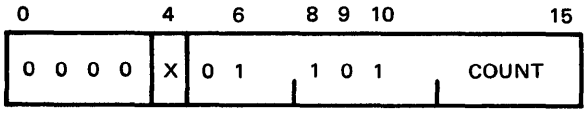
SLL **SHORT LOGICAL LEFT SHIFT** **1.38 + .23N μsec**



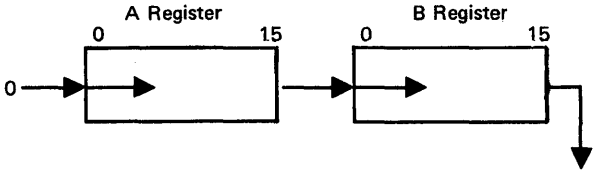
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator will be shifted to the left, the desired number of places. Zeros are inserted into A_{15} and data is lost from A_0. The location counter is incremented by one when complete. The index register will be added to the instruction, if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$.



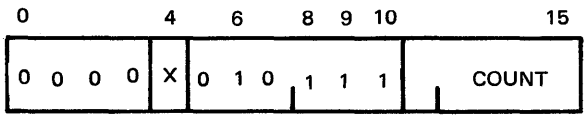
LLR **LONG LOGICAL RIGHT SHIFT** **.23N - 2.07 μsec**



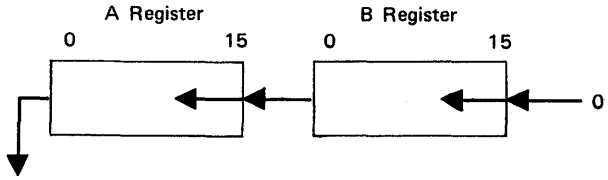
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator and B Register will be shifted to the right, the desired number of places. Zeros are inserted into A_0 and data is lost from B_{15}. The location counter is incremented by one when complete. The index register will be added to the instruction if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$.



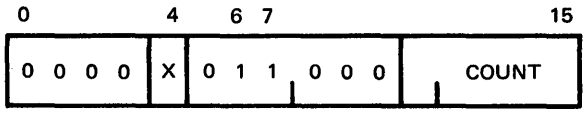
LLL **LONG LOGICAL LEFT SHIFT** **1.38 + .23N μsec**



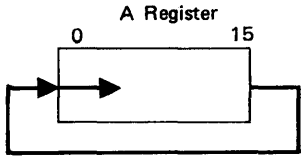
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator and B Register will be shifted to the left, the desired number of places. Zeros are inserted into B_{15} and data is lost from A_0. The location counter is incremented by one when complete. The index register will be added to the instruction if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$.



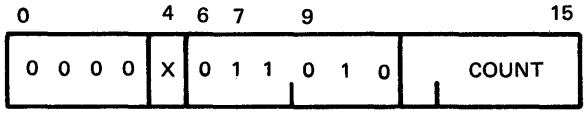
SRR **SHORT ROTATE RIGHT** **1.38 + .23N μsec**



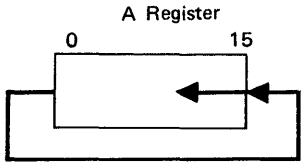
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator will be shifted to the right, the desired number of places. No data is lost. The content of A_{15} is inserted in A_0 and the shift continues in a circular manner. The location counter is incremented when the shift is completed. The index register will be added to the instruction, if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$.



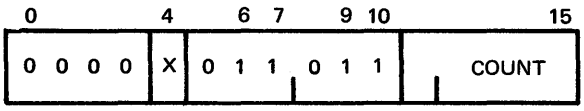
SRL **SHORT ROTATE LEFT** **1.38 + .23N μsec**



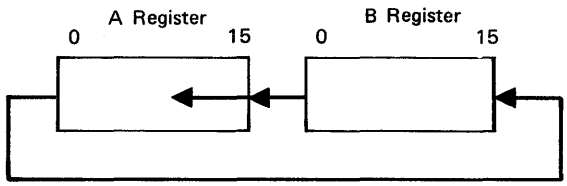
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator will be shifted to the left, the desired number of places. No data is lost. The content of A_0 is inserted into A_{15} and the shift continues in a circular manner. The location counter is incremented by one when the shift is completed. The index register will be added to the instruction, if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$



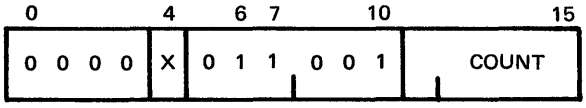
LRL LONG ROTATE LEFT .23N - 2.07 μsec



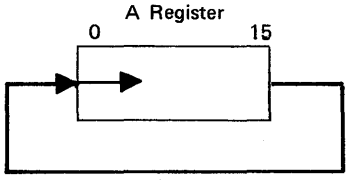
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator and B Register will be shifted to the left, until i = 0. No data is lost. The content of A_0 is inserted into B_{15} and B_0 is transferred to A_{15}; the shift continues in a circular manner. The location counter is incremented by one when the shift is completed. The index register will be added to the instruction, if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$.



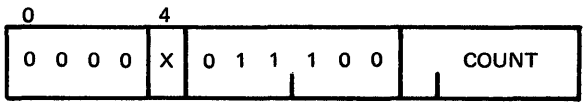
LRR LONG ROTATE RIGHT .23N - 2.07 μsec



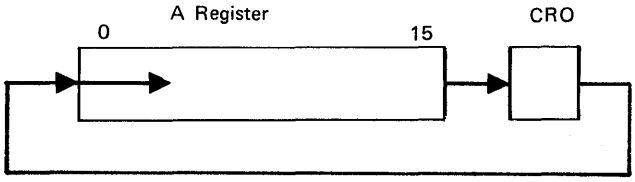
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator and B Register will be shifted to the right, until i = 0. No data is lost. The contents of B_{15} is inserted into A_0 and A_{15} is transferred to B_0; the shift continues in a circular manner. The location counter is incremented by one when the shift is completed. The index register will be added to the instruction, if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$. However, $(X)_8$ must be zero, and the addition of the index register must not cause a carry from INS_9.



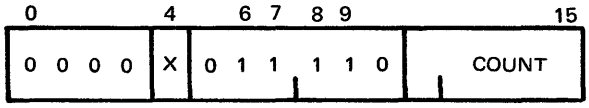
SCR SHORT CIRCULATE RIGHT 1.38 + .23N μsec



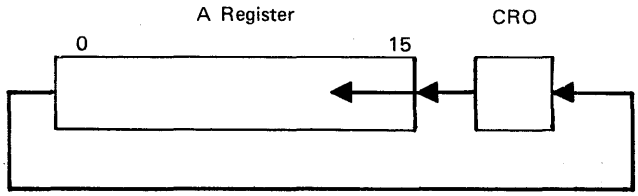
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator will be shifted to the right, until i = 0. No data is lost, the content of A_{15} is inserted in the carryout indicator (CRO) and the carryout indicator is transferred to A_0 until i = 0; while the shift continues in a circular manner. The location counter is incremented by one when the shift is completed. The index register will be added to the instruction, if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$. However, $(X)_8$ must be zero, and the addition of the index register must not cause a carry from INS_9.



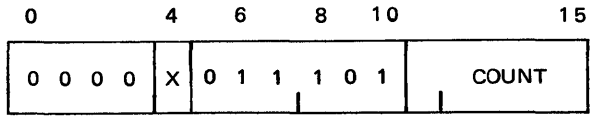
SCL SHORT CIRCULATE LEFT 1.38 + .23N μsec



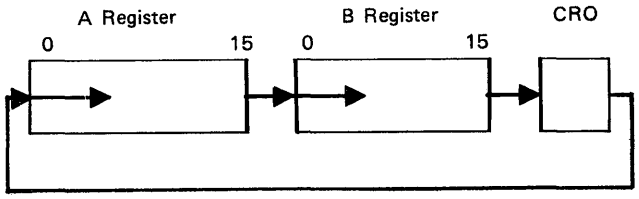
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator will be shifted to the left, until i = 0. No data is lost. The content of the carryout indicator (CRO) is inserted into A_{15} and A_0 is transferred to the carryout, until i = 0; while the shift continues in a circular manner. The location counter is incremented by one when the shift is completed. The index register will be added to the instruction, if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$. However, $(X)_8$ must be zero, and the addition of the index register must not cause a carry from INS_9.



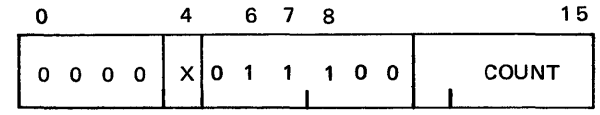
LCR LONG CIRCULATE RIGHT 1.38 + .23N μsec



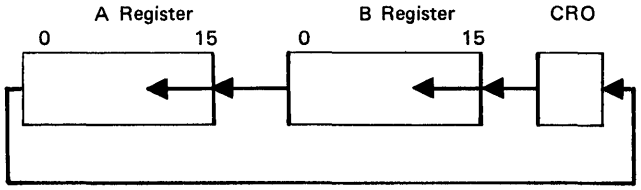
The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator and B register will be shifted to the right, until i = 0. No data is lost. The content of the carryout indicator (CRO) is inserted into A_0; A_{15} is transferred into B_0; and B_{15} is moved to the carryout indicator, until i = 0. The shift continues in a circular manner until complete; then the location counter is incremented by one. The index register will be added to the instruction, if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15} + (X)_{0\text{-}15}$. However, $(X)_8$ must be zero, and the addition of the index register must not cause a carry from INS_9.



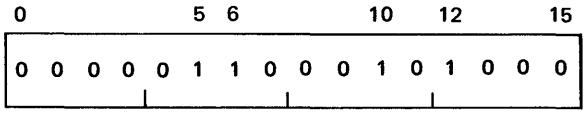
LCL LONG CIRCULATE LEFT 1.38 + .23N μsec



The number of binary positions to be shifted is determined by the number placed in the "count" field ($0 < i < 32$). The contents of the accumulator and B register will be shifted to the left, until i = 0. No data is lost. The content of the carryout indicator (CRO) is inserted into B_{15}; B_0 is transferred into A_{15}; and A_0 is brougnt into the carryout indicator, until i = 0. The shift continues in a circular manner until complete; then the location counter is incremented by one. The index register will be added to the instruction, if bit 4 is equal to one. This will result in $(INS)_{0\text{-}15}+(X)_{0\text{-}15}$. However, $(X)_8$ must be zero, and the addition of the index register must not cause a carry from INS_9.



LLO LOCATE LEADING ONE 1.05 μsec



If the A register is equal to zero, the location counter is incremented by one and the next sequential instruction is executed.

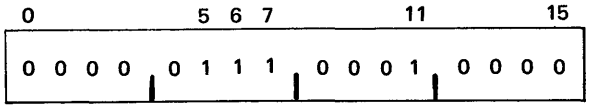
If A is not equal to zero, then the contents of A are shifted to the left one binary position. Then, the X register is incremented by one and $(A)_0$ is tested. If it is not equal to one, the short logical left shift continues, and the X register is incremented by one for each bit position shifted. When $(A)_0$ is found equal to one, the location counter is incremented by two and A_0 is reset (made equal to zero).

If A = 0	1.15 μsec
If A ≠ 0	1.84 + .23N μsec

(N = bit position which is equal to one)

One use for this instruction is to determine what interrupt has occurred in interrupt or device service routines. It will quickly indicate which bit in the A register was set (=1). The number (N) being already in the index register, a jump can quickly be made to the proper servicing routine.

NDX NORMALIZE AND DECREMENT INDEX 1.50 μsec



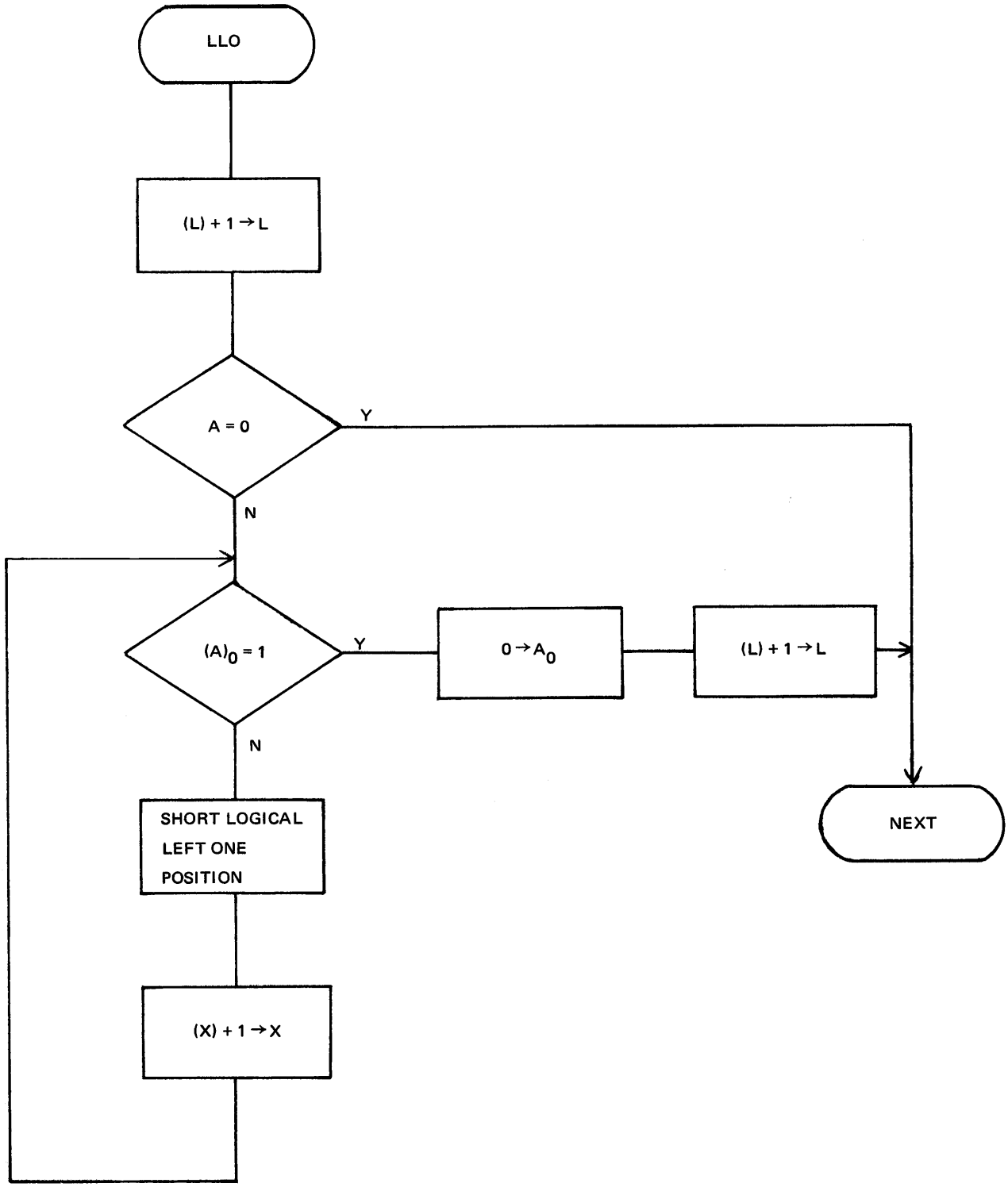
If A and B are both zero, the index register is reset to zero and the next sequential instruction is executed.

If either the A or B register are not equal to zero, then the two registers are linked and their contents are shifted left until $(A)_0 \neq (A)_1$. For each bit position shifted, the index register is decremented by one.

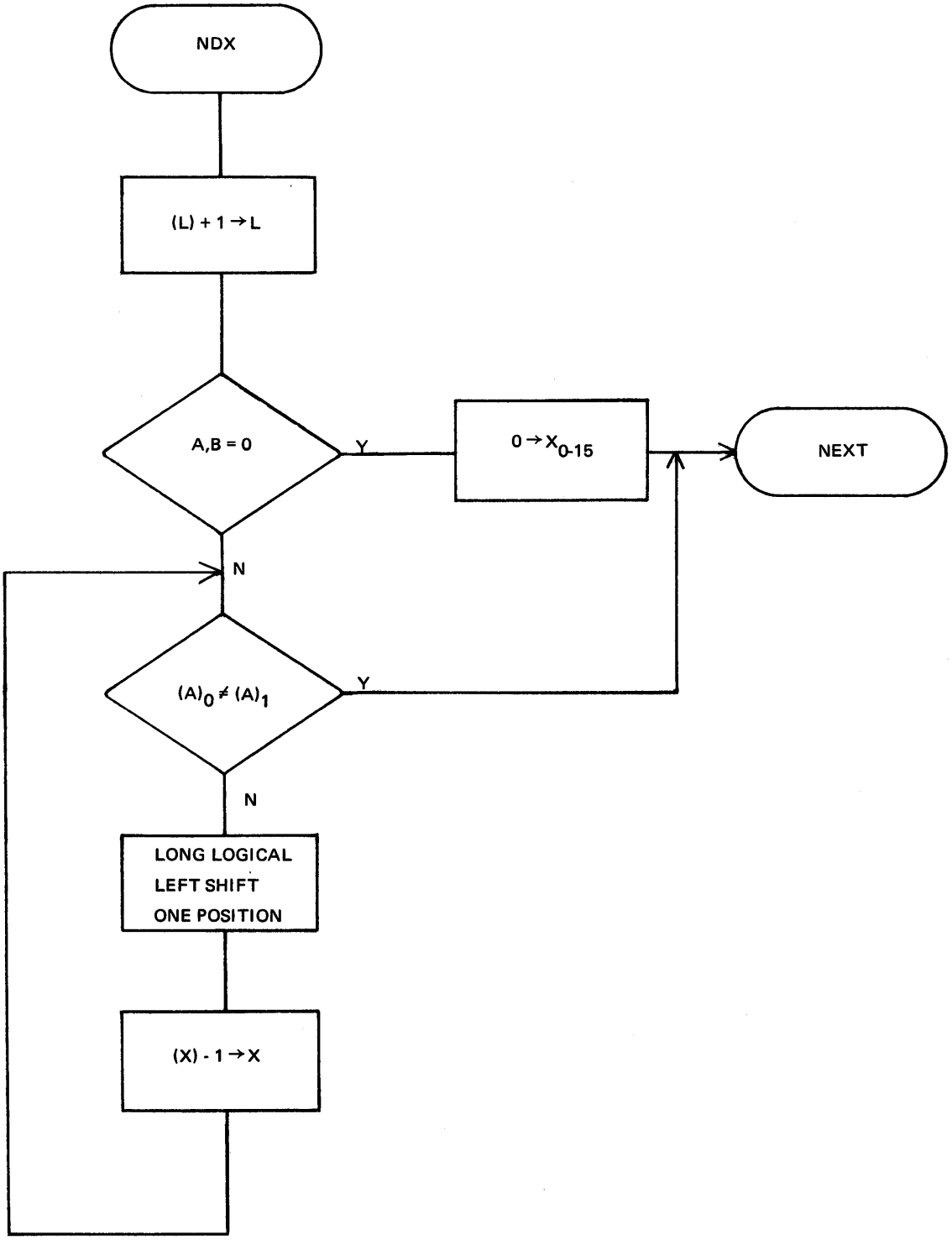
The normalize instruction is used in arithmetic subroutines to give greater precision for quantities represented in floating point notation. Usually, the exponent is placed in the X register and the normalize instruction is then executed.

The time of execution is:

		B	
		= 0	≠ 0
A	= 0	1.38	1.84 + .46N
	≠ 0	1.61	2.07 + .46N



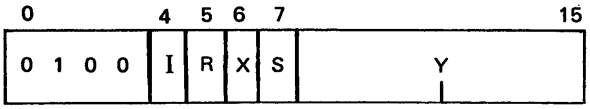
LOCATE LEADING ONES



NORMALIZE AND DECREMENT INDEX

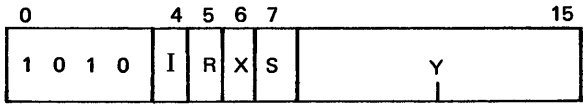
JUMP & SKIP INSTRUCTIONS

JMP **JUMP** **.92 μsec**



After all modifications indicated by the address control bits, the effective address is placed in the location counter. Control is then transferred to the instruction sequence at that location. See section on addressing modes for a detailed description of how the effective address is calculated.

JSL **JUMP AND STORE LOCATION** **2.76 μsec**



The contents of the location counter plus one is stored at the address specified by the contents of the effective address. Then, the effective address value plus one is placed in the location counter, and control is transferred to that location.

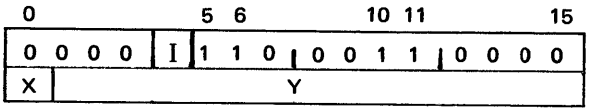
Example: (addresses in decimal)

```
500     JSL     SUB
501     LDA     TAX
 .       .       .
 .       .       .
 .       .       .
600     SUB     PAR     SAVE
601             RINC    A,A,Z
```

Location Counter: 500 — before; 601 — after

SAVE: 501

JRT **JUMP RETURN** **2.76 μsec**



The contents of the effective address are copied into the location counter and control transferred to that location.

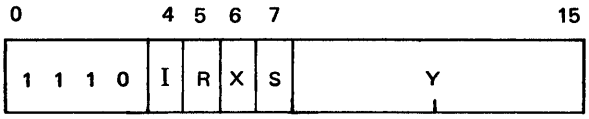
Example: (addresses in decimal)

```
600     SUB     PAR     SAVE
601             RINC    A,A,Z
 .               .
 .               .
 .               .
610             JRT
                PAR     SAVE
```

Location Counter: 610 — Before; 501 — After

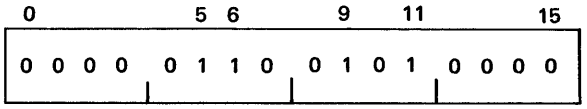
SAVE: 501

SKN **SKIP IF A ≠ MEMORY** **1.84 μsec**



The contents of the A register are compared to the contents of the effective address. If they are equal, the location counter is incremented by one. If they are not equal, the location counter is incremented by two.

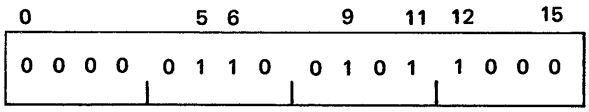
COT **CARRYOUT TEST** **1.15 μsec**



The carry out indicator is tested. If it is set (=1), the location counter is incremented by one and the CRO is reset.

If the CRO is reset (=0), when tested, the location counter is incremented by two.

OFT **OVERFLOW TEST** **1.15 μsec**

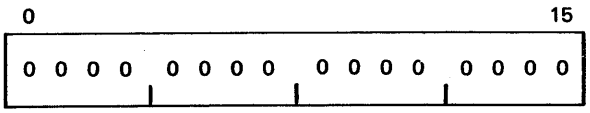


The overflow indicator is tested. If it is set (=1), the location counter is incremented by one and the OVF is reset.

If the OVF is reset (=0) when tested, the location counter is incremented by two.

CONTROL INSTRUCTIONS

HLT **HALT**

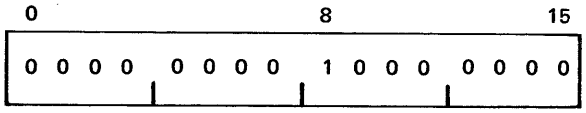


The machine decodes the halt instruction and switches from the run to halt mode. To return it to the run mode, the operator presses the run switch on the console. The 4700 can also switch to the run mode as a result of an interrupt. All of the following interrupts will cause it to be placed in the run mode:

1. Power Off
2. Power On
3. Channel Interrupt
4. Console Interrupt
5. Any external interrupt

Halt is a privileged instruction when memory mapping is implemented.

ENA **ENABLE INTERRUPTS** **.92 μsec**

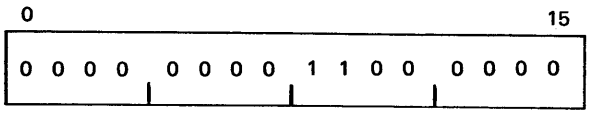


This instruction activates the interrupt system.

Until it is executed, no interrupt will be honored. However, any armed interrupts which occur will be "remembered" and serviced in priority sequence, when the system is enabled.

Once the interrupt system has been activated, it can be deactivated only by the Disable Interrupts instruction or by pressing the system reset button on the console which will disable the interrupt system and disarm the interrupts.

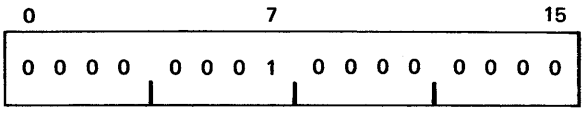
DIS **DISABLE INTERRUPTS** **.92μsec**



This instruction deactivates the interrupt system.

After it is executed, interrupts will not be serviced, even if armed. It is a privileged instruction if memory mapping is implemented.

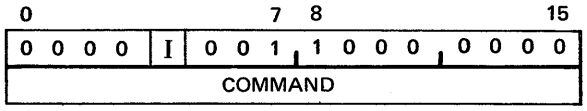
CLI **CLEAR INTERRUPT** **.92 μsec**



This instruction clears the active interrupt which has highest priority. Since only one interrupt can be serviced at a time, all lower priority interrupts requesting service stay in the waiting state, until this instruction is executed; then, the next highest will become active.

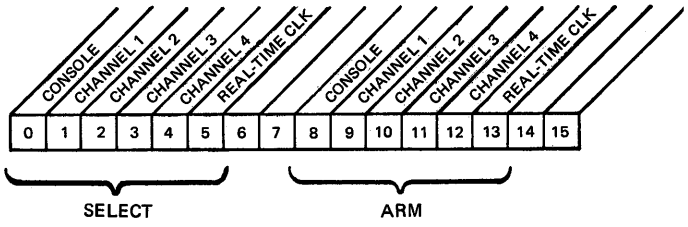
This instruction resets the active and waiting flip-flops of interrupt being serviced. See priority interrupt system. CLI is a privileged instruction if memory mapping is implemented. Interrupt requests will not be allowed to go active until completion of the instruction following a CLI instruction.

ARM **ARM INTERRUPTS** **1.84 μsec**



The arm interrupts instruction controls the individual arming of the programmable internal interrupts. Control information is transmitted to the internal interrupt system from the following location by execution of the ARM instruction. The command information will be located in the address contained in the second word if bit 4 is set. It is a privileged instruction when memory mapping is implemented.

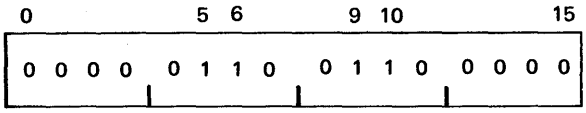
The format of the command to be loaded into the A register before executing the instruction is:



The SELECT and ARM/DISARM bits for a given channel are interpreted as follows:

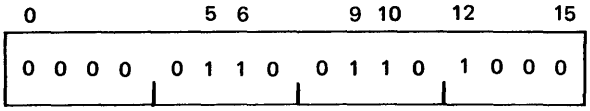
SELECT	ARM/DISARM	Action
0	X	unchanged
0	X	unchanged
1	0	DISARM
1	1	ARM

SCO **SET CARRY OUT** **.95 μsec**



This instruction causes the carry out indicator to be set (=1).

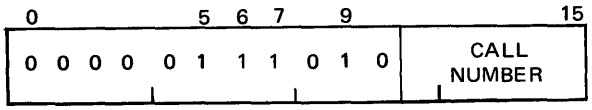
SOF **SET OVERFLOW** **.95 μsec**



This instruction causes the overflow indicator to be set (=1).

SYSTEM INSTRUCTIONS

SYCL **SYSTEM CALL** **7.36 μsec**

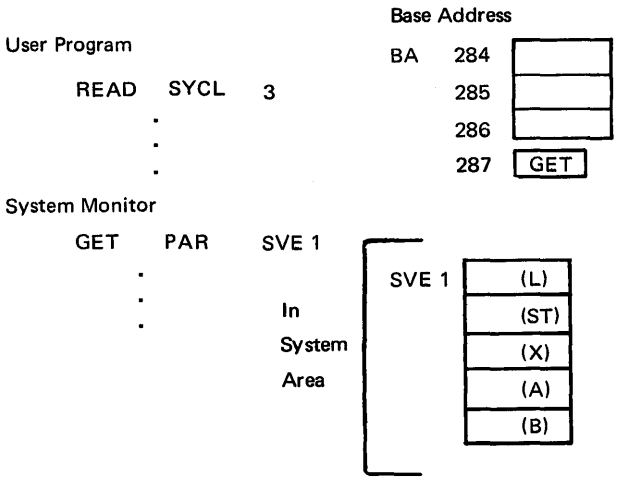


The system call causes a trap to a reserved area of 64 locations starting at the base address (284_{10}). The call number in bits 11 to 15 of the instruction, plus the base address gives the location through which control is transferred.

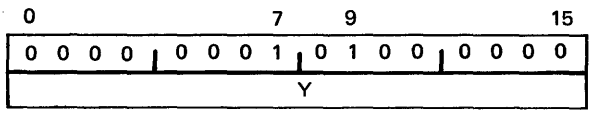
Control is transferred to the address stored in the reserved location. The instruction automatically stores the contents of the L, ST, X, A, and B registers in five locations starting with the address specified in the first word of the routine being called.

The mode, CRO, and OVF indicators of the status register will be reset (=0) before control is transferred.

Example:



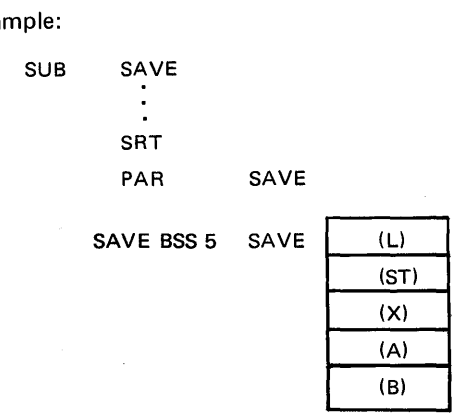
SRT **SYSTEM RETURN** **6.44 μsec**



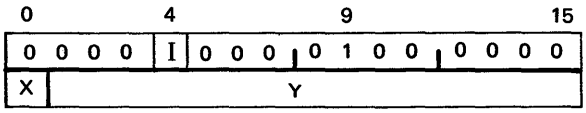
The system return is used to return control to the interrupted program after an interrupt or system call. It is a privileged instruction when memory mapping is implemented.

The instruction restores the contents of the L, ST, X, A and B registers at the time of the interrupt or system call, by returning on the same "save" location assembled in the first word of the subroutine and reloading the registers with the values stored in the five reserved locations.

Example:



TSL **TEST AND SET LOCK** **3.06 μsec**



For systems operating in a multiple processor environment, this instruction provides the capability of restricting access to routines which might be executed by both processors at the same time.

In such situations, each CPU must perform this instruction prior to entering the restricted sequence of instructions. Thus, protection is accomplished by individual routines and not by physical core locations.

This is a privileged instruction. It will test the effective location for zero; if it is, the location counter is incremented by one. If the effective location is not zero, the location counter is incremented by two and the lock location is set to zero.

This instruction may be indexed and indirectly addressed:

Example

```
SYSTEM                TSL
MONITOR               PAR     LOCK
CPU–1                 JMP     *–2
                      JSL     COMMON
-------------------------------------------
            COMMON    PAR     XSAV
COMMON                .
ROUTINE               .
MODULE No. 5          .
                      LDL     –1
                      STA     LOCK
            LOCK      BSS     1
-------------------------------------------
SYSTEM                TSL
MONITOR               PAR     LOCK
CPU–2                 JMP     *–2
                      JSL     COMMON
```

INPUT/OUTPUT SYSTEM

The capabilities of the 4700 system are:

1. Data chaining which permits scatter read; gather write techniques.
2. Mixed mode operations with different devices on same multiplexor channel.
3. Servicing of multiple slow devices on same channel.
4. High data transfer rates for fast devices.
5. Full word or byte data transfer.

I/O control operations are separated into three categories which direct the functions of three kinds of control equipment. To avoid the confusion of calling all of them instructions, the following nomenclature will be used:

Instructions are executed by the CPU;

Commands are executed by the channel;

Orders are executed by the device.

Data transfers may be either byte or word oriented. Full word data transfer is performed through the parallel I/O interface and is controlled by the Read Parallel and Write Parallel instructions.

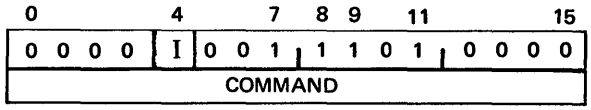
The parallel I/O interface is used to read and control all devices on the parallel I/O bus, such as: interrupts and special devices. The memory mapping unit uses an internal parallel I/O bus for the LSMP and LUMP instructions. The particular device is selected by the address data contained in the ACT command.

PARALLEL I/O (16 bit word oriented)

The ACT instruction is used to activate any I/O device which utilizes the parallel I/O interface. It is a double word instruction, the first word of which is in the extended op code format. The second word contains the control information or the address of the control information. The double-word instruction must occupy contiguous memory locations and may start on even or odd numbered locations.

PARALLEL I/O INSTRUCTIONS

ACT **ACTIVATE PARALLEL I/O** **1.84 μsec**



The ACT instruction is used to activate any I/O device which utilizes the parallel I/O interface. It is necessary to give an ACT only if more than one device is connected to the parallel interface.

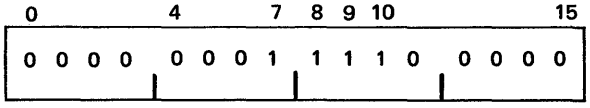
The interpretation of the control information depends on the device and the user's system requirements. It may be indirectly addressed. If bit 4 of the ACT equals one, the second word becomes a pointer to the command information. The pointer is in the indirect address format and may be indexed.

2nd word if I = 1

The I/O command is output directly from memory (L+1), without being loaded into the accumulator by the program.

The RDP and WTP use the extended op code format with the data being input into the accumulator or output from the accumulator. Transfer of each word is done under control of the CPU, using RDP or WTP, as necessary.

WTP **WRITE PARALLEL** **2.07 μsec**

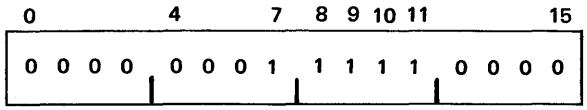


The WTP transfers a 16-bit data word from the A register to the active device on the parallel I/O interface. The A register must previously have been loaded with the data to be transferred.

The transfer occurs if the device is ready; and the location counter is incremented by two.

If the device is not ready when tested by the instruction, the transfer is not performed. Instead, the machine proceeds to the next instruction.

RDP **READ PARALLEL** **2.07 μsec**



The RDP transfers a 16-bit data word into the A register from active device on the parallel I/O interface.

The transfer occurs when the device is ready; and the location counter is incremented by two.

If the device is not ready when tested by the instruction, the transfer is not performed. Instead, the machine proceeds to the next instruction.

I/O CHANNELS (8 bit byte oriented)

The 4700 may have from one to four byte oriented data channels. The basic 4700 is equipped with one multiplexor channel. Additional optional channels may be added in any combination of either multiplexor or selector channels.

MULTIPLEXOR CHANNEL

The multiplexor channel is designed to service up to 64 half-duplex device controllers on a multiplex or "party-line" basis. The multiplexor channel is under direct control of the CPU and utilizes the CPU data and address paths to memory.

A SCC 4700 system may have one to four multiplexor channels. The multiplexor channel may operate in one of two modes:

1. Single Byte
2. Block

When in either the byte or block mode, more than one device controller may be actively engaged in data transfer at any one time. In either case, data transfer between the channel and the device controllers is on a byte oriented multiplex basis. The active device controllers need not operate in the same mode: i.e., some device controllers may be engaged in data transfer in the byte mode while other device controllers are transferring data in the block mode.

SELECTOR CHANNEL

A selector channel provides a buffered channel for up to 64 byte oriented device controllers on a block transfer basis. The selector channel provides for direct memory access at a high data rate between an I/O device and memory. Only one device at a time may be active on a selector channel.

The selector channel is activated by the CPU and proceeds autonomously of the CPU to access memory for command and control information. Utilizing the start address and byte count specified by the command and control information, the selector channel performs the data transfer asynchronously and independently of the CPU; the selector channel deactivates the channel at the end of the data transfer. Block transfers may be chained (that is, the selector channel can sequentially execute a series of block transfers).

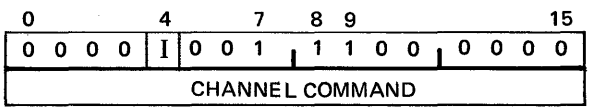
I/O SYSTEM OPERATION

The IOC instruction controls the operation of the multiplexor and selector channels. Each IOC transmits a channel command to the appropriate channel controller, which in turn selects the desired device.

CHANNEL I/O INSTRUCTION

The IOC instruction transmits the control information required by either the multiplexor or selector channel controller. This control information is called an I/O command and is sent directly to the channel controller which changes it to a sequence of signals acceptable to the control unit. The IOC is in the extended format and must be followed immediately by a channel command or the address of the channel command.

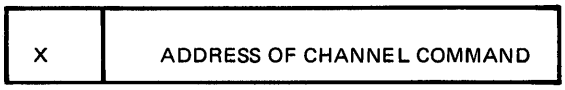
IOC **INPUT/OUTPUT CONTROL 2.99 μsec**



The IOC is a CPU instruction which transmits the control information required by either the multiplexor or selector channel controller. This control information is called a

channel command and is sent directly to the channel controller which changes it to a sequence of signals acceptable to the control unit.

The IOC must always be followed immediately by a channel command or the address of a channel command. One level of indirect addressing and indexing is allowed.



2nd word if I = 1

The IOC is executed in 2.99 μsec, providing an acknowledge is received from the device immediately. Otherwise, it delays, 0.30 μsec before checking availability again.

CHANNEL I/O COMMANDS

The I/O commands have a single word format. They are sent directly to the channel controller and specify information to select the following.

1. Indexing
2. Channel Operation
3. Block or character mode
   (used only with SIO)
4. Arm/Disarm interrupt
   (used only with SIO)
5. Channel
6. Device controller

There are 10 channel commands available to the programmer and they are:

		Format
HIO	Halt I/O	CC
SIO	Start I/O	CC
XMT	Transmit Character	CC
EOA	Execute Order in A	CC
OUS	Output Unit Status	CC
TWC	Terminate when complete	CC
SDR	Skip if Device Ready	CC
SDA	Skip if Device Available	CC
IIU	Input Interrupting Unit	CC
IUS	Input Unit Status	CC

Each channel command must be immediately preceded by an IOC instruction or contained in the location specified by the indirect address of the IOC.

(See Figure 1)

FUNCTIONS OF I/O CHANNEL COMMANDS

HIO - Halt I/O

Causes addressed device to immediately halt and clears settings of the entire device including status register. Inhibits interrupt from the device. May cause information to be lost.

SIO - Start I/O

Causes channel controller to be activated so that order may be passed to device. Sets block and interrupt mode bits. Initiates channel operation for block transfers by starting automatic transmission of data under control of the multiplexor or selector channel.

XMT - Transmit

Inputs or outputs character to/from least significant half of A. Direction of transfer will be determined by the device code:

even — input

odd — output

EOA - Execute Order in Accumulator

This command is device dependent. Data in the least significant half of the A register will be output to the device controller to initiate the desired operation.

IDN - Input Device Number

This command is not available to the programmer. It is used by the micro code of the CPU to identify an interrupting device when servicing the M-channel for block transfers.

OUS - Output Unit Status

Transmits 8 bits of data from the least significant half of the A register to the status register of the device controller. Bits of the status register may be set only; the device status cannot be reset to cleared by this command.

TWC - Terminate when Complete

This is the usual command to be given to stop a device. Device will stop data transfer, but completes current mechanical operation. The status register is not cleared. Device will give a channel interrupt, if armed.

SDR - Skip if Device is Ready

Tests status register of device and increments L register by two, if the device is not *ready.*

IIU - Input Interrupting Unit

Device number of interrupting unit is input into the least significant 6 bits of the index register.

IUS - Input Unit Status

The status register of the device being addressed is input into the least significant 8 bits of the A register.

SDA - Skip if Device is Available

Tests status register of device and increments L register by two, if the device is *available.*

I/O DEVICE ORDERS

To initiate the I/O operation, for either input or output a device order must be transmitted to the device controller. They give specific instructions to the device and must be transmitted before the operation can be performed.

The interpretation of the control information depends on the device. The codes for each device and the format of orders for them are contained in the manual concerning the individual peripheral device.

BYTE TRANSFERS

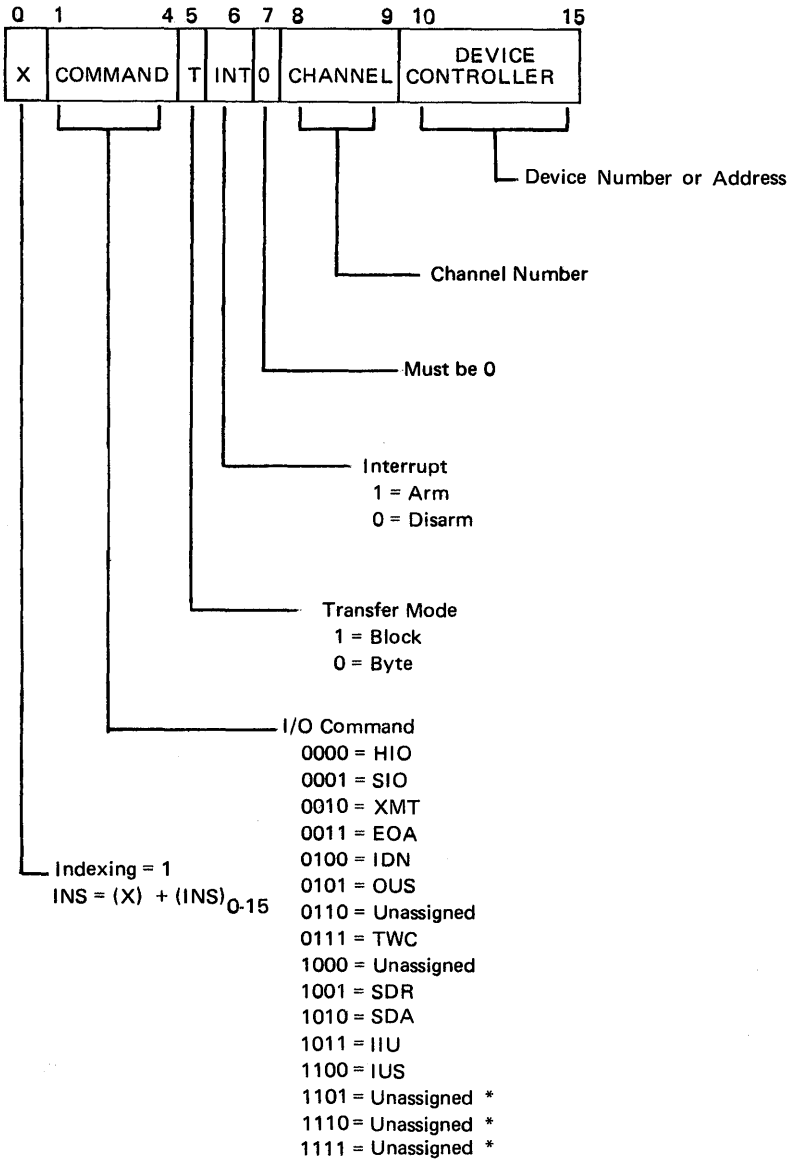
Both the multiplexor and selector channels accept data in either byte or block oriented modes. In the single byte mode, data transfers usually are under program control, with data being output or input to the low order eight bits of the accumulator.

BLOCK TRANSFERS

Block transfers directly to or from memory can be made through either the multiplexor or selector channel. In the selector channel, block transfers of multiple bytes of data occur asynchronously and independently of the CPU. Memory cycles may be stolen, if the same memory module is accessed simultaneously.

In the multiplexor channel, block transfers are initiated by execution of the Start I/O command as in the selector channel. Transfer of data then procedes automatically until completion, without further attention from the program.

The format used for these channel commands is:



* If used, these codes will also cause data to be input to the A register from the channel.

Figure 1.

Block transfers on the multiplexor channel are programmed exactly the same way as for the selector channel. The only differences occur in the way the hardware handles the data transfer. Those to be considered are:

1. slower data rate.

2. address and byte count are kept in core memory, rather than channel registers.

3. the micro-code of the CPU handles the interrupt and data transfer, which results in stealing some CPU time and memory cycles.

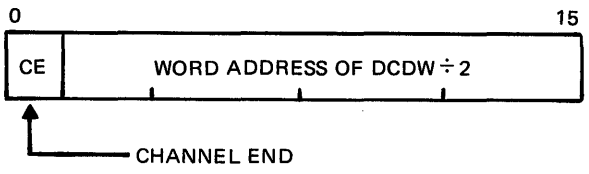
4. multiple devices can be active on the multiplexor channel at the same time, but only one device at a time can be active on the selector channel.

To perform a block transfer on either the multiplexor or selector channel, certain steps must be accomplished.

1. An input/output data area is reserved in memory

2. A pointer to the Data Control Double Word (DCDW) controlling this transfer is placed at the proper location in dedicated core memory for the particular device and channel. The CPU will look to that location (designated in the channel command) for a pointer to the necessary DCDW. (The channel end bit should be set to zero.)

The format of the DCDW pointer is:



DCDW POINTER

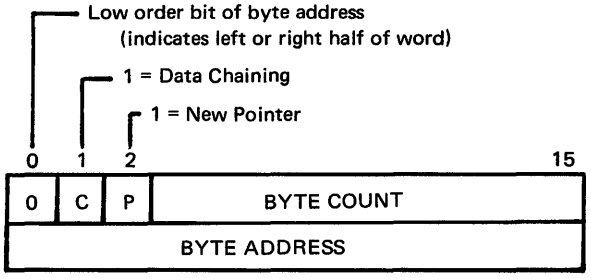
The byte address in the pointer real core must be doubled to determine the location of the DCDW.

Therefore, it is necessary that the first DCDW of a data chain begins on an even location. All DCDW's in the chain must follow in contiguous locations.

After transfer is complete, the channel end bit of the pointer is set. If the program logic uses this bit to determine completion of transfer, it must be reset before the same pointer is referenced again.

3. The DCDW (or series of DCDW's) may be assembled anywhere in core storage. They give the byte count and byte address of the data area (or areas) to/from which data will be transferred.

Format of the DCDW is:



DATA CONTROL DOUBLE WORD

4. A device order must be written to initiate the device controller. It must be the first byte to be output. The device order may be placed with the data to be output; or output from a table of device orders.

A device order must be transmitted to activate the device controller for all transfers, either input or output. For block transfers, it will be output from memory. (For the character mode, it must be placed in the accumulator and transmitted with an EOA (Execute Order in A) channel command).

The format of the device orders are device dependent and may be found in the peripheral manual for the device being used. However, the device order for "normal" operation (forward, with leader, etc.) will be 00000000_2.

5. After this information has been prepared and assembled in the proper sequence and format, the desired data transfer can be accomplished by writing an IOC instruction followed by an SIO channel command.

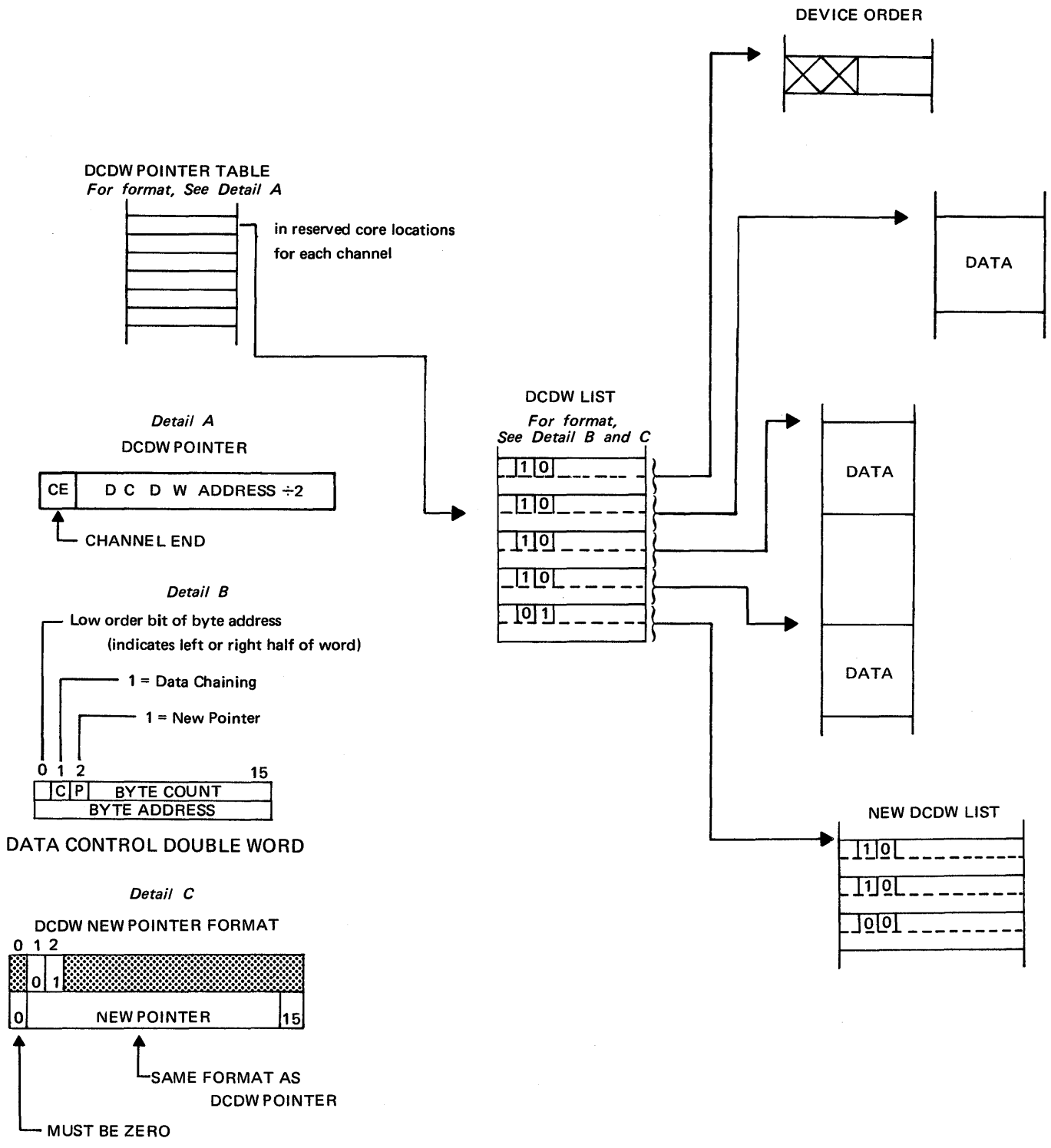
Example:

```
IOC
SIO OE05
```

On execution of the SIO command, the selector channel will locate the data to be transferred by means of the DCDW and perform the transfer asynchronously and independently of the CPU.

For block transfers on the multiplex channel, the CPU performs the necessary update and transfers each byte directly from memory as each device becomes available. The accumulator is not affected for either input or output.

BLOCK TRANSFER OPERATION



MEMORY MAPPING

Memory mapping is a technique which facilitates operation in a multiprogrammed environment. The concept of virtual and real memory is essential to understanding memory mapping.

> *Virtual* memory for the SCC 4700 is an imiginary memory space of 32K, contiguous, 16-bit words. Virtual memory allows the programmer to write a program on the assumption that he may utilize the entire core memory.
>
> *Real* memory is the actual memory space of the program being executed. Real memory may be larger or smaller than 32K.

Virtual memory is an abstraction and exists only in the mind of the programmer. It is, however, an important concept in the creation of a multiprogramming system because it allows a program to be divided into segments. A segment of virtual memory is called a *page* and contains 512 words or locations. The 32K of virtual memory is divided into 64 pages of 512 words.

Real memory is also divided into 512 word segments which are called *blocks*. Since the maximum size addressable by the SCC 4700 is 64K, there may be 128 blocks in real memory.

By segmenting the program in virtual memory and putting it into blocks of real memory, virtual memory is *mapped* onto the real core memory area. These concepts are illustrated in the following diagram:

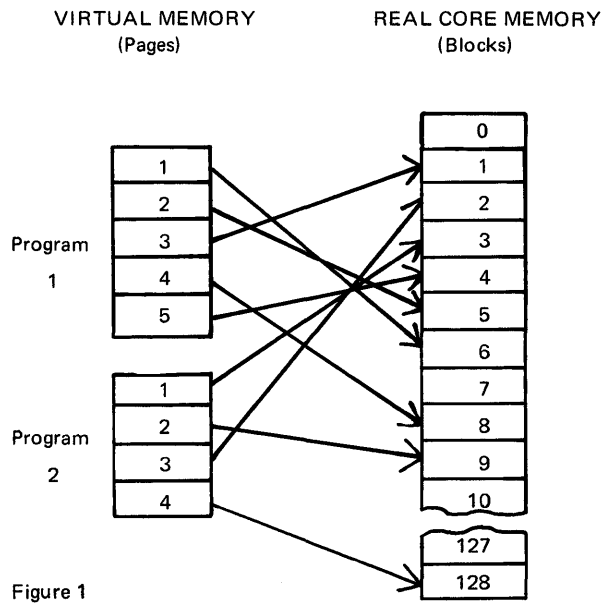


Figure 1

Since every program is assembled independently (without regard to where it may be placed at execution time), its real location is determined by the executive system through the use of the associative registers in the memory map unit.

The program is broken down into pages and loaded into blocks of real core locations. The blocks need not be consecutive, nor does the whole program need to reside in core at a given time.

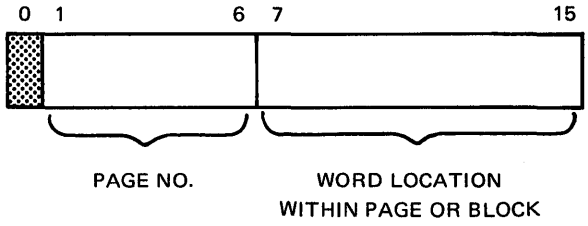


Figure 1 is an example where no two consecutive blocks follow one another.

Whole programs or only a page or two at a time may be in core. The executive system maintains a *page table* which identifies the block location of each page for each program being executed.

When an instruction is executed, a match for the desired page is sought in the memory map registers. If it is not found, the hardware traps to either the system or the user page table pointer, depending on the setting of the mode bit, and loads the desired page table word in the next available register.

The page table pointer contains the starting address (α) of a list of pages being used by the program.

There are multiple page tables as required. One system and one user page table are in use by the system at any given time.

The next available memory map register is then loaded automatically and indirectly from the page table with the contents of α + page number.

The format of the page table word and consequently of the memory map register is shown in Figure 3. By means of the memory map registers, the virtual addresses of the assembled program instructions are transformed to the addresses of real core locations where they are actually stored.

When memory mapping is implemented, every effective address formed by memory reference instructions must be transformed into a real address before execution.

When a program instruction is executed, it forms a 15-bit effective address after all address modification is complete. This is placed in the S register of the CPU.

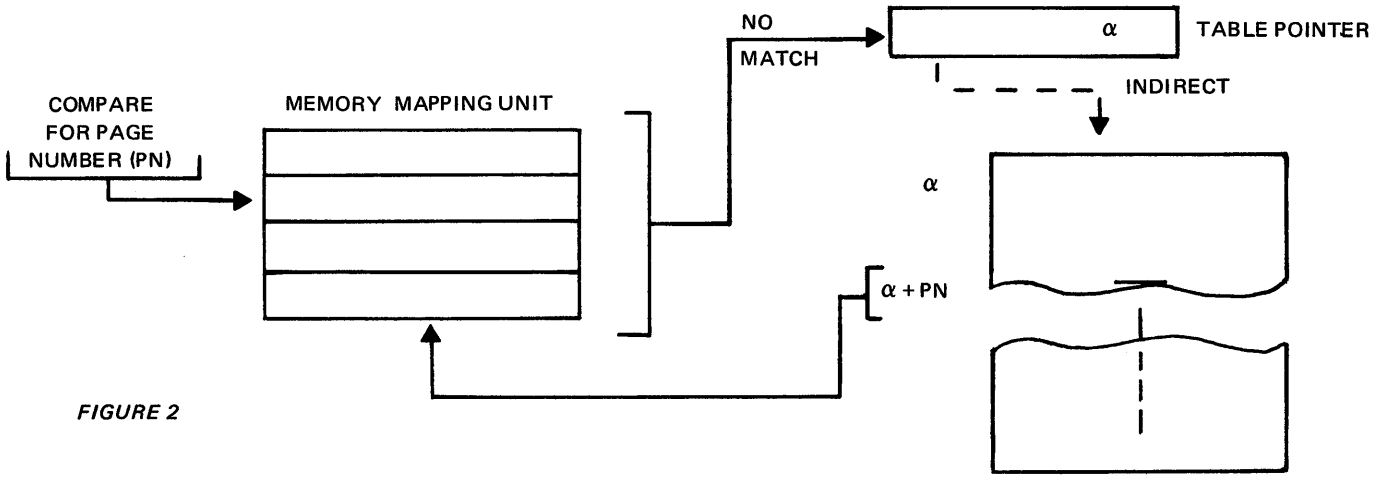
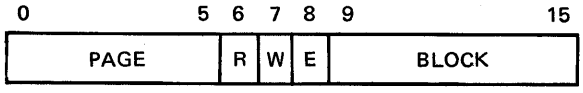


FIGURE 2

The associative registers are loaded by the map unit from an entry in the 64 word page table contained in memory. The entry from the program page table is combined with current machine conditions to form the format described above. The entry is assumed to have the following format:



The memory map unit also contains two 16-bit registers designated the System Map Table Pointer (SMTP) register and the User Map Table Pointer (UMTP) register. These registers are loaded with the starting address of the system map table and a user map table, respectively. The mechanism for loading each of these registers is a LSMP or LUMP instruction. Note that since these are 16-bit registers, the tables may be located anywhere in memory on any word boundary. Loading of either of these registers automatically clears the associative registers.

There is one-bit register in the machine designated the MODE which indicates whether the computer is in the user or system mode. The MODE indicator is part of the machine status register and thus is automatically saved when an interrupt occurs. System calls, system traps, or interrupts set MODE to the system state (after saving its current status).

MEMORY MAPPING

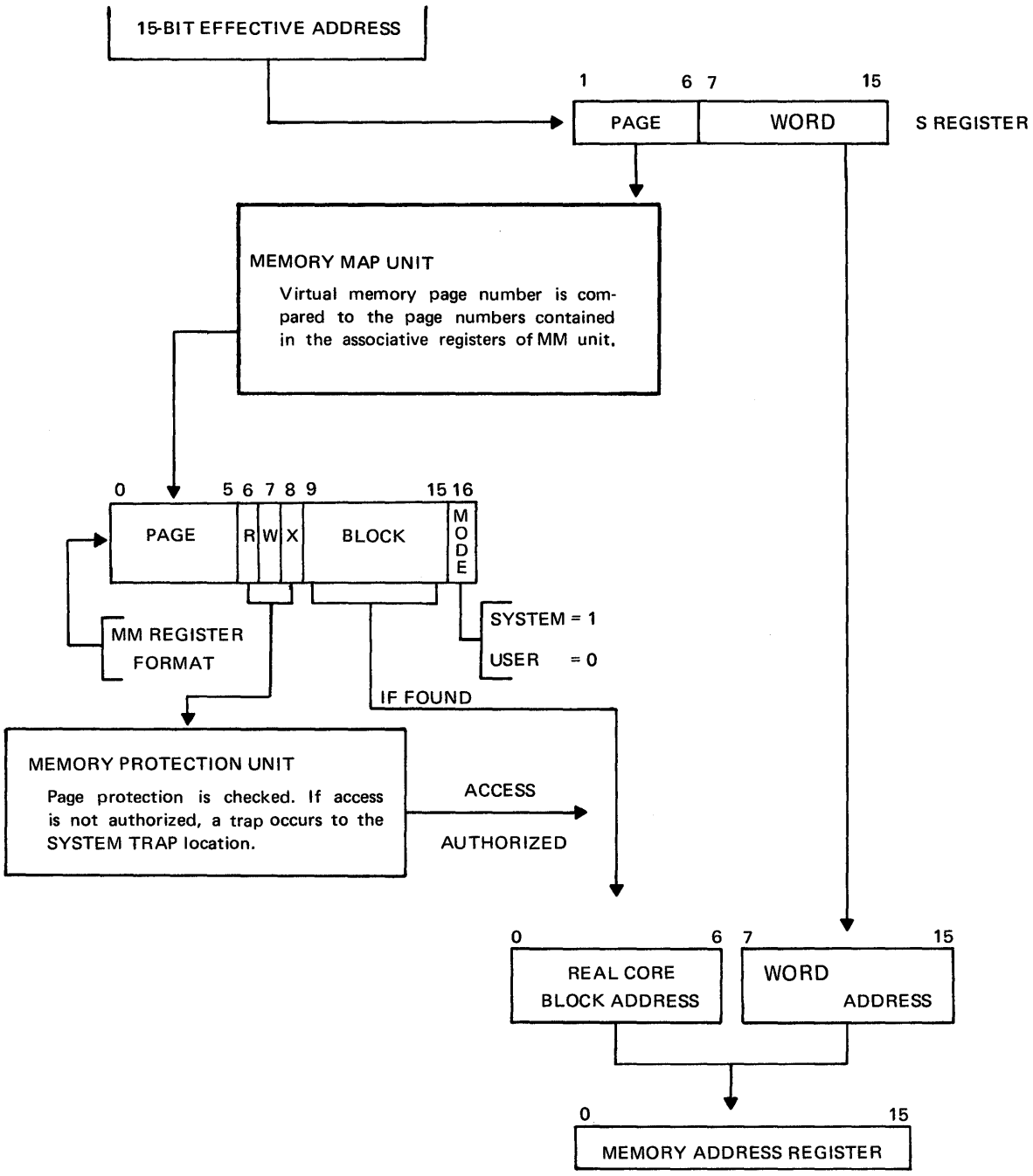
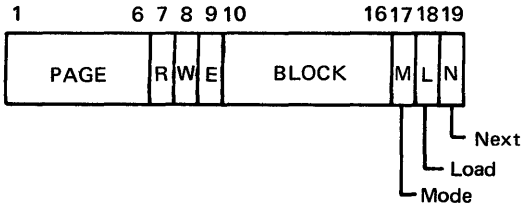


FIGURE 3

MEMORY PROTECTION

The SCC 4700 memory map and protection unit contains 8 to 32 associative registers and two address registers. A minimum of eight registers (two units) must be implemented initially. The number of associative registers may be increased in groups of four. The format of each associative register is as follows:



Where Page is a 6-bit page number to be compared with the high order 6 bits of the address.

R = 0 if read access is not permitted for this page.
R = 1 if read access is permitted.

W = 0 if this page cannot be written into.
W = 1 if the page may be written.

E = 0 if the page may not be executed.
E = 1 if execution is permitted

Block is a 7-bit block number which is used as the high order portion of the 16-bit address presented to memory.

Mode = 1 if this register contains system mapping information.
Mode = 0 if this register contains user map information.
Load = 1 if this register is loaded.
Load = 0 if this register is empty.
Next = 1 if this is the next associative register to be loaded.

Memory requests are generated by the CPU and are presented to the map unit. The memory request consists of the following:

A. A 15-bit address.
B. A bit indicating whether this request is to be made to the user's space or to the system's space.
C. The type request being made, i.e., read, write, execute.

The map unit partitions the 15-bit address from the CPU into two fields: a 6 bit page number field and a 9 bit word address field.

The map unit compares the information presented with the memory request to all the associative registers simultaneously for a match. The map unit compares the MODE bit in the associative register with the mode of the current request. If the modes agree, the page number from the CPU is compared with the page number in the associative register. These bits are decoded and compared simultaneously.

A match causes the request type to be checked against the protection bits in the associative register. An allowable request results in the block number in the associative register being linked with the 9-bit word address to form a 16-bit memory address. Memory is then accessed with this address.

If the memory request cycle is one which is not allowed by the memory protection bits, an interrupt signal is sent to the CPU.

The hardware, upon detecting that the page number in question is not presently in any associative register, initiates the following actions:

A. The current page number is added to the SMT P or the UMTP depending upon whether the current mode is system or user, respectively. This forms the address of the entry in the map table which corresponds to the current page.

B. This address is used to load the next available associative register and is assumed to contain memory protection bits and the block number for the page in question. Associative registers are allocated in a simple round robin fashion. Loading of an associative register causes the register NEXT bit to be read and the NEXT bit in the following associative register to be set.

System page 0 is not mapped; hence, all interrupts and traps are routed to block zero.

CONTROL CONSOLE
and
DISPLAY PANEL

The display/control console itself is only eight inches high, by 22-5/16" wide and 5-1/4" deep. The console is mounted on the basic machine in the front of the blue and grey cabinet. Inserted into the cabinet at a 10° slant, the console is placed above a convenient 16" X 26" writing shelf. Chrome-plated pushbuttons and knobs accent its black satin-finished surface.

The register displays and status indicator lights are hidden behind a pane of glare-free black glass. Only when lighted are the back-lighted indicators visible. However, a shadow grid in the display panel marks the relative position of the indicators when they are off.

Permanent lettering identifies the switches and the display registers. The indicator lights of the display panel are different colors for better visibility and the status indicators are colored and labeled for quick identification.

Color scheme for display registers and status indicators:

Yellow - L (register)
White - Reg (select)
Orange - SW (register)
- Carryout (CRO)
- Overflow (OVF)
Red - Halt
- Memory Parity Error (MPE)
Green - Power

This display panel contains 64 lamps (28 volt, 40 milliamp), which give a medium bright, diffused light. These lamps are replaceable from the rear without removing the display panel from the cabinet.

Thirty-one chrome-plated pushbuttons and two rotary switches are utilized in this operator interface station to control the computer system. If desired, the display/control console can be mounted as a remote unit. Connected to the CPU by up to 30 feet of cable, this optional remote console may also contain a selectric typewriter for operator messages and commands.

The contents of the display registers are normally updated at the line voltage rate (50-60 cycles) while the CPU is in the run mode. It is possible to inhibit the update of the displays either manually, by a switch on the maintenance panel inside the cabinet; or under program control, by disarming the 60 cycle real-time clock.

If the display update is inhibited, the register displays will be updated only on execution of a halt, and when data is entered or displayed by use of the control console.

If display update is not inhibited, the M register will contain the current reading of the real-time clock. (This is equal to the contents of Location 2.)

If memory mapping is not implemented, all addresses displayed will be real core addresses.

If memory mapping option is implemented, then addresses displayed will normally be virtual memory addresses (or addresses before mapping is performed). However, provision has been made to show the real core address when desired.*

*A switch on the maintenance panel inside the machine will inhibit mapping. (These will be the real core locations of the instructions or data.)

The switches which will be operative while the CPU is in the run mode and power switch is in ON position will be:

1. Lamp test
2. Console interrupt
3. System reset
4. Halt
5. Reg Select
6. Power
7. Run
8. The 16 switches of the Switch Register.

A. DISPLAY PANEL

1. Status (of CPU)

   The status indicators show the state of certain internal conditions. Each indicator is marked to indicate the

corresponding function. When lighted, their color calls the attention of the operator to the internal state of the CPU.

The indicators and their colors are:

		Color
1. CRO	(carryout)	orange
2. OVF	(overflow)	orange
3. MPE	(memory parity error)	red
4. HALT		red
5. POWER	(power on)	green

2. L (L-register)

This register display shows the contents of the location counter. It will contain the address of the next instruction to be executed. This is the real core address, except when memory mapping is implemented.*

*A switch on the maintenance panel inside the machine will inhibit mapping. (These will be the real core locations of the instructions or data.)

3. REG (Register)

The desired register is selected by use of the *Reg Select* switch. When the display button is depressed, the contents of the register indicated by the *Reg Select* knob (M, S, A, B, or E) are displayed in this row of indicator lights (ON=1).

The M register is the path to and from memory. On execution of a halt, the M register contains the next instruction.

If the M register is selected and the display button is pressed, the contents of the address specified by the location counter is displayed in the *Reg* indicators.

4. SW (Switch Register)

Data to be entered in a register or memory location from the console is placed in the switch register by the operator. These indicators show the current contents of the switch register (ON=1).

The switch register is an external holding register. Data entered in this register by use of the 16 console switches is retained in the switch register until stored by the operator or under program control. The register may be changed or cleared as necessary without effecting either memory or the registers of the CPU.

B. CONTROL CONSOLE

1. Switch Register
   a) Depressing one of these 16 buttons will set (=1) the corresponding bit of the switch register and illuminate the indicator in the *SW* row of indicators (ON = 1) for that binary position. Contents of the switch register may be entered into one of the other registers or a memory location from the control console; or into the accumulator under program control. (See instruction description of Load Accumulator from Switches.)

   b) Sw Clear (Switch Register Clear)
   Resets (=0) entire switch register and all switch register indicators on display panel (OFF = 0).

2. Function Switches (L to R)
   a) Power (rotary key switch)
      1) OFF - all power supplies off.
      2) ON - all power supplies on; master clear of all internal and external interrupts when bringing power up occurs automatically.
      All console switches are operative.
      3) SW - only following switches are operative in run or halt mode:
         1. sw clear
         2. switch register
         3. lamp test
         4. console interrupt

      Data may be entered into switch register for program modification and control - but registers and memory locations cannot be altered or examined from the control console.

4) LOCK

Power is on and all control console switches except lamp test are inoperative. Program execution cannot be interrupted or data entered from the console.

b) Lamp Test

Supplies power to all indicator lamps on display panel at same time regardless of state. No data is altered nor functions affected. This switch is always operative, as long as machine power is on.

c) Console Interrupt

Causes a trap to location 4 to occur. The console interrupt will be serviced by the software routine written to handle this condition for the system being executed. If the console interrupt occurs while the CPU is in the halt mode - the CPU will transfer to the run mode.

d) System Reset

Operative only when power switch is in ON position; disabled in the run mode. This causes all the indicators and switch settings to be reset. It is the master clear signal which will initialize the central processor and all channel interrupts.

The interrupt system will be disabled and all interrupts disarmed. The status register will be reset; except for Halt and Interrupts Disabled, which will be set.

Data in core memory and the programmable registers (A,B,X,E and F) will not be changed.

e) Program Load

Operative only when power switch is in ON position; disabled in the run mode. Programs may be loaded from any device. See description of Initial Program Load for steps required to use this feature.

f) Store

Operative only when power switch is in ON position; disabled in the run mode. Used to store data from the switch register into the address specified by the L-Register.

Since data stored must pass through the M register, the data last transferred to memory can be displayed in the *Reg* row of indicators by selecting the M register.

g) Increment and Store

Operative only when power switch is in ON position; disabled in run mode. The L register is incremented by one (L+1), and then, the contents of the switch register are stored into the address specified by the new value of the location counter.

h) Increment and Display

Operative only when power switch is in ON position; disabled in the run mode. The L register is incremented by one (L+1), and then, the contents of the memory location specified by the new value of the location counter are brought into the M register. This data may be displayed in the *Reg* row of indicators by selecting the M register with the *Reg Select* knob.

i) Display

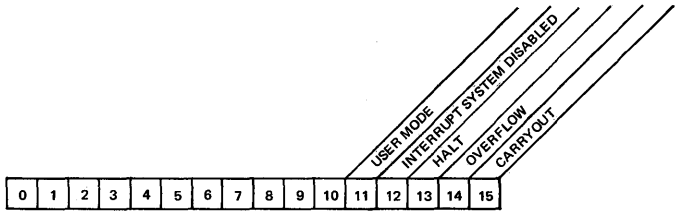
Operative only when the power switch is in ON position; disabled in the run mode.

This switch brings the contents of the memory address specified in the L register into the M register. This data may be displayed in the *Reg* row of indicators by selecting the M register with the *Reg Select* knob.

j) Set Status (Register)

Operative only when the power switch is in ON position; disabled in the run mode.

This switch transfers the contents of the switch registers to the corresponding indicators in the status register. The format of the status register is:



The Halt indicator cannot be set (=1) or reset from the switch register. It can be set only by the Halt button, Halt instruction and System Reset Button.

k) Set L (L register)

Operative only when the power switch is in ON position; disabled in the run mode.

This switch transfers the contents of the switch register to the location counter.

l) Set Reg (indicated by Reg Select)

Operative only when the power switch is in ON position; disabled in the run mode.

This switch transfers the contents of the switch register to the register indicated by the setting of the Reg Select knob. They are:

M - Memory register
X - Index
A - Accumulator
B - Extended Accumulator
E - Exponent register (optional)

m) Run

Operative only when the power switch is in ON position.

Resets halt indicator and puts machine in run mode.

n) Halt

Operative only when the power switch is in ON position.

This switch stops program execution. If sets (=1) the halt indicator and updates the display registers.

Only the CPU is halted; it will not affect input/output devices which are running. The CPU is in a "wait" state and may be interrupted from a halt by the following interrupts:

1. Power off
2. Power on
3. Console interrupt
4. Channel interrupts
5. External interrupts
6. Real-Time clock (traps on zero)

These interrupts cause the machine to go to run. The RTC update does not cause machine to run (except when the trap location goes to zero).

o) Reg Select

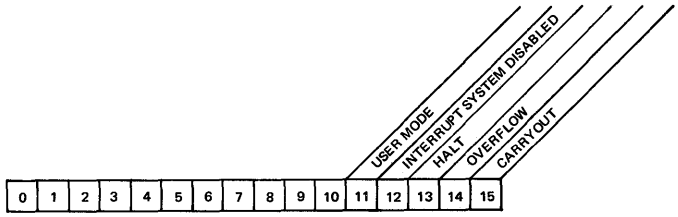
Operative only when the power switch is in ON

By using this switch, the operator may select which register (M, X, A, B or E) is to be displayed in the *Reg* row of indicators; or the register into which data will be entered by the *Set Reg* button.

THE STATUS REGISTER

The status register is a composite of five condition indicators, which set the mode of operation of the machine and/or indicate certain special conditions have occurred.

The format of the status register is:



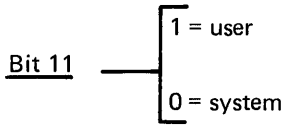
The state of these indicators (0 or 1) can be changed from the control console; under program control; or by the CPU in response to changes in processing or the operating environment.

MODE INDICATOR

The user mode indicator is meaningful only when memory mapping is implemented. When the optional memory mapping unit is installed and operating, bit 11 of the register indicates the mode of the machine; if set (=1), the CPU is in the user mode. It will be reset (=0) when executing instructions in the system mode.

The machine will always be in the system mode if memory mapping is not implemented.

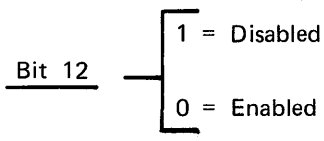
Mode Indicator



INTERRUPT INDICATOR

The interrupt system may be enabled or disabled under program control. If enabled, interrupts will be allowed to go active according to their priority. If disabled, interrupts will be remembered if armed but will not be allowed to alter the execution of the program.

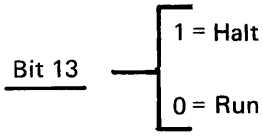
Interrupt System



HALT INDICATOR

The CPU may be halted by execution of the Halt instruction or by pressing the Halt button on the control console. Only single step execution of instructions can be performed while this indicator is set.

Halt



The Halt indicator can be set (=1) only by the Halt button, the Halt instruction and the System Reset button. It can be reset (=0) only by certain interrupts; some traps; and the Run button.

It can be neither set nor reset by transferring data from the switch register by use of the set status button on the control console.

OVERFLOW INDICATOR

This indicator (Bit 14) will be set if the result of the arithmetic operation in the adder exceeds the maximum signed magnitude quantity which can be contained in the accumulator. In other words, if the sign of the two operands are equal, but the sign of the result is different, the overflow indicator will be set. An arithmetic operation may result in both an overflow and carryout.

If overflow occurs, the results of the operation (contents of the register) are not valid. When overflow occurs, it always sets the OVF indicator (bit 14); however, if overflow does not occur, this will not reset (=0) the indicator.

CARRYOUT INDICATOR

This indicator (Bit 15) will be set if a carry occurs in the adder from the high order (Bit 0) position. Thus, if the arithmetic operation results in a quantity greater than the accumulator, the carryout indicator is set.

Carryout occurs according to the result of the arithmetic operation.

SYSTEM TRAP INDICATOR

Location five in core memory is reserved for the system trap indicator. If a system trap occurs, a system call will be executed on the location stored in location six. At that time the system trap indicator will be set to show what condition caused the trap.

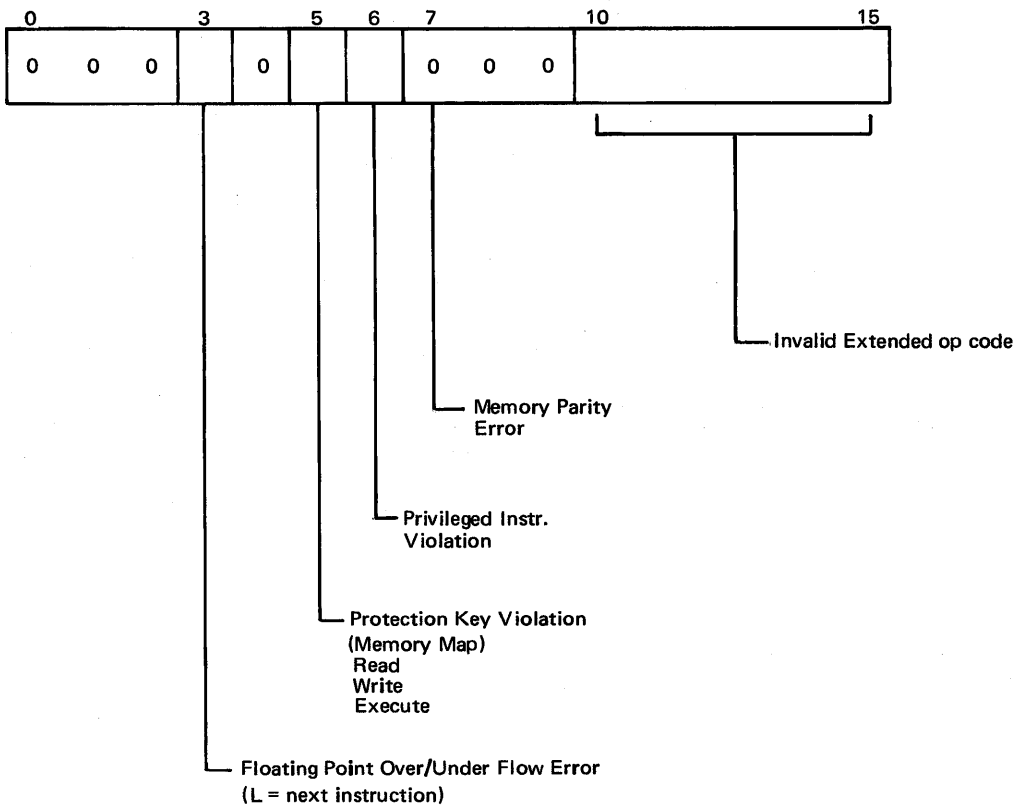
If bits 3, 5, 6, or 7 are set (=1), it will indicate the condition shown on the diagram. If more than one error condition occurs simultaneously, all of the corresponding bits in the trap indicator will be set.

If the system interrupt occurs and bits 3,5,6, and 7 are all equal to zero, this will indicate that an attempt was made to execute an unimplemented instruction.

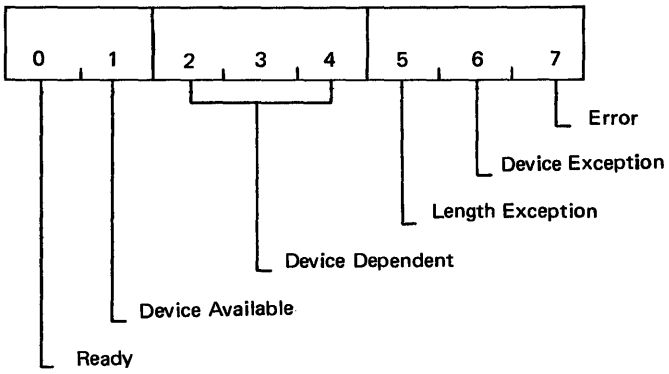
Since all sixteen possible combinations are utilized in the standard machine for basic operation codes, any unimplemented instruction must be of the extended op code set. These commands have the format:

The operation code is 0000 (bits 0 to 3); the extended operation code is contained in bits 7 to 12. When the trap occurs, bits 3,5,6, and 7 of the system trap indicator are set zero and bits 7 to 12 of the instruction which caused the trap are stored in bits 10 to 15 of the indicator word. The location counter will contain the address of the instruction which caused the trap.

In the basic machine (without memory parity, memory mapping unit, or optional instruction sets), only the unimplemented instruction trap is operative.



DEVICE STATUS BYTE



DEFINITIONS:

Bits

0 - Ready
1 - Device Available
2 - Device Dependent
3 - Device Dependent
4 - Device Dependent
5 - Length Exception ① ② *
6 - Device Exception ② ③ ④ *
7 - Error ② ⑤ *

* Notes

1. Input or output
2. May be set by OUS
3. Error flip-flops not set; device not terminated.
4. Types of conditions:
   Tape mark on tape
   Special character
   End of tape (EOT)
   Beginning of tape (BOT)
   Out of paper
5. Types of conditions:
   Parity error
   Rate overrun
   Card jam
   Interlocks open
6. Status byte is cleared by HIO, SIO, and MCLR.
7. Interrupts are generated only on termination (except for HIO and MCLR).
8. Devices are automatically terminated for emergency conditions, such as:
   (a) broken tape
   (b) interlocks open
   (c) mechanical problems

PERIPHERAL DEVICE CODES

(700 Series)

INPUT			OUTPUT			REMARKS
OCTAL	HEXA-DECIMAL	FUNCTION	OCTAL	HEXA-DECIMAL	FUNCTION	STANDARD DEVICES
00	00		01	01		
02	02	Read Storage No. 1	03	03	Write Storage No. 1	Drum or Disk
04	04	Keyboard No. 1	05	05	Printer No. 1	Teletype
06	06	Tape Reader No. 1	07	07	Tape Punch No. 1	ASR 33
10	08	Byte Input No. 1	11	09	Byte Output No. 1	
12	0A	Read Storage No. 2	13	0B	Write Storage No. 2	Drum or Disk
14	0C	Keyboard No. 2	15	0D	Printer No. 2	Teletype
16	0E	Tape Reader No. 2	17	0F	Tape Punch No. 2	ASR 35
20	10		21	11	Line Printer	
22	12		23	13	X-Y Plotter No. 1	
24	14	Keyboard No. 3	25	15	Printer No. 3	IBM Selectric
26	16	Card Reader	27	17	Card Punch	
30	18	Byte Input No. 2	31	19	Byte Output No. 2	
32	1A		33	1B	X-Y Plotter No. 2	
34	1C		35	1D		
36	1E		37	1F		
40	20	Incremental Tape Read A	41	21	Incremental Tape Write A	
42	22	Analog - Digital Converter (in)	43	23	Digital - Analog Converter (out)	
44	24	System in A No. 1	45	25	System Out A No. 1	Unique Device
46	26	System in B No. 1	47	27	System Out B No. 1	(Controller Oriented)
50	28	Byte Input No. 3	51	29	Byte Output No. 3	
52	2A		53	2B		
54	2C	System in A No. 2	55	2D	System Out A No. 2	Unique Device
56	2E	System in B No. 2	57	2F	System Out B No. 2	(Controller Oriented)
60	30	Magnetic Tape A No. 1 Read	61	31	Magnetic Tape A No. 1 Write	
62	32	Magnetic Tape B No. 1 Read	63	33	Magnetic Tape B No. 1 Write	
64	34	Magnetic Tape C No. 1 Read	65	35	Magnetic Tape C No. 1 Write	
66	36	Magnetic Tape D No. 1 Read	67	37	Magnetic Tape D No. 1 Write	
70	38	Magnetic Tape A No. 2 Read	71	39	Magnetic Tape A No. 2 Write	
72	3A	Magnetic Tape B No. 2 Read	73	3B	Magnetic Tape B No. 2 Write	
74	3C	Magnetic Tape C No. 2 Read	75	3D	Magnetic Tape C No. 2 Write	
76	3E	Magnetic Tape D No. 2 Read	77	3F	Magnetic Tape D No. 2 Write	

SCC 4700 TRAPS & INTERRUPTS

RESERVED CORE ASSIGNMENTS			PRIORITY LEVELS		TIMING	
DECIMALS	**HEXADECIMAL**	**DESCRIPTION**	**SEQUENCE**	**REMARKS**	**μsec**	**MULTIPLE FUNCTIONS**
0	0	Power Up Interrupt	2		6.67	
1	1	Power Down Interrupt	5		6.67	
2	2	Real-Time Clock (location being decremented)	8		1.38	Decrement Location
3	3	Real-Time Clock Trap location			6.67	Decrement, Test and Trap
4	4	Console Interrupt	9		6.67	
5	5	System Trap Indicator	1	Unimplemented Instruction ①	7.59	Unimplemented Instruction
6	6	System Trap Location	3	System Protection ②	7.68	Protection Violation Privileged Instruction Memory Parity
7	7	Unassigned				
8	8	Channel 1 Interrupt	7	Service Request (lowest location has highest priority)	6.90	Service Request (I/O or external interrupts)
9	9	Channel 2 Interrupt				
10	A	Channel 3 Interrupt				
11	B	Channel 4 Interrupt				
12 – 27	C – 1B	Reserved for External Interrupts				
28 – 283	1C – 11B	Reserved for DCDW Pointers (64/channel)	6	Transfer Request	7.60 10.12 12.40	Normal Transfer (block mode) Transfer with data chaining Transfer with chaining and new pointer.
284 – 347	11C – 15B	Reserved for System Calls (64 monitor entry points)			7.36	Storage of register plus transfer

① In basic machine, only unimplemented instruction trap of the system interrupts is operative.

② Another interrupt is implemented with memory mapping . Called "Map No-match", it is 4th in priority sequence.

FUNCTIONAL MNEMONIC LIST

LOAD AND STORE INSTRUCTIONS

MNEMONIC	OP CODE	INSTRUCTION NAME	LOGICAL FUNCTION	FORMAT TYPE	TIME μsec	INSTRUCTION SET
LAS	0640	Load Accumulator From Switches	(Switches) → A	E	.95	ST
LDA	CXXX	Load Accumulator	(Q) → A	B	1.84	ST
LDB	2XXX	Load B Register	(Q) → B	B	1.84	ST
LDD	0680	Load Double	(Q) → A; (Q+1) → B	D	3.56	DP
LDF	0688	Load Floating	(Q) → A; (Q+1) → B; (Q+2) → E	D	4.48	FP
LDH	1XXX	Load Halfword	$0 \rightarrow A_{0\text{-}7}$; (Q) → $A_{8\text{-}15}$	B	2.09	ST
LDL	64XX	Load A, Literal	$(INS)_7 \rightarrow INS_{0\text{-}6}$; (INS) → A	L	.92	ST
LDLB	6CXX	Load B, Literal	$(INS)_7 \rightarrow INS_{0\text{-}6}$; (INS) → B	L	.92	ST
LDS	0648	Load Status	(Status) → A	E	.95	ST
LDX	0678	Load Index	(Q) → X	D	2.76	ST
STA	7XXX	Store Accumulator	(A) → Q	B	1.84	ST
STB	3XXX	Store B Register	(B) → Q	B	1.84	ST
STD	0690	Store Double	(A) → Q; (B) → Q+1	D	3.68	DP
STF	0698	Store Floating	(A) → Q; (B) → Q+1; (E) → Q+2	D	4.60	FP
STH	5XXX	Store Halfword	$(A)_{8\text{-}15} \rightarrow Q$	B	1.84	ST
STX	0638	Store Index	(X) → A	D	2.76	ST

ARITHMETIC INSTRUCTIONS

ADC	0608	Add Carry	(A)+(CRO) → A; Carry → CRO; 1 → OVF, if overflow	E	1.20	ST
ADD	9XXX	Add to Accumulator	(A)+(Q) → A	B	1.84	ST
ADL	66XX	Add to A, Literal	$(INS)_7 \rightarrow INS_{0\text{-}6}$; (INS)+(A) → A	L	1.05	ST
ADLB	6EXX	Add to B, Literal	$(INS)_7 \rightarrow INS_{0\text{-}6}$; (INS)+(B) → B	L	1.05	ST
DAD	06A0	Double Add	(A,B)+(Q,Q+1) → A,B If overflow, 1 → OFV If carryout, 1 → CRO	D	3.91	DP
DDV	06B8	Double Divide	(A,B) ÷ (Q,Q+1) → A,B If overflow, 1 → OVF; 0 → CRO	D	33.54-39.19	DP
DFA	06C8	Double Floating Add	(A,B,E)+(Q,Q+1,Q+2) → A,B,E + denotes floating add	D	8.03-8.58 ‡¤	DFP
DFD	06F8	Double Floating Divide	(A,B,E) ÷ (Q,Q+1,Q+2) → A,B,E ÷ denotes floating divide	D	35.26-40.91 ¤	DFP
DFM	06E8	Double Floating Multiply	(A,B,E) X (Q,Q+1,Q+2) → A,B,E X denotes floating multiply	D	29.61-29.91 ¤	DFP
DFS	06D8	Double Floating Subtract	(A,B,E) - (Q,Q+1,Q+2) → A,B,E - denotes floating subtract	D	8.03-8.58 ‡¤	DFP
DIV	0708	Divide	(B,A)/(Q) → A; remainder → B	D	7.29-7.94	MD
DMP	06B0	Double Multiply	(A,B,) X (Q,Q+1) → A,B If overflow, 1→ OVF; 0 → CRO	D	27.84-28.14	DP
DSB	06A8	Double Subtract	(A,B) - (Q,Q+1) → A,B If overflow, 1→ OVF; If carryout, 1 → CRO	D	3.91	DP
FAD	06C0	Floating Point Add	(A,B) + (Q,Q+1) → A,B + denotes floating add	D	9.11-12.01 ‡¤	FP
FDV	06F0	Floating Point Divide	(A,B) ÷ (Q,Q+1) → A,B ÷ denotes floating divide	D	38.14-46.09 ¤	FP

ARITHMETIC INSTRUCTIONS (CONTINUED)

MNE-MONIC	OP CODE	INSTRUCTION NAME	LOGICAL FUNCTION	FORMAT TYPE	TIME μsec	INSTRUC-TION SET
FMP	06E0	Floating Point Multiply	(A,B) X (Q,Q+1) → A,B X denotes floating multiply	D	30.59-33.19 ¤	FP
FSB	06D0	Floating Point Subtract	(A,B) - (Q,Q+1) → A,B - denotes floating subtract	D	9.81-13.06 ‡¤	FP

MIN	8XXX	Memory Increment Skip on Zero	(Q) + 1 → Q
			If (Q)=0, (L)+2 → L
			If (Q)≠0, (L)+1 → L
				B	2.14	ST

MPY	0700	Multiply	(A) X (Q) → B,A	D	6.84-8.29	MD
SUB	DXXX	Subtract from Accumulator	(A) - (Q) → A	B	1.84	ST

LOGICAL INSTRUCTIONS

AAB	0600	AND Accumulator With B Register	(A) ∩ (B) → A	E	1.10	ST
AND	FXXX	AND with Accumulator	(A) ∩ (Q) → A	B	1.84	ST
ANL	60XX	AND The Accumulator, Literal	$(INS)_7 \rightarrow INS_{0-6}$; (INS) ∩ (A) → A	L	1.05	ST
ANLB	68XX	AND the B register, Literal	$(INS)_7 \rightarrow INS_{0-6}$; (INS) ∩ (B) → B	L	1.05	ST
AOB	0610	OR Accumulator With B register	(A) ∪ (B) → A	E	1.10	ST
XOL	62XX	Exclusive OR the Accumulator, Literal	$(INS)_7 \rightarrow INS_{0-6}$; (INS) ⊕ (A) → A	L	1.05	ST
XOLB	6AXX	Exclusive OR the B register Literal	$(INS)_7 \rightarrow INS_{0-6}$; (INS) ⊕ (B) → B	L	1.05	ST
XOR	BXXX	Exclusive OR With Accumulator	(A) ⊕ (Q) → A	B	1.84	ST

REGISTER MANIPULATION

ESA	0620	Extend Sign of Accumulator	$(A)_0 \rightarrow B_{0-15}$	E	1.00	ST
XAX	0618	Exchange Accumulator and Index	(A) → X; (X) → A	E	1.10	ST
XBX	0670	Exchange B register and Index	(B) → X; (X) → B	E	1.10	ST

OPERATE GROUP

RADD	Register Add	S + D → D		1.45
RCMP	Register Complement	$\overline{S}$ → D		1.25
RCPY	Register Copy	S → D		1.05
RDEC	Register Decrement	S - 1 → D		1.25
RINC	Register Increment	S + 1 → D		1.10
RNEG	Register Negate	$\overline{S}$ + 1 → D		1.25
RSUB	Register Subtract	S - D → D		1.45
RXOR	Register Exclusive OR	S ⊕ D → D		1.45

‡ +.25N for alignment

¤ +.55N for normalize

SHIFT INSTRUCTIONS

LAL	0260	Long Arithmetic, Left Shift	$(A)_i \rightarrow A_{i-1}$, $(B)_i \rightarrow B_{i-1}$, $0<i<16$ $(B)_0 \rightarrow A_{15}$; $0 \rightarrow B_{15}$ OVF Set if sign changes	S	1.38+.23N	ST
LAR	0220	Long Arithmetic, Right Shift	$(A)_i \rightarrow A_{i+1}$, $(B)_i \rightarrow B_{i+1}$, for $0<i<16$ $(A)_0 \rightarrow A_1$; $(A)_{15} \rightarrow B_0$	S	1.38+.23N	ST
LCL	03E0	Long Circulate, Left	$(A)_i \rightarrow A_{i-1}$, $(B)_i \rightarrow B_{i-1}$, $0<i<16$ $(CRO) \rightarrow B_{15}$; $(B)_0 \rightarrow A_{15}$; $(A)_0 \rightarrow CRO$	S	1.38+.23N	ST
LCR	03A0	Long Circulate, Right	$(A)_i \rightarrow A_{i+1}$, $(B)_i \rightarrow B_{i+1}$, $0<i<15$ $(CRO) \rightarrow A_0$; $(A)_{15} \rightarrow B_0$; $(B)_0 \rightarrow CRO$	S	1.38+.23N	ST
LLL	02E0	Long Logical, Left Shift	$(A)_i \rightarrow A_{i-1}$, $(B)_i \rightarrow B_{i-1}$; $0<i<16$ $(B)_0 \rightarrow A_{15}$; $0 \rightarrow B_{15}$	S	.23N - 2.07	ST
LLO	0628	Locate Leading Ones	If A = 0, (L) + 1 → L; If A ≠ 0, $(A)_i \rightarrow A_{i-1}$ until $(A)_0 = 1$. (X) + 1 → X for each position shifted. (L) + 2 → L when $(A)_0 = 1$; and $0 \rightarrow A_0$	E	1.05 (min.) 1.65 + .20N (max.)	ST
LLR	02A0	Long Logical, Right Shift	$(A)_i \rightarrow A_{i+1}$, $(B)_i \rightarrow B_{i+1}$, $0<i<15$ $0 \rightarrow A_0$; $(A)_{15} \rightarrow (B)_0$	S	.23N - 2.07	ST
LRL	0360	Long Rotate Left	$(A)_i \rightarrow A_{i-1}$, $(B)_i \rightarrow B_{i-1}$, $0<i<16$ $(B)_0 \rightarrow A_{15}$; $(A)_0 \rightarrow B_{15}$	S	.23N - 2.07	ST
LRR	0320	Long Rotate Right	$(A)_i \rightarrow A_{i+1}$, $(B)_i \rightarrow B_{i+1}$, for $0<i<15$ $(B)_{15} \rightarrow A_0$; $(A)_{15} \rightarrow B_0$	S	.23N - 2.07	ST
NDX	0710	Normalize and Decrement X	If A and B = 0; set X = 0 and (L) + 1 → L. If A or B ≠ 0; then, $A_i \rightarrow A_{i-1}$; $B_i \rightarrow B_{i-1}$; $B_0 \rightarrow A_{15}$ until $(A)_0 = 1$. (X) - 1 → X for each position shifted; and (L) + 1 → L.	E	1.50 (min.) A ≠ 0, 1.45 + .45N (max.) A = 0, 1.70 + .45N (max.)	ST
SAL	0240	Short Arithmetic, Left Shift	$(A)_i \rightarrow A_{i-1}$ for $0<i<16$; $0 \rightarrow A_{15}$. OVF set if sign changes	S	1.38+.23N	ST
SAR	0200	Short Arithmetic Right Shift	$(A)_i \rightarrow A_{i+1}$ for $0<i<15$; $(A)_0 \rightarrow A_1$	S	1.38+.23N	ST
SCL	03C0	Short Circulate, Left	$(A)_i \rightarrow A_{i-1}$, $0<i<16$; $(A)_0 \rightarrow CRO$; $(CRO) \rightarrow A_{15}$	S	1.38+.23N	ST
SCR	0380	Short Circulate, Right	$(A)_i \rightarrow A_{i+1}$, $0<i<15$ $(A)_{15} \rightarrow CRO$; $(CRO) \rightarrow A_0$	S	1.38+.23N	ST
SLL	02C0	Short Logical, Left Shift	$(A)_i \rightarrow A_{i-1}$, $0<i<16$; $0 \rightarrow A_{15}$	S	1.38+.23N	ST
SLR	0280	Short Logical, Right Shift	$(A)_i \rightarrow A_{i+1}$, $0<i<15$; $0 \rightarrow A_0$	S	1.38+.23N	ST
SRL	0340	Short Rotate Left	$(A)_i \rightarrow A_{i-1}$ for $0<i<16$; $(A)_0 \rightarrow A_{15}$	S	1.38+.23N	ST
SRR	0300	Short Rotate Right	$(A) \rightarrow A_{i+1}$ for $0<i<15$; $(A)_{15} \rightarrow A_0$	S	1.38+.23N	ST

JUMP AND SKIP INSTRUCTIONS

COT	0650	Carry Out Test	(L) + 1 → L if CRO = 1 (L) + 2 → L if CRO = 0; 0 → CRO	E	1.15	ST
JMP	4XXX	Jump	Q → L	B	.92	ST
JRT	0630	Jump Return	(Q) → L	D	2.76	ST
JSL	AXXX	Jump and Store Location	(L) + 1 → (Q); Q + 1 → L	B	2.76	ST
OFT	0658	Overflow Test	(L) + 1 → L, 0 → OVF, if OVF = 1 (L) + 2 → L, if OVF = 0	E	1.15	ST

JUMP AND SKIP INSTRUCTIONS (CONTINUED)

SKN	EXXX	Skip if A ≠ to Memory	(L) + 1 → L, If (A) = (Q) (L) + 2 → L, If (A) ≠ (Q)	B	1.84	ST

CONTROL INSTRUCTIONS

ARM	0180	Arm Interrupts	Arm selected interrupts	D	1.84	ST
CLI	0100	Clear Interrupt	Clear Current Priority Interrupt	E	.92	ST
DIS	00C0	Disable Interrupts	Disable Interrupt System	E	.92	ST
ENA	0080	Enable Interrupts	Enable Interrupt System	E	.92	ST
HLT	0000	Halt		E		ST
SCO	0660	Set Carry Out	1 → CRO	E	.95	ST
SOF	0668	Set Overflow	1 → OVF	E	.95	ST

SYSTEM INSTRUCTIONS

SRT	0140	System Return	(Q) → L (Q + 1) → Status (Q + 2) → X (Q + 3) → A (Q + 4) → B	D	6.44	ST
SYCL	0740	System Call	(L) → ((BA + INS_{10-15})) (ST) → ((BA + INS_{10-15})) + 1 (X) → ((BA + INS_{10-15})) + 2 (A) → ((BA + INS_{10-15})) + 3 (B) → ((BA + INS_{10-15})) + 4 0 → CRO, OVF (BA + INS_{10-15}) + 1 → L	SCL	7.66	ST

TSL	0040	Test and Set Lock	If (Q) = 0, (L) + 1 → L
			IF (Q) ≠ 0, (L) + 2 → L
			0 → Q
				D	3.06	ST

I/O INSTRUCTIONS

ACT	01D0	Activate Parallel I/O		D	1.84	ST
IOC	01C0	Input/Output Control		D	3.19 - 3.44	ST
RDP	01F0	Read Parallel	(DEVICE) → A, L+2 if device ready	E	1.80	ST
WTP	01E0	Write Parallel	(A) → (DEVICE), L+2 if device ready	E	1.80	ST

CHANNEL COMMANDS

EOA	Execute Order in A	CC		ST
HIO	Halt I/O	CC		ST
IIU	Input Interrupt Unit	CC		ST
IUS	Input Unit Status	CC		ST
OUS	Output Unit Status	CC		ST
SDA	Skip If Device Available	CC		ST
SDR	Skip if Device Ready	CC		ST
SIO	Start I/O	CC		ST
TWC	Terminate when complete	CC		ST
XMT	Transmit Characters	CC		ST

NUMERIC MNEMONIC LIST

HEXADECIMAL CODE	MNEMONIC
0000	HLT
0040	TSL
0080	ENA
00C0	DIS
0100	CLI
0140	SRT
0180	ARM
01C0	IOC
01D0	ACT
01E0	WTP
01F0	RDP
0200	SAR
0220	LAR
0240	SAL
0260	LAL
0280	SLR
02A0	LLR
02C0	SLL
02E0	LLL
0300	SRR
0320	LRR
0340	SRL
0360	LRL
0380	SCR
03A0	LCR
03C0	SCL
03E0	LCL
0600	AAB
0608	ADC
0610	AOB
0618	XAX
0620	ESA
0628	LLO
0630	JRT
0638	STX
0640	LAS
0648	LDS
0650	COT
0658	OFT
0660	SCO
0668	SOF
0670	XBX

HEXADECIMAL CODE	MNEMONIC
0678	LDX
0680	LDD
0688	LDF
0690	STD
0698	STF
06A0	DAD
06A8	DSB
06B0	DMP
06B8	DDV
06C0	FAD
06C8	DFA
06D0	FSB
06D8	DFS
06E0	FMP
06E8	DFM
06F0	FDV
06F8	DFD
0700	MPY
0708	DIV
0710	NDX
0740	SYCL
1XXX	LDH
2XXX	LDB
3XXX	STB
4XXX	JMP
5XXX	STH
60XX	ANL
62XX	XOL
64XX	LDL
66XX	ADL
68XX	ANLB
6AXX	XOLB
6CXX	LDLB
6EXX	ADLB
7XXX	STA
8XXX	MIN
9XXX	ADD
AXXX	JSL
BXXX	XOR
CXXX	LDA
DXXX	SUB
EXXX	SKN
FXXX	AND

ALPHABETIC INSTRUCTION LIST

MNEMONIC	INSTRUCTION NAME	OPERATION CODE HEX.	OPERATION CODE BINARY	TIME MICRO-SECONDS Min.	TIME MICRO-SECONDS Max.
AAB	AND Accumulator & B Register	0600	0000011000000000	1.10	
ACT	Activate Channel	0100	0000000111010000		
ADC	Add Carry	0608	0000011000001000	1.20	
ADD	Add to Accumulator	9XXX	1001XXXXXXXXXXX	1.84	
ADL	Add to A, Literal	66XX	01100110XXXXXXXX	1.10	
ADLB	Add to B, Literal	6EXX	01101110XXXXXXXX	1.10	
AND	AND with Accumulator	FXXX	1111XXXXXXXXXXX	1.84	
ANL	AND the Accumulator, Literal	60XX	01100000XXXXXXXX	1.10	
ANLB	AND the B Register, Literal	68XX	01101000XXXXXXXX	1.10	
AOB	OR Accumulator with B Register	0610	0000011000010000	1.10	
ARM	Arm Interrupts	0180	0000000110000000	1.84	
CLI	Clear Interrupt	0100	0000000100000000	0.92	
COT	Carryout Test	0650	0000011001010000	1.15	
DAD	Double Add	06A0	0000011010100000	3.91	
DDV	Double Divide	06B8	0000011010111000	36.39 - 48.39	
DFA	Double Floating Add	06C8	0000011011001000	7.83 - 8.98‡	
DFD	Double Floating Divide	06F8	0000011011111000	38.21 - 43.11	
DFM	Double Floating Multiply	06E8	0000011011101000	29.91 - 31.26	
DFS	Double Floating Subtract	06D8	0000011011011000	7.83 - 8.98‡	
DIS	Disable Interrupts	00C0	0000000011000000	0.92	
DIV	Divide	0708	0000011100001000	8.14 - 8.99	
DMP	Double Multiply	06B0	0000011010110000	28.29 - 28.84	
DSB	Double Subtract	06A8	0000011010101000	3.91	
ENA	Enable Interrupts	0080	0000000010000000	0.92	
EOA	Execute Order in A	*	X0011XX0XXXXXXXX		
ESA	Extend Sign of Accumulator	0620	0000011000100000	1.00	
FAD	Floating Point Add	06C0	0000011011000000	9.11 - 12.01	
FDV	Floating Point Divide	06F0	0000011011110000	40.74 - 47.99	
FMP	Floating Point Multiply	06E0	0000011011100000	31.99 - 33.49	
FSB	Floating Point Subtract	06D0	0000011011010000	9.81 - 13.06	
HIO	Halt I/O	*	X0000XX0XXXXXXXX		
HLT	Halt	0000	0000000000000000		
IDN	Input Device Number	*	X0100XX0XXXXXXXX		
IIU	Input Interrupting Unit	*	X1011XX0XXXXXXXX		
IOC	I/O Control	01C0	0000000111000000		
IUS	Input Unit Status	*	X1100XX0XXXXXXXX		
JMP	Jump	4XXX	0100XXXXXXXXXXX	0.92	
JRT	Jump Return	0630	0000011000110000	2.76	
JSL	Jump and Store Location	AXXX	1010XXXXXXXXXXX	2.76	
LAL	Long Arithmetic Left Shift	0260	00000010011XXXXX	1.38 + .23N	
LAS	Load Accumulator from Switches	0640	0000011001000000	0.95	

* I/O Commands

‡ +.21N for alignment

¤ +.55N for normalize

MNEMONIC	INSTRUCTION NAME	OPERATION CODE HEX.	OPERATION CODE BINARY	TIME MICRO-SECONDS Min.	TIME MICRO-SECONDS Max.
LAR	Long Arithmetic Right Shift	0220	00000010001XXXXX	1.38 + .23N	
LCL	Long Circulate Left	03E0	00000011111XXXXX	1.38 + .23N	
LCR	Long Circulate Right	03A0	00000011101XXXXX	1.38 + .23N	
LDA	Load Accumulator	CXXX	1100XXXXXXXXXXXX	1.84	
LDB	Load B Register	2XXX	0010XXXXXXXXXXXX	1.84	
LDD	Load Double	0680	0000011010000000	3.56	
LDF	Load Floating	0688	0000011010001000	4.48	
LDH	Load Halfword	1XXX	0001XXXXXXXXXXXX	2.09	
LDL	Load A, Literal	64XX	01100100XXXXXXXX	0.92	
LDLB	Load B, Literal	6CXX	01101100XXXXXXXX	0.92	
LDS	Load Status A	0648	0000011001001000	0.95	
LDX	Load Index	0678	0000011001111000	2.76	
LLL	Long Logical Left Shift	02E0	00000010111XXXXX	.23N	-2.07
LLO	Locate Leading Ones	0628	0000011000101000	1.05	
LLR	Long Logical Right Shift	02A0	00000010101XXXXX	.23N	-2.07
LRL	Long Rotate Left	0360	00000011011XXXXX	.23N	-2.07
LRR	Long Rotate Right	0320	00000011001XXXXX	.23N	-2.07
MIN	Memory Increment & Skip on Zero	8XXX	1000XXXXXXXXXXXX	2.14	
MPY	Multiply	0700	0000011100000000	6.84	- 8.44
NDX	Normalize & Decrement Index	0710	0000011100010000	1.50	
OFT	Overflow Test	0658	0000011001011000	1.15	
OUS	Output Unit Status	*	X0101XX0XXXXXXXX		
RADD	Register Add	**		1.45	
RCMP	Register Complement	**		1.25	
RCPY	Register Copy	**		1.05	
RDEC	Register Decrement	**		1.25	
RDP	Read Parallel	01F0	0000000111110000		
RINC	Register Increment	**		1.10	
RNEG	Register Negate	**		1.25	
RSUB	Register Subtract	**		1.45	
RXOR	Register Exclusive OR	**		1.45	
SAL	Short Arithmetic Left Shift	0240	00000010010XXXXX	1.38 + .23N	
SAR	Short Arithmetic Right Shift	0200	00000010000XXXXX	1.38 + .23N	
SCL	Short Circulate Left	03C0	00000011110XXXXX	1.38 + .23N	
SCO	Set Carryout	0660	0000011000100000	0.95	
SCR	Short Circulate Right	0380	00000001100XXXXX	1.38 + .23N	
SDA	Skip if Device Available	*	X1010XX0XXXXXXXX		
SDR	Skip if Device Ready	*	X1001XX0XXXXXXXX		
SIO	Start I/O	*	X0001XX0XXXXXXXX		
SKN	Skip if A ≠ to Memory	EXXX	1110XXXXXXXXXXXX	1.84	
SLL	Short Logical Left Shift	02C0	00000010110XXXXX	1.38 + .23N	

* I/O Commands

** Operate Group

MNEMONIC	INSTRUCTION NAME	OPERATION CODE HEX.	OPERATION CODE BINARY	TIME MICRO-SECONDS Min.	TIME MICRO-SECONDS Max.
SLR	Short Logical Right Shift	0280	00000010100XXXXX	1.38 +	.23N
SOF	Set Overflow	0668	0000011001101000	0.95	
SRL	Short Rotate Left	0340	00000011010XXXXX	1.38 +	.23N
SRR	Short Rotate Right	0300	00000011000XXXXX	1.38 +	.23N
SRT	System Return	0140	0000000101000000	6.44	
STA	Store Accumulator	7XXX	0111XXXXXXXXXXXX	1.84	
STB	Store B Register	3XXX	0011XXXXXXXXXXXX	1.84	
STD	Store Double	0690	0000011010010000	3.68	
STF	Store Floating	0698	0000011010011000	4.60	
STH	Store Halfword	5XXX	0101000000000000	1.84	
STX	Store Index	0638	0000011000111000	2.76	
SUB	Subtract from Accumulator	DXXX	1101XXXXXXXXXXXX	1.84	
SYCL	System Call	0740	0000011101000000	7.66	
TSL	Test and Set Lock	0040	0000000001000000	3.06	
TWC	Terminate When Complete	*	X0111XX0XXXXXXXX		
WTP	Write Parallel	01EO	0000000111100000		
XAX	Exchange Accumulator & Index	0618	0000011000011000	1.10	
XBX	Exchange B and X	0670	0000011001110000	1.10	
XMT	Transmit	*	X0010XX0XXXXXXXX		
XOL	Exclusive OR the Accumulator, Literal	62XX	01100010XXXXXXXX	1.10	
XOLB	Exclusive OR the B Register, Literal	6AXX	01101010XXXXXXXX	1.10	
XOR	Exclusive OR with Accumulator	BXXX	1011XXXXXXXXXXXX	1.84	

* I/O Commands

All specifications subject to change.

Scientific Control Corporation

P.O. Box 34529 Dallas, Texas 75234 214—242-6561 TWX 910-860-5509

EASTERN REGION

4321 Hartwick Road, Suite 104
College Park, Maryland 20740
301-779-3330

7100 Baltimore Ave., Suite 105
College Park, Maryland 20740
301-779-2510

6990 Lake Ellenor Dr., Suite 112
Orlando, Florida 32809
305-855-5833

1222 Route 46, Suite 217
Parsippany, New Jersey 07054
201-335-3001

2024 Riverdale Street
West Springfield, Mass. 01089
413-781-0063

CENTRAL REGION

612 Exchange Bank Building
Dallas, Texas 75235
214-358-1331

400 Brookes Lane, Suite 125
Hazelwood, Missouri 63042
314-848-3500

3110 Southwest Freeway, Suite 12
Houston, Texas 77006
713-526-5721

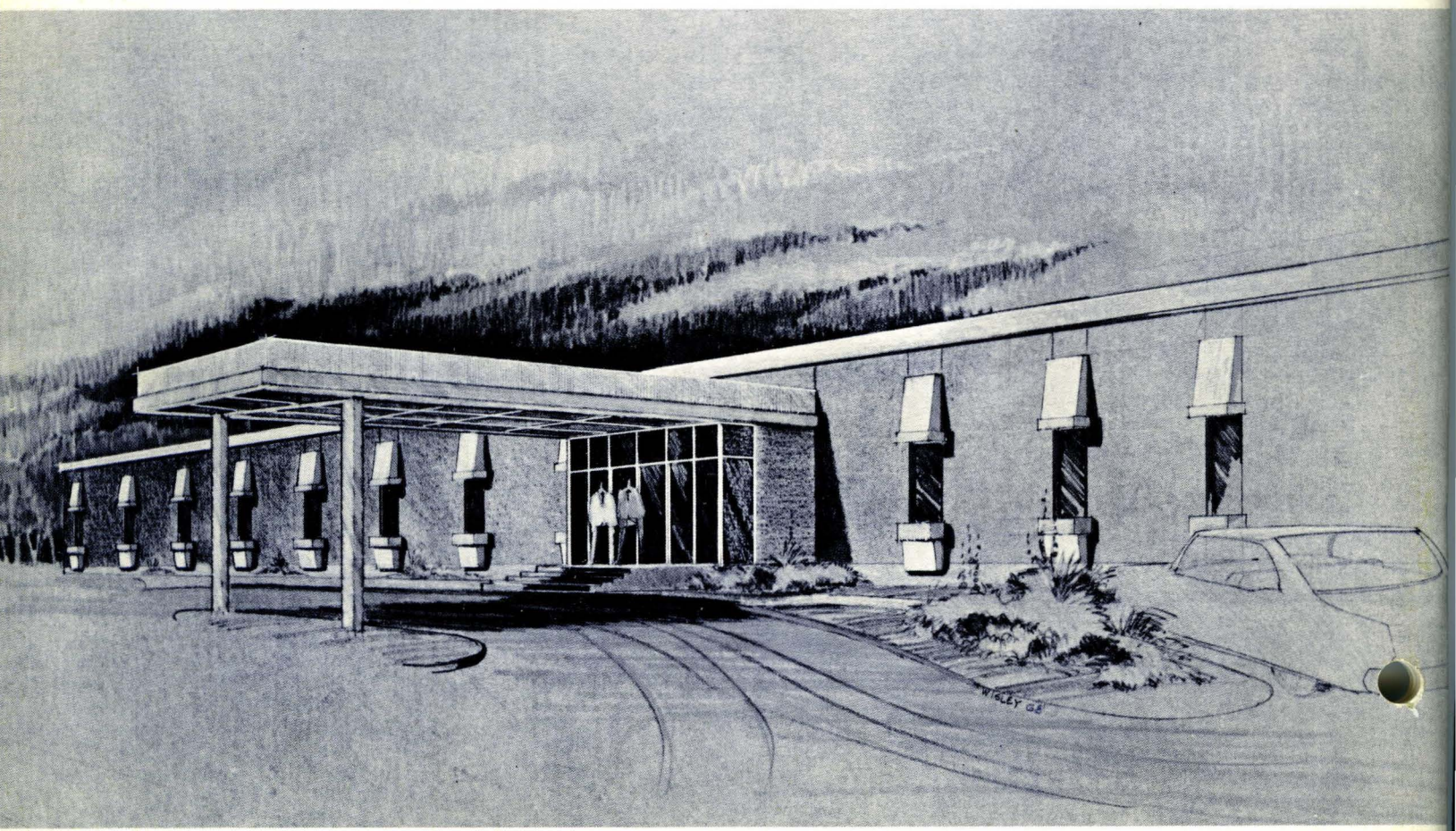
2510 Dempster Street, Suite 102
Des Plaines, Illinois 60016
312-297-2470

288 Clayton Street, Suite 204
Denver, Colorado 80206
303-322-0516

WESTERN REGION

780 Welch Road, Suite 208
Palo Alto, California 94304
415-328-8980

9550 Flair Drive, Suite 306
El Monte, California 91731
213-443-0143



SCC 52-11-26-81