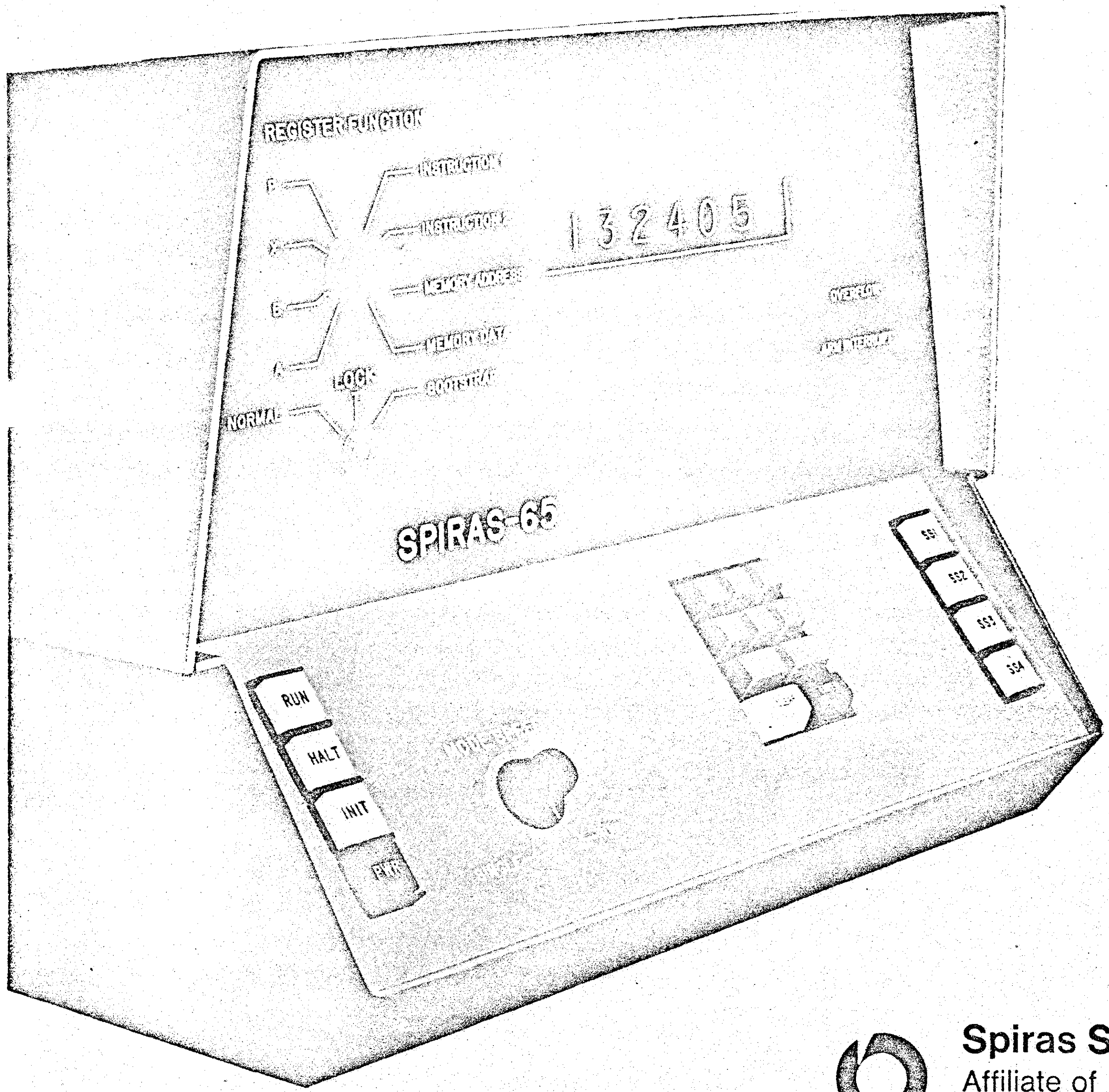
SPIRAS-65 REFERENCE MANUAL



Spiras Systems, Inc.
Affiliate of
USM Corporation

SPIRAS®-65 REFERENCE MANUAL

October 1969

Published by the Technical Communications Department, Spiras Systems, Inc.

Spiras Systems, Inc.
Affiliate of
USM Corporation

332 Second Avenue
Waltham, Massachusetts 02154
(617) 891-7300 • TWX 710-324-6699

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

SECTION 1

SPIRAS-65 ORGANIZATION

1.1 DESCRIPTION

The description of the SPIRAS-65 which follows does not reflect the actual hardware implementation but describes the computer from the programmers viewpoint.

A simplified block diagram of the SPIRAS-65 is shown in Figure 1-2. The computer consists of a control unit, arithmetic unit, core memory unit and the console which is attached to the I/O buss of the computer.

1.2 CONTROL UNIT

The control unit causes an instruction word to be fetched from core memory and deposited in the instruction register. The instruction is then decoded and executed unless the computer is in one of the following conditions:

- HALT mode
- single-step mode
- variable speed mode
- an interrupt has been requested
- a direct memory control data transfer has been requested

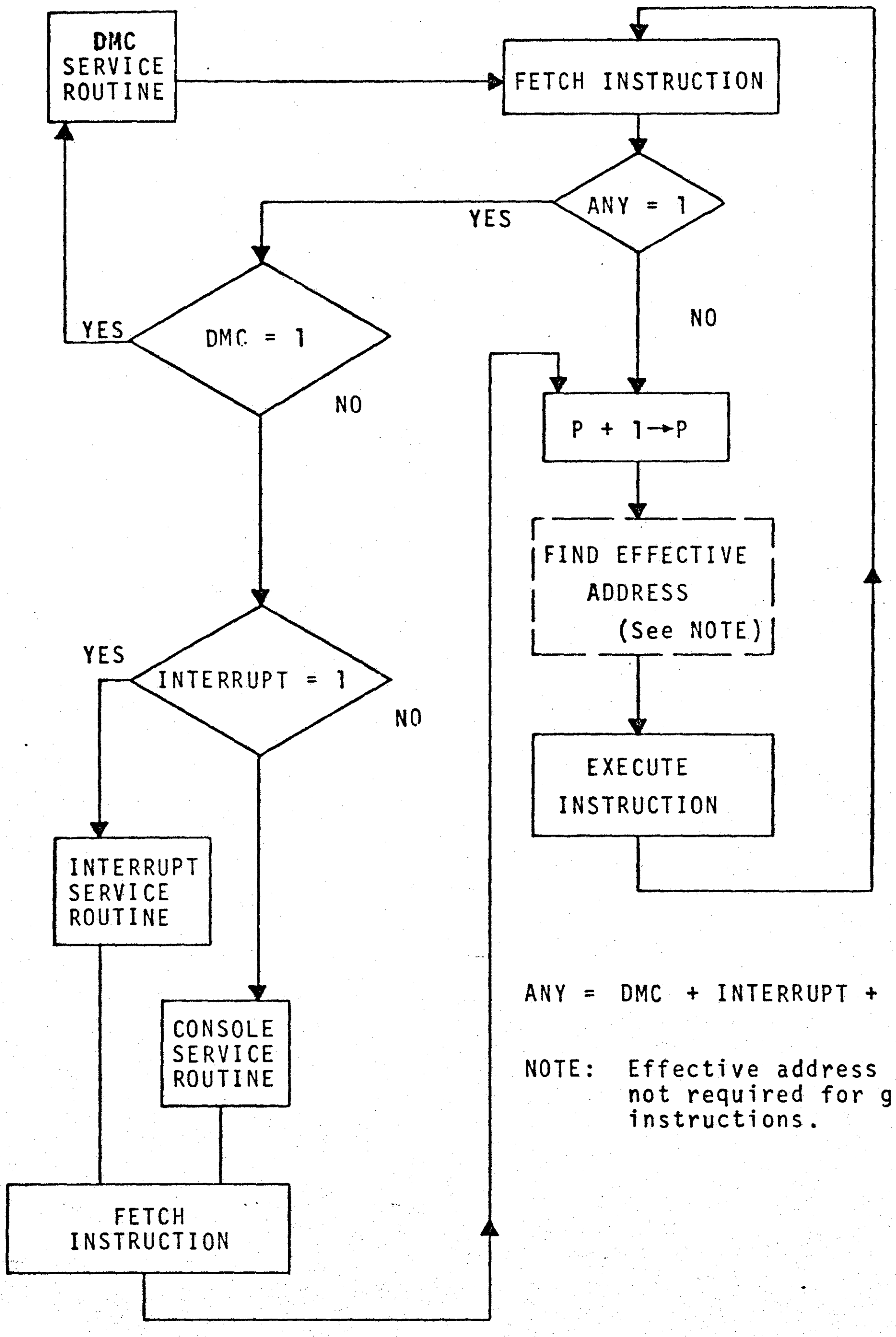
The basic control cycle is shown in Figure 1-1.

1.3 ARITHMETIC UNIT

The arithmetic unit consists of the A register (upper accumulator), the B register (lower accumulator) which is used in the multiplication and division operations, the X register (an index register), the P register (indicating the address of the next instruction) and the adder unit which can perform the operations of ADD, AND, OR, EXCLUSIVE OR, Left Shift, and Right Shift.

1.4 CORE MEMORY

The core memory unit is used for the storage of programs and data. It is modular in 4096 word increments to a maximum of 65,536 words. If a non-implemented memory location is addressed, no operation results and an all zero word will be read.



```
ANY = DMC + INTERRUPT + HALT

NOTE:  Effective address computation
       not required for generic
       instructions.
```

Figure 1-1. Basic Control Cycle

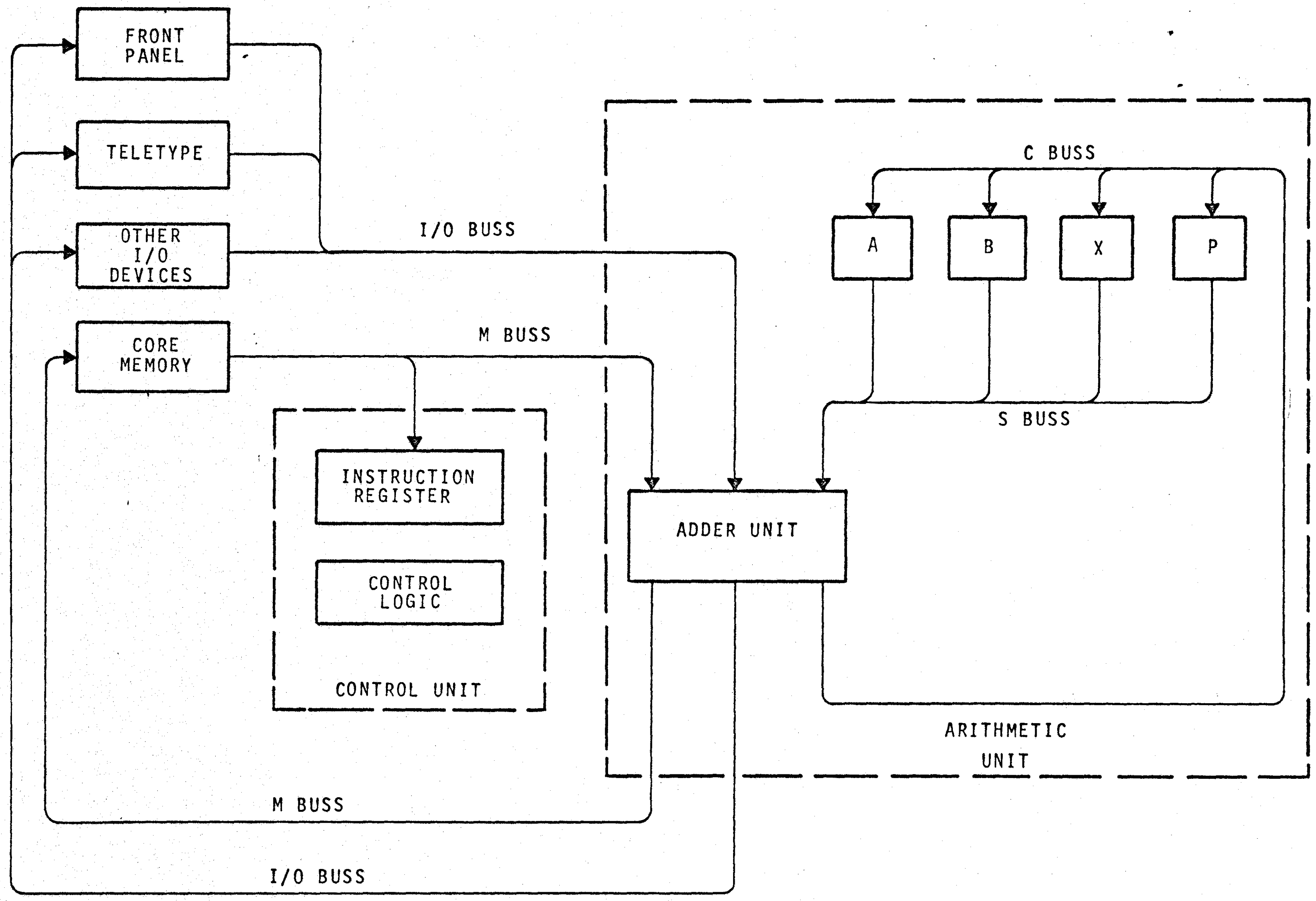


Figure 1-2. SPIRAS-65 Organization

1.5 INPUT/OUTPUT

Input/output devices attached to the SPIRAS-65 share a party-line I/O buss. Only one device can communicate with the computer at any given instant. An attempt to sense or input data from a non-existent device will cause an all zero word to be transferred. An attempt to output data, or a control word to a non-existent device results in no-operation.

SECTION 2

CONSOLE OPERATION

The console controls and their functions are listed in Appendix E. The reader should familiarize himself with these before proceeding.

2.1 POWER-ON AND BOOTSTRAP SEQUENCE

1) Depress POWER
2) Turn key switch to NORMAL
3) Depress INITIALIZE
4) Set MODE SPEED fully clockwise
5) Select P on REGISTER FUNCTION
6) Depress CLEAR
7) Key in location where loading is to begin
8) Depress ENTER
9) Select X
10) Depress CLEAR
11) Key in bootstrap device number

2	=	ASR
3	=	Card Reader
4	=	High Speed Paper Tape Reader
11	=	Magnetic Tape Unit #1
12	=	Magnetic Tape Unit #2
13	=	Magnetic Tape Unit #3
14	=	Magnetic Tape Unit #4

12) Depress ENTER
13) Select B (If relocatable program)

14) Key in BIAS

15) Depress ENTER

16) Select BOOTSTRAP

17) Depress INIT

18) Select NORMAL

The bootstrap micro-program will start the medium (except ASR), ignore leading zero bytes, assemble two bytes per word starting at the location determined by P until an all zero sixteen-bit word is loaded. Control is now transferred to the location determined by P. The micro-program computes a check-sum which is the arithmetic sum of all sixteen bit words loaded (overflow is ignored). The secondary bootstrap program (program which was loaded) should examine this check-sum.

2.2 REGISTER DISPLAY

The contents of the A,B,X, P registers may be displayed in the HALT, SINGLE STEP, or VARIABLE SPEED modes by selecting the desired register on the REGISTER FUNCTION switch. The contents of the selected register may be modified by depressing CLEAR, keying in new data, and depressing ENTER. The REGISTER FUNCTION switch is not functional when the processor is in the RUN mode.

The key switch must be in NORMAL position.

2.3 DISPLAYING MEMORY (HALT MODE)

To display the contents of memory starting at location XXX.

1) Select NORMAL

2) Depress HALT, INIT

3) Select MEMORY ADDRESS

4) Depress CLEAR

5) Key in XXX

6) Depress ENTER

7) Select MEMORY DATA
(The contents of location XXX is now displayed)

8) Depress ENTER

The contents of location XXX + 1 is now displayed. One can select MEMORY ADDRESS which will now show XXX + 1.

2.4 DISPLAY AND WRITE MEMORY (HALT MODE)

To display the contents of Location XXX and then write YYY into location XXX:

1) Select NORMAL

2) Depress HALT, INIT

3) Select MEMORY ADDRESS

4) Depress CLEAR

5) Key in XXX

6) Depress ENTER

7) Select MEMORY DATA

8) Depress CLEAR

9) Key in YYY

10) Depress ENTER

The contents of XXX is now YYY and the contents of XXX + 1 is displayed. If it is desired to modify location XXX + 1 repeat the process or examine XXX + 2 by depressing ENTER. The DISPLAY and DISPLAY/WRITE operations may be intermixed.

2.5 SINGLE STEP DEBUGGING

To single step through a program after it has been loaded:

1) Select NORMAL

2) Depress HALT, INIT

3) Select SINGLE STEP

4) Select P

5) Depress CLEAR

6) Key in location for start of single step operation

7) Depress ENTER

8) Select INSTRUCTION 1

The instruction which will be executed upon depressing RUN is displayed. This instruction may be modified by depressing CLEAR, keying in a new instruction, and depressing ENTER.

9) Select INSTRUCTION 2

   If the instruction to be executed is a two word instruction, the second word is displayed here, otherwise the next sequential instruction is displayed. The contents can be modified.

10) Select and set up A, B, X as necessary

11) Depress RUN

   Examine register contents with the aid of the REGISTER FUNCTION switch to see the results of each step of program execution.

NOTE

If the Octal keyset and/or NIXIE display are used as programmed output devices, they will not operate properly in the SINGLE STEP and VARIABLE speed modes because these devices are used by the console service routine.

The instructions CALL/CALS, ARM/ARMF and DRM/DRMF always execute the next sequential instruction before control passes to DMC, interrupt or console service routines: therefore, if a LDAS instruction is located at location Ø2Ø1, and the P counter is set to Ø1ØØ, and location Ø1ØØ contains a CALS Ø200 instruction, the P counter will show 2Ø2 after RUN is depressed in the SINGLE STEP mode.

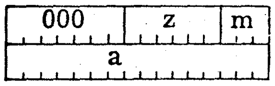
SECTION 3

INSTRUCTION FORMATS AND ADDRESSING MODES

The instructions in the SPIRAS-65 may be sixteen or thirty-two bits in length. Thirty-two bit instructions are stored in two consecutive memory locations with the first sixteen bits stored in the lower memory location.

A portion of the instruction set is implemented in both short (16 bit) and normal (32 bit) forms. This feature saves core locations when the referenced data is within addressing range and stores the full address or data with the instruction when extended addressing is required.

3.1 LONG INSTRUCTION FORMAT



z is the operation code

m is the mode

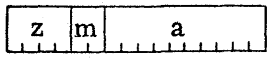
a is the address or operand

m	MODE	Effective Address/Operand	
0	Immediate	The operand is a.	
1	Direct	The address is a.	(e = a)
2	Indirect	The address is stored at a.	(e = (a))
3	Indirect pre-indexed with X	The address is stored at a plus contents of register X.	(e = (a + X))
4	Index with A	The address is a, plus contents of register A.	(e = a +A)
5	Indirect post-Indexed with X	The address is stored at a, plus contents of register X.	(e = (a) + X)
6	Index with X	The address is a, plus contents of register X.	(e = a + X)
7	Index with P	The address is a, plus contents of program counter P.	(e = a + P)
Multilevel indirect addressing is permitted.			

NOTE

The P register always points to the next instruction in sequence. The state of the P register must be taken into account when computing effective addresses.

3.2 SHORT INSTRUCTION FORMAT



z is the operation code

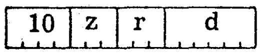
m is the addressing mode

a is the address

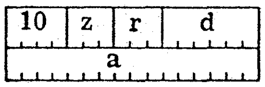
m	Octal	Addressing Mode	Effective Address
00	0,1	Direct	The address is a. Range of a is 0 to 1023_{10}. (e = a)
01	2,3	Indexed with X	The address is a, plus contents of register X. Range of a is 0 to 1023_{10}. (e = a + X)
10	4,5	Indirect	The address is stored at a. Range of a is 0 to 1023_{10}. (e = (a)).
11	6,7	Relative to P	The address is a, plus contents of program counter P. Range of a is $\pm 511_{10}$. (e = P $\pm$ a)

The m bits are combined with the most significant address bit in octal presentation.

3.3 INPUT/OUTPUT FORMAT



or

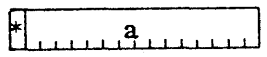


z is the operation code
r is the register mode
d is the device address
a is the memory address

Input/Output instructions are one word if there is no memory reference. Memory reference instructions require two words, the second of which is a sixteen-bit address.

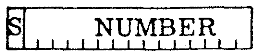
r	Mode	Effective Address
0	IMMEDIATE	The address is the address of the instruction plus one.
1	Register A	Register A
2	Register B	Register B
3	Register X	Register X
4	Direct	The address is a. (e = a)
5	Indexed with X	The address is a plus the contents of register X. (e = a + X)
6	Indirect	The address is stored at a. (e = (a))
7	Indirect post-indexed with X	The address which is stored at a is added to register X. (e = (a) + X)

3.4 INDIRECT ADDRESS FORMAT



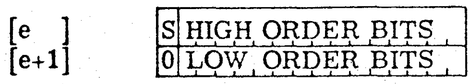
All forms of instructions requiring indirect address pointers use the indirect address format shown. A one in the sign bit position is used to indicate that another level of indirect addressing is to be involved.

3.5 SINGLE PRECISION FIXED POINT FORMAT



Single precision numbers consist of 15 bits plus the sign bit S. Negative numbers are represented in two's complement form.

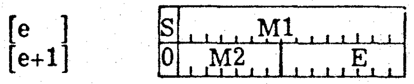
3.6 DOUBLE PRECISION FIXED POINT FORMAT



Double precision numbers consist of 30 bits plus the sign bit, S. The sign bit of the second word is always zero. Negative data is represented in two's complement form.

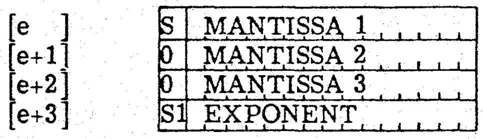
3.7 SINGLE PRECISION FLOATING POINT FORMAT

S	=	sign of mantissa
M1	=	high order part of mantissa
M2	=	low order part of mantissa
E	=	biased exponent ($+200_8$)



The mantissa consists of 22 bits plus the sign bit, S. The exponent consists of 8 bits with bit 7 serving as the sign. The sign bit of the second word is always zero. Negative data is represented in two's complement form. Adding 200_8 to the exponent results in an offset-by-2^8 notation, making the sign bit of the exponent "1" if the exponent is positive, and "0" if it is negative.

3.8 DOUBLE PRECISION FLOATING POINT FORMAT

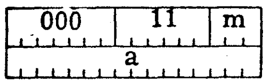


The mantissa consists of 45 bits plus the sign bit S. The exponent consists of 15 bits plus the sign bit S1. The sign bit of words 2 and 3 is always zero. Negative data is represented in two's complement form.

SECTION 4

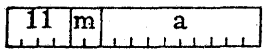
LOAD/STORE INSTRUCTIONS

LDA Load A register



Timing: 3 cycles

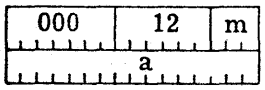
LDAS Load A register short form



Timing: 2 cycles

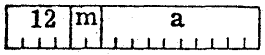
The contents of the effective memory location (operand) are placed in the A register.

LDB Load B register



Timing: 3 cycles

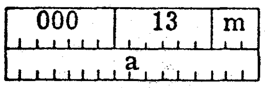
LDBS Load B register short form



Timing: 2 cycles

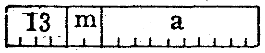
The contents of the effective memory location (operand) are placed in the B register.

LDX Load X register



Timing: 3 cycles

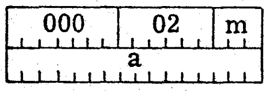
LDXS Load X register short form



Timing: 2 cycles

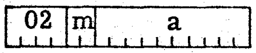
The contents of the effective memory location (operand) are placed in the X register.

STA Store register A



Timing: 3 cycles

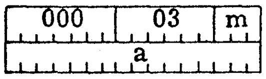
STAS Store register A short form



Timing: 2 cycles

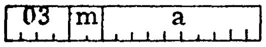
The contents of register A replace the contents of the effective memory location (operand).

STB Store register B



Timing: 3 cycles

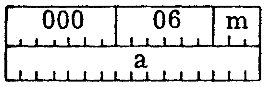
STBS Store register B short form



Timing: 2 cycles

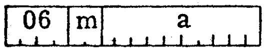
The contents of register B replace the contents of the effective memory location (operand).

STX Store register X



Timing: 3 cycles

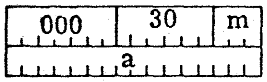
STXS Store register X short form



Timing: 2 cycles

The contents of register X replace the contents of the effective memory location (operand).

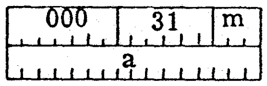
DLD Double Precision Load



Timing: 4 cycles

The double precision or floating point number contained in the two successive memory locations beginning with the effective memory location is placed in registers A and B with the most significant half in register A. The immediate addressing mode should not be used.

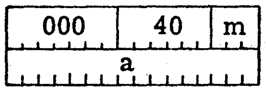
DST Double Precision Store



Timing: 5 cycles

The double precision or floating point number in registers A and B is placed in the two successive memory locations beginning with the effective memory location. The immediate addressing mode should not be used.

LEA Load Effective Address



Timing: 4 cycles*

The effective address is resolved taking into account indexing and all levels of indirect, and this address replaces the contents of the X register.

Typical use would be to fetch the argument address for a subroutine such as in the following example:

```
CALS    SUBR              SUBR  DATA    0
PTR     A                        .
PTR*    B                        .
  .                              .
  .                              .
  .                             LDXS  SUBR
                                LEA*  0(X)
                                 .
                                 .
                                 .
                                 .
                                LEA*  1(X)
                                 .
```

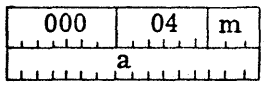
Another use of the LEA instruction is when it is necessary to set the index register to an address within a program that is to be "self-relative." A LDXI TABLE instruction would set the register correctly but would not be a self-relative instruction. A LEA TABLE(P) instruction would also set the register as desired but would also be self-relative.

* Includes the first indirect cycle.

SECTION 5

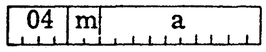
ARITHMETIC INSTRUCTIONS

<u>ADD</u> Add to register A



Timing: 3 cycles

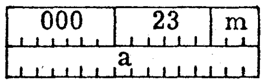
<u>ADDS</u> Add to register A short form



Timing: 2 cycles

The contents of the effective memory location (operand) are added to the contents of the A register. The sum, mod 2^{15} is placed in the A register. If the sum is $\geq 2^{15}$ or $<-2^{15}$ the overflow flag is set. Otherwise it is reset.

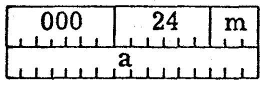
<u>ADB</u> Add to register B



Timing: 3 cycles

The contents of the effective memory location (operand) are added to the contents of register B. The sum, mod 2^{15}, is placed in register B. The overflow flag is not affected.

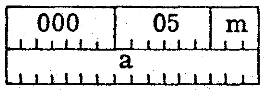
<u>ADX</u> Add to register X



Timing: 3 cycles

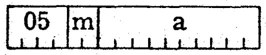
The contents of the effective memory location (operand) are added to the contents of register X. The sum, mod 2^{15}, is placed in register B. The overflow flag is not affected.

<u>SUB</u> Subtract from register A



3 cycles

<u>SUBS</u> Subtract from register A short form



Timing: 2 cycles

The contents of the effective memory location (operand) are subtracted from the contents of the A register. The difference, mod 2^{15}, is placed in the A register. If the difference is $\geq 2^{15}$ or $<-2^{15}$ the overflow flagis set. Otherwise it is reset.

MUL Multiply

000	011	m
a		

Timing: 11 cycles

MULS Multiply Short Form

011	m	a

Timing: 10 cycles

The contents of the effective memory location (operand) are multiplied by the contents of register B. The result is placed in registers A and B in double precision format, i.e., most significant half in register A, least significant half in register B and the sign bit of register B set to "0". The overflow flag is not affected. (NOTE: Multiplying -2^{15} by -2^{15} produces zero.)

DIV Divide

000	27	m
a		

Timing: 15 cycles

The contents of registers A and B (double precision format) are divided by the contents of the effective memory location (operand). The quotient is placed in register B and the remainder is placed in register A with the sign of the dividend. The overflow flag is set if A _ operand. An attempt to execute an improper divide will cause the overflow flag to be set and registers A and B to be unaltered.

For single precision fractional divide, the fractional dividend should be in the A register and the B register should be set to zero. For single precision integer divide, the integer dividend should be placed in the B register and the A register should be set to zero if the integer is positive and to all ones if the integer is negative. Integer division may be set up by loading the A register with the integer and performing an ASRD 15 instruction.

DADD Double Precision Add

000	32	m
	a	

Timing: 5 cycles

The double precision number contained in the two successive memory locations starting with the effective memory location is added to the double precision number in registers A and B. The sum, mod 2^{30} is placed in registers A and B in double precision format. The overflow flag is set if the sum is greater than full scale or less than minus full scale. The immediate addressing mode should not be used.

DSUB Double Precision Subtract

000	33	m
	a	

Timing: 6 cycles

The double precision number contained in the two successive memory locations starting with the effective memory location is subtracted from the double precision number in registers A and B. The difference, mod 2^{30}, is placed in registers A and B in double precision format. The overflow flag is set if the difference is $\geq -2^{30}$. The immediate addressing mode should not be used.

FADD Floating Point Add

000	34	m
	a	

Timing: 11-28 cycles and normalize time

The floating point number in the two successive memory locations starting with the effective address is added to the floating point number in registers A and B. The sum is placed in registers A and B in normalized form. If the sum is greater than full scale or less than minus full scale the overflow flag is set. A floating point number may be normalized by adding it to zero. The immediate addressing mode should not be used.

FSUB Floating Point Subtract

000	35	m
	a	

Timing: 11-28 cycles and normalize time

The floating point number in the two successive memory locations beginning with the effective address is subtracted from the floating point number in registers A and B. The difference is placed in registers A and B in normalized form. If the difference is greater than full scale or less than minus full scale the overflow flag is set. The immediate addressing mode should not be used.

FMUL Floating Point Multiply

000	36	m
	a	

Timing: 60-70 cycles

The floating point number in the two successive memory locations starting with the effective address is multiplied by the floating point number in registers A and B. The product is placed in registers A and B. If the product is greater than plus full scale or less than minus full scale the overflow flag is set. The immediate addressing mode should not be used.

FDIV Floating Point Divide

000	37	m
	a	

Timing: 60-70 cycles

The floating point number in the two successive memory locations starting with the effective address divides the floating point number in registers A and B. The quotient is placed in registers A and B. If the quotient is greater than full scale or less than minus full scale the overflow flag is set. An attempt to divide by zero will cause registers A and B to be set to plus or minus full scale and the overflow flag to be set. The immediate addressing mode should not be used.

Floating Point Normalize: See Floating Point Add

INR Increment and replace

000	26	m
	a	

Timing: 4 cycles

The contents of the effective memory location (operand) are incremented by one and replaced. The overflow is not affected.

DCR Decrement and replace

000	25	m
	a	

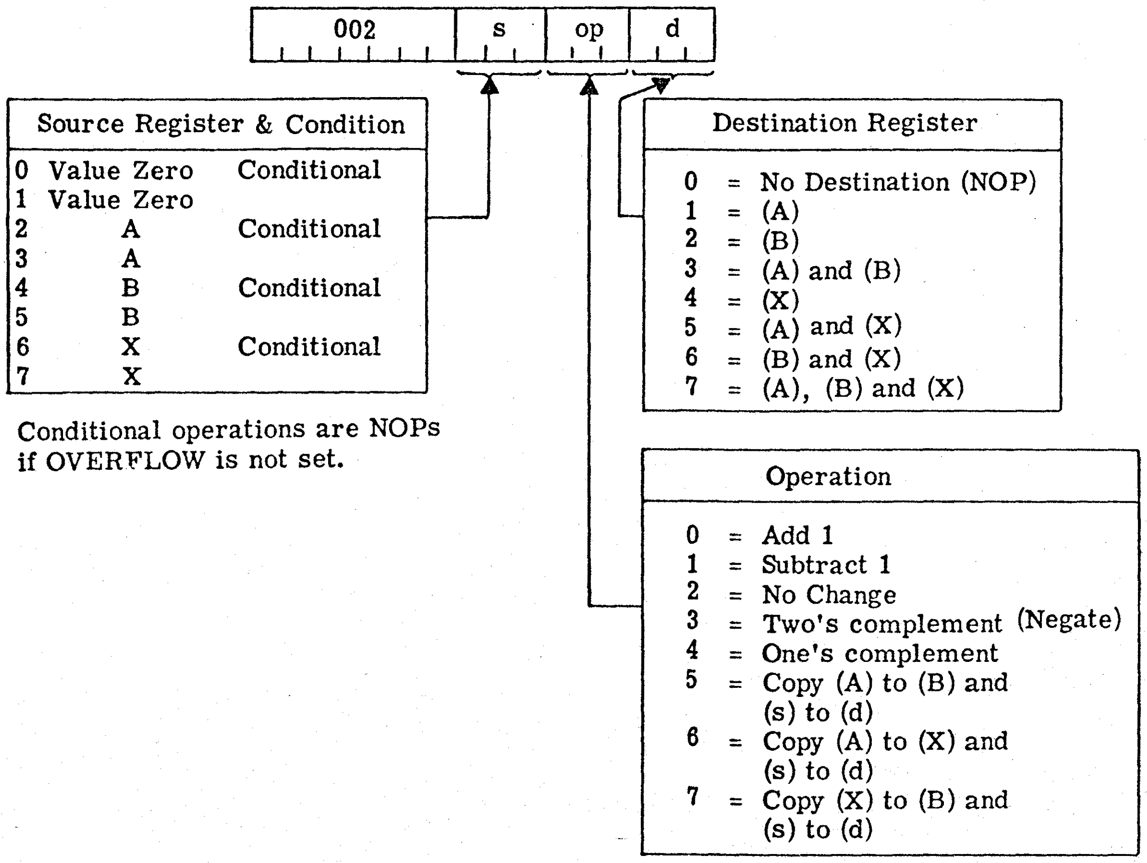
Timing: 4 cycles

The contents of the effective memory location (operand) are decremented by one and replaced. The overflow is not affected.

SECTION 6

REGISTER COPY INSTRUCTIONS

The format of the register copy instruction is as follows:



Source Register & Condition		
0	Value Zero	Conditional
1	Value Zero	
2	A	Conditional
3	A	
4	B	Conditional
5	B	
6	X	Conditional
7	X	

Conditional operations are NOPs if OVERFLOW is not set.

Destination Register		
0	=	No Destination (NOP)
1	=	(A)
2	=	(B)
3	=	(A) and (B)
4	=	(X)
5	=	(A) and (X)
6	=	(B) and (X)
7	=	(A), (B) and (X)

Operation		
0	=	Add 1
1	=	Subtract 1
2	=	No Change
3	=	Two's complement (Negate)
4	=	One's complement
5	=	Copy (A) to (B) and (s) to (d)
6	=	Copy (A) to (X) and (s) to (d)
7	=	Copy (X) to (B) and (s) to (d)

Thus, the instruction 002235 specifies that the contents of the source, register A, are negated and placed in registers A and X if the overflow flag is set.

There are 445 Register Change Instructions. Mnemonics for the more useful Register Change Instructions follow.

RGC	nnn	Copy operation depends on the value nnn.
CP	s,d	Copy
CPF	s,d	Copy if overflow is on
CPI	s,d	Copy and increment
CPIF	s,d	Copy and increment if overflow is on
CPD	s,d	Copy and decrement
CPDF	s,d	Copy and decrement if overflow is on
CPC	s,d	Copy and (one's) complement
CPCF	s,d	Copy and (one's) complement if overflow is on
CPN	s,d	Copy and negate (two's complement)
CPNF	s,d	Copy and negate if overflow is on
CAB	s,d	Simultaneously copy (A) to (B) and (s) to (d)
CABF	s,d	Same as CAB if overflow is on
CAX	s,d	Simultaneously copy (A) to (X) and (s) to (d)
CAXF	s,d	Same as CAX if overflow is on
CXB	s,d	Simultaneously copy (X) to (B) and (s) to (d)
CXBF	s,d	Same as CXB if overflow is on

NOTE

Conditional operations are NOP if the overflow flag is off.

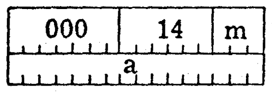
$$s = \left\{\begin{array}{l} \emptyset \\ A \\ B \\ X \end{array}\right. \qquad d = \left\{\begin{array}{ll} \emptyset & \\ A & \\ B & \\ X & \\ A,B & \text{(Add 0.2 cycles to time)} \\ A,X & \text{(Add 0.2 cycles to time)} \\ B,X & \text{(Add 0.2 cycles to time)} \\ A,B,X & \text{(Add 0.4 cycles to time)} \end{array}\right.$$

Timing: 1.4 cycles if unconditional
1.6 cycles if conditioned on overflow

SECTION 7

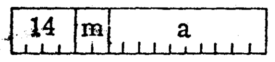
LOGICAL/CONTROL INSTRUCTIONS

XOR Exclusive OR with A



Timing: 3 cycles

XORS Exclusive OR with A short form

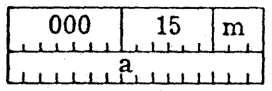


Timing: 2 cycles

A bit by bit exclusive OR is performed on the contents of register A and the contents of the effective memory location (operand). The result is placed in register A.

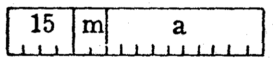
$(A)_i$	$(e)_i$	A_i XOR $(e)_i$
0	0	0
0	1	1
1	0	1
1	1	0

AND Logical AND with A



Timing: 3 cycles

ANDS Logical AND with A short form



Timing: 2 cycles

A bit by bit logical AND is performed on the contents of register A and the contents of the effective memory location (operand). The result is placed in register A.

$(A)_i$	$(e)_i$	A_i AND $(e)_i$
0	0	0
0	1	0
1	0	0
1	1	1

ORA Logical OR with A

000	16	m
a		

Timing: 3 cycles

ORAS OR with A short form

16	m	a

Timing: 2 cycles

A bit by bit logical OR is performed on the contents of register A and the contents of the effective memory location (operand). The result is placed in register A.

$(A)_i$	$[e]_i$	$(A)_i$ OR $[e]_i$
0	0	0
0	1	1
1	0	1
1	1	1

HLT Halt

000000

Computation is halted. When the RUN button is pressed after execution of a Halt instruction computation starts with the next instruction is sequence.

NOP No Operation

002000

Timing: 1.6 cycles

Execution of the No Operation instruction affects only the program counter P.

OVF Set Overflow

00174	n

Timing: 1.2 cycles

n = 0; Set overflow OFF

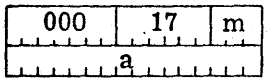
n = 1; Set overflow ON

TRAPPED INSTRUCTIONS: Instructions 00041n thru 00077n will be trapped if executed. These instructions may be used for simulation purposes under software control. Trap instructions will cause the instruction in location 0003 to be executed. This instruction is usually a CALS instruction. Indirect address chains of greater than 31 indirects, or system protect violations (when memory and instruction protect feature is implemented) will generate a trap causing the instruction in location 0002 to be executed.

SECTION 8

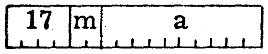
JMP/CALL INSTRUCTIONS

<u>JMP</u> JUMP Unconditionally



Timing: 3 cycles

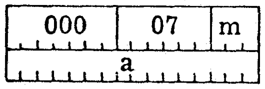
<u>JMPS</u> JUMP - Short form



Timing: 2 cycles

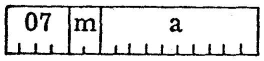
The next instruction executed is at the effective memory location.

<u>CALL</u> CALL Unconditionally



Timing: 3 cycles

<u>CALS</u> CALL - Short form



Timing: 2 cycles

The contents of the program counter P are incremented by one for the short form call and placed in the effective memory location. The contents of the program counter P are incremented by two for the long form call and placed in the effective memory location. The next instruction executed is at the effective memory location plus one.

(If an interrupt occurs during a CALL or CALS instruction, one additional instruction will be executed before the interrupt occurs. In the single step or variable speed modes the instruction following will be executed before halting again.)

SECTION 9

SKIP INSTRUCTIONS

9.1 SKIP ON CONDITION INSTRUCTIONS

The Skip instructions have the form:

00	z	CCC

z is the operation code
CCC is the condition code

z	Operation
4	Skip on any tested condition true
5	Skip on all tested conditions false

CCC	Condition Tested
000	Unconditional
001	Sense Switch 1
002	Sense Switch 2
004	Sense Switch 3
010	Sense Switch 4
020	Overflow Flag
040	Register A Positive
100	Register A Zero
200	Register B Zero
400	Register X Zero

NOTE

The skip instruction should always be followed by a single word instruction.

SKT Skip on any conditions true

00	4	CCC

Timing: 1 cycle

If any of the tested conditions is true skip one word, otherwise execute next sequential instruction.

SKF Skip on all conditions false

00	5	CCC

Timing: 1 cycle

If all of the tested conditions are false, skip one word, otherwise execute next sequential instruction.

The most common combinations of the CCC Field have been given mnemonic names as shown by the next 21 instructions.

SS1	Skip if Sense Switch 1 set	00 | 4 | 001	Timing: 1 cycle
SNS1	Skip if Sense Switch 1 not set	00 | 5 | 001	Timing: 1 cycle
SS2	Skip if Sense Switch 2 set	00 | 4 | 002	Timing: 1 cycle
SNS2	Skip if Sense Switch 2 not set	00 | 5 | 002	Timing: 1 cycle
SS3	Skip if Sense Switch 3 set	00 | 4 | 004	Timing : 1 cycle
SNS3	Skip if Sense Switch 3 not set	00 | 5 | 004	Timing: 1 cycle
SS4	Skip if Sense Switch 4 set	00 | 4 | 010	Timing: 1 cycle
SNS4	Skip if Sense Switch 4 not set	00 | 5 | 010	Timing: 1 cycle
SOF	Skip if overflow flag set	00 | 4 | 020	Timing: 1 cycle
SNOF	Skip if overflow flag not set	00 | 5 | 020	Timing: 1 cycle
SAZ	Skip if (A) = 0	00 | 4 | 100	Timing: 1 cycle
SANZ	Skip if (A) $\neq$ 0	00 | 5 | 100	Timing: 1 cycle

SAP	Skip if (A) $>$ 0	00	4	040
		Timing: 1 cycle		
SANP	Skip if (A) $\leq$ 0	00	5	040
		Timing: 1 cycle		
SAN	Skip if (A) $<$ 0	00	5	140
		Timing: 1 cycle		
SANN	Skip if (A) $\geq$ 0	00	4	140
		Timing: 1 cycle		
SBZ	Skip if (B) = 0	00	4	200
		Timing: 1 cycle		
SBNZ	Skip if (B) $\neq$ 0	00	5	200
		Timing: 1 cycle		
SXZ	Skip if (X) = 0	00	4	400
		Timing: 1 cycle		
SXNZ	Skip if (X) $\neq$ 0	00	5	400
		Timing: 1 cycle		
SKIP	Skip unconditionally	00	5	000
		Timing: 1 cycle		

9.2 COMPARE AND SKIP INSTRUCTIONS

CAS	Compare with A register and skip	000	20	m
		a		

If (A) $<$ [e]   execute next (short form) instruction

Timing: 3 cycles

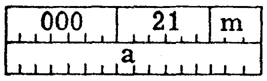
If (A) = [e]   skip next (short form) instruction

Timing: 3 cycles

If (A) $>$ [e]   skip next two (short form) instructions

Timing: 3 cycles

CBS Compare with B register and skip



If (B) < [e] Execute next (short form) instruction

Timing: 3 cycles

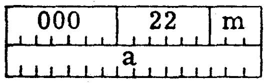
If (B) = [e] skip next (short form) instruction

Timing: 3 cycles

If (B) > [e] skip next two (short form) instruction

Timing: 3 cycles

CXS Compare with X and skip



If (X) < [e] execute next (short form) instruction

Timing: 3 cycles

If (X) = [e] skip next (short form) instruction

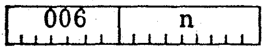
Timing: 3 cycles

If (X) > [e] skip next two (short form) instructions

Timing: 3 cycles

9.3 MODIFY AND SKIP INSTRUCTION

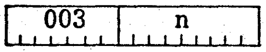
IXS Increment X and Skip if Zero



Timing: 1.4 cycles

The contents of register X are incremented by n (000_8 to 777_8) and replaced. If the result is not zero, the next instruction in sequence is executed. If the result is zero, the next instruction in sequence (should be a short form instruction) is skipped. The overflow is not affected.

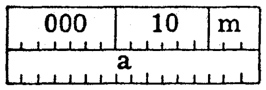
DXS Decrement X and Skip if Zero



Timing: 1.4 cycles

The contents of register X are decremented by the complement of n (000_8 to 777_8) and replaced. If the result is not zero, the next instruction is executed. If the result is zero, the next instruction in sequence (should be a short form instruction) is skipped. The overflow is not affected.

__DRS__ Decrement memory and Skip if Zero



Timing: 4 cycles

$[e] - 1 \rightarrow [e]$; then

If $[e] \neq 0$ execute next (short form) instruction

If $[e] = 0$ skip next (short form) instruction

SECTION 10

SHIFT INSTRUCTIONS

10.1 DIRECT SHIFT INSTRUCTION FORMAT

0	0	1	s + n

s specifies the kind of shift

n specifies the number of bit positions shifted (0 - 31)

10.2 INDEXED SHIFT INSTRUCTION FORMAT

0	0	7	s + n

The right nine bits of the index register are added to the right nine bits of the instruction. The sum of this addition determines the effective value of s and n.

10.3 INSTRUCTION SHIFT TYPES

s	Shift Type
ØØØ	Arithmetic shift left of A
Ø4Ø	Logical shift left of A
1ØØ	Arithmetic shift right of A
14Ø	Logical shift right of A
2ØØ	Arithmetic shift left of B
24Ø	Logical shift left of B
3ØØ	Arithmetic shift right of B
34Ø	Logical shift right of B
4ØØ	Arithmetic shift left of A, B
44Ø	Logical shift left of A, B
5ØØ	Arithmetic shift right of A, B
54Ø	Logical shift right of A, B
6ØØ	Logical rotate left of A
64Ø	Logical rotate left of B
7ØØ	Logical rotate left of A, B

10.4 SYMBOLIC SHIFT INSTRUCTIONS

The following descriptions specify the symbolic names accepted by the Assembler for the above types of shift. The variable field contains the value for n and may optionally be followed by an X within parentheses (indexed shift).

Examples:

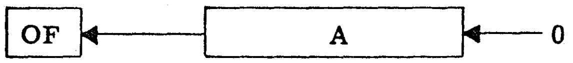
```
LSLA   6
LSLA   6(X)
```

LSLA Logical shift left of A

ØØ1	Ø4Ø+n

Timing: 1+.2 cycles

The contents of register A are shifted left n bit positions, where $0 \leq n \leq 37_8$. Zeros are shifted into the right of A. Bits shifted out of the sign bit of A set or reset the overflow flag.

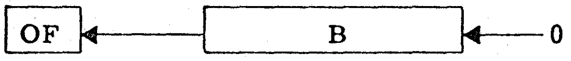


LSLB Logical shift left of B

ØØ1	24Ø+n

Timing: 1+.2n cycles

The contents of register B are shifted left n bit positions, where $0 \leq n \leq 37_8$. Zeros are shifted into the least significant bits of B. Bits shifted out of the sign bit of B set or reset the overflow flag.

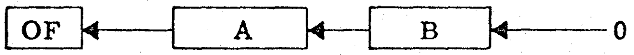


LSLD Logical shift left double

ØØ1	44Ø+n

Timing: 1+.4n cycles

The contents of registers A and B are shifted n bit positions, where $0 \leq n \leq 37_8$. The sign bit of the B register is shifted into the right of the A register. Zeros are shifted into the right of the B register. Bits shifted out of the sign position of A set or reset the overflow flag.

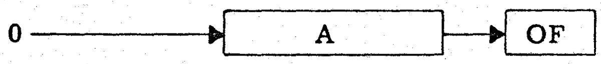


LSRA Logical shift right of A

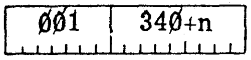
ØØ1	14Ø+n

Timing: 1+.2N cycles

The contents of register A are shifted right n bit positions, where $0 \leq n \leq 37_8$. Zeros are shifted into the sign bit of A and bits shifted out of the right of the A register set or reset the overflow flag.

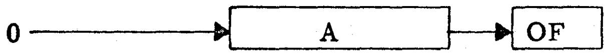


LSRB Logical shift right of B

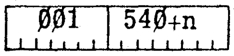


Timing: 1+.2n cycles

The contents of register B are shifted right n bit positions, where $0 \leq n \leq 37_8$. Zeros are shifted into the sign bit of B and bits shifted out of the right of the B register set of reset the overflow flag.

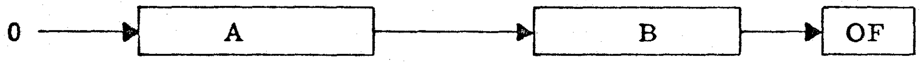


LSRD Logical shift right double

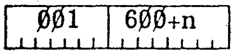


Timing: 1+.4n cycles

The contents of registers A and B are shifted right n bit positions where $0 \leq n \leq 37_8$. The rightmost bit of A is shifted into the sign bit of B. Zeros are shifted into the sign bit of A. Bits shifted out of the rightmost bit of B set or reset the overflow flag.

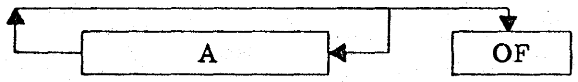


LRLA Logical rotate left of A

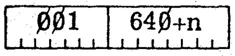


Timing: 1+.2n cycles

The contents of register A are rotated left n bit positions, where $0 \leq n \leq 37_8$. The sign bit of A is shifted into the right-most bit of the A register. The last bit shifted into the right-most bit of A sets or resets the overflow flag.

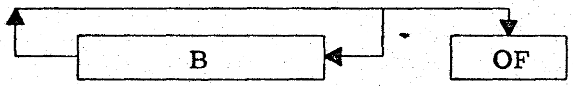


LRLB Logical rotate left of B



Timing: 1+.2n

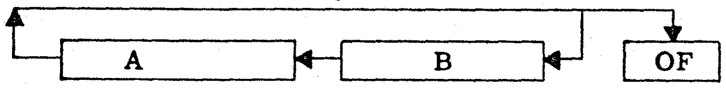
The contents of register B are rotated left n bit positions, where $0 \leq n \leq 37_8$. The sign bit of B is shifted into the right-most bit of B. The last bit shifted into the rightmost bit of B sets or resets the overflow flag.



LRLD Logical rotate left double | ØØ1 | 7ØØ+n |

Timing: 1+.6n cycles

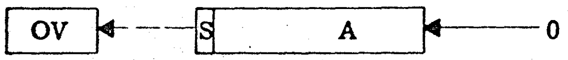
The contents of registers A and B are rotated left n bit positions, where $0 \leq n \leq 37_8$. The sign bit of B is shifted into the right most bit of A and the sign bit of A is shifted into the rightmost bit of B. The last bit shifted into the rightmost bit of B sets or resets the overflow flag.



ASLA Arithmetic shift left of A | ØØ1 | ØØØ+n |

Timing: 1+ .4n cycles

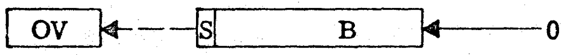
The contents of register A are shifted left n bit positions, where $0 \leq n \leq 37_8$. Zeros are shifted into the rightmost bit of A. The overflow flag is set and remains set if significant bits are lost, otherwise it is reset. The sign bit is unaltered.



ASLB Arithmetic shift left of B | ØØ1 | 2ØØ+n |

Timing: 1+ .4n cycles

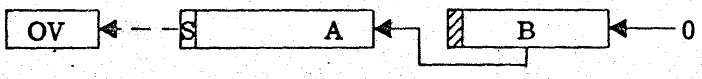
The contents of register B are shifted left n bit positions, where $0 \leq n \leq 37_8$. Zeros are shifted into the rightmost bits of B. The overflow flag is set and remains set if significant bits are lost. The sign bit is unaltered.



ASLD Arithmetic shift left double | ØØ1 | 4ØØ+n |

Timing: 1+.6n cycles

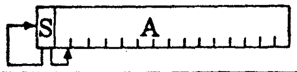
The contents of registers A and B are shifted left n bit positions, where $0 \leq n \leq 37_8$. The bit next to the sign bit in B is shifted into the right of A and zeros are shifted into the right of B. The overflow flag is set and remains set if the significant bits are lost, otherwise it is reset. The sign bit of the A register is unaltered and the sign bit of the B register is set to zero.



ASRA	Arithmetic shift right of A	ØØ1 | 1ØØ+n

Timing: 1+.2N cycles

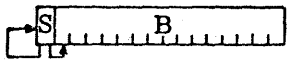
The contents of register A are shifted right n bit positions, where $0 \leq n \leq 37_8$. The sign bit of A is copied into the bit to the right of the sign. The overflow flag is reset.



ASRB	Arithmetic shift right of B	ØØ1 | 3ØØ+n

Timing: 1+.2n cycles

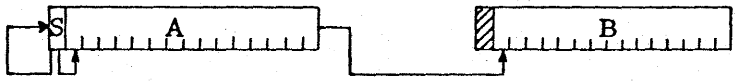
The contents of register B are shifted right n bit positions, where $0 \leq n \leq 37_8$. The sign bit of B is copied into the bit to the right of the sign bit. The overflow flag is reset.



ASRD	Arithmetic shift right double	ØØ1 | 5ØØ+n

Timing: 1+.4n cycles

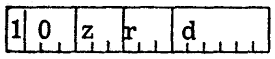
The contents of registers A and B are shifted right n bit positions, where $0 \leq n \leq 37_8$. The sign bits of A and B remain unchanged. The sign bit of A is copied into the bit to the right of the sign bit, and bits shifted from the right of A go into the bit to the right of the sign bit in B. The overflow flag is reset.



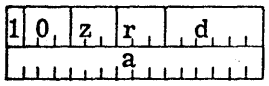
SECTION 11

INPUT/OUTPUT INSTRUCTIONS

Input and output instructions have format



or



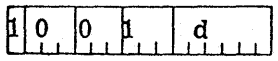
z specifies the operation

r specifies whether register A, register B, register X, memory or an immediate instruction is involved.

d is the device address

a is the address or operand if there is one. The address may be direct, indexed, indirect, or indirect post-indexed, depending on the r field.

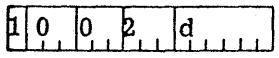
EXCA External control from A



Timing: 1 cycle

A 16-bit command word is sent to device d from register A.

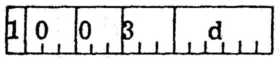
EXCB External control from B



Timing: 1 cycle

A 16-bit command word is sent to device d from register B.

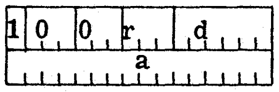
EXCX External control from X



Timing: 1 cycle

A 16-bit command word is sent to device d from register X.

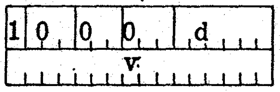
EXCM External control from memory



Timing: 2 cycles

A 16-bit command word is sent to device d from the effective memory location.

EXCI External control immediate



Timing: 2 cycles

The 16-bit command word v is sent to device d.

SENA Sense status to A

1	0	1	1	d

Timing: 1 cycle

The status bits for device d are placed in register A.

SENB Sense status to B

1	0	1	2	d

Timing: 1 cycle

The status bits for device d are place in register B.

SENX Sense status to X

1	0	1	3	d

Timing: 1 cycle

The status bits for device d are placed in register X.

SENM Sense status to memory

1	0	1	r	d
a				

Timing: 2 cycles

The status bits for device d are placed in the effective memory location.

SENS Sense status and skip if zero

1	0	1	0	d
MASK				

Timing: 2 cycles

If any of the masked status bits for device d is "1" the next instruction in sequence is executed. If not, the next instruction in sequence is skipped. The next instruction must be short form. (See Section 13 for Mask Descriptions.)

OTA Output from A

1	0	2	1	d

Timing: 1 cycle

The contents of register A are transferred to device d.

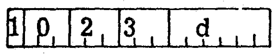
OTB Output from B

1	0	2	2	d

Timing: 1 cycle

The contents of register B are transferred to device d.

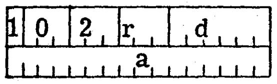
OTX Output from X



Timing: 1 cycle

The contents of register X are transferred to device d.

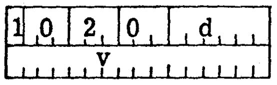
OTM Output from memory



Timing: 2 cycles

The contents of the effective memory location are transferred to device d.

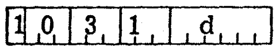
OTI Output Immediate



Timing: 2 cycles

The operand v is transferred to device d.

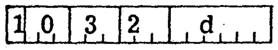
CIA Clear and input to A



Timing: 1 cycle

Register B is cleared and a data word from device d is transferred into register A.

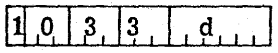
CIB Clear and input to B



Timing: 1 cycle

Register B is cleared and a data word from device d is tranferred into register B.

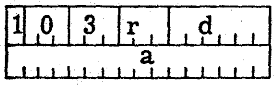
CIX Clear and input to X



Timing: 1 cycle

Register X is cleared and a data word from device d is transferred into register X.

CIM Clear and input to memory



Timing: 2 cycles

A data word from device d replaces the contents of the effective memory location.

INA	Input and OR with A	1 0 5 1 d
		Timing: 1 cycle

A logic OR is performed on the contents of register X and a data word from device d. The result is placed in register X.

INB	Input and OR with B	1 0 5 2 d
		Timing: 1 cycle

A logic OR is performed on the contents of register B and a data word from device d. The result is placed in register B.

INX	Input and OR with X	1 0 5 3 d
		Timing: 1 cycle

A logical OR is performed on the contents of register X and a data word from device d. The result is placed in register X.

ARM	Arm Interrupt	1 0 4 0 0 2
		Timing: 1 cycle

The Arm Interrrupt Flag is set and the Overflow Flag is reset. If an interrupt is waiting to be processed, one additional instruction following the ARM instruction is processed before the interrupt takes place.

ARMF	Arm Interrupt and Set Overflow	1 0 4 0 0 3
		Timing: 1 cycle

Same as ARM except the Overflow Flag is set on.

DRM	Disarm Interrupt	1 0 4 0 0 0
		Timing: 1 cycle

The Arm Interrupt Flag is reset causing interrupts to be held back by the computer until interrupts are allowed again. The Overflow Flag is reset.

DRMF	Disarm Interrupt and Set Overflow	1 0 4 0 0 1
		Timing: 1 cycle

Same as DRM escept the Overflow Flag is set on.

SECTION 12

INPUT/OUTPUT PROCEDURES

12.1 GENERAL PROCEDURES

There are five general procedures for performing I/O functions with the SPIRAS-65 computer.

12.1.1 Programmed Input/Output

In this procedure, a sense instruction is used to check the status of the device (busy, etc.), if necessary an External Control instruction is used to start a motion (start card reader, etc.) and an Input or Output instruction to input or output a data byte or word. A combination of Sense and Input or Output instructions must be executed for each character or word to be processed in this manner.

12.1.2 Interrupted Input/Output

Rather than waiting for the device to be non-busy or periodically checking I/O status as was done in Procedure 1, Interrupts can be armed and requested such that after the Input or Output of a data byte or word is started, no additional checking is required. When the specified I/O action is completed, the computer program is interrupted, an interrupt handling program initiates the next I/O action, and the interrupted program is resumed.

12.1.3 Direct Controlled Input/Output and Interrupt

The procedure requires that two control words be set up for the I/O device being used (DMC words). The first word specifies the location of the first data word transfer. The second word specifies the location of the last word to be processed. After arming interrupts, enabling the DMC and initiating the first input/output action, the computer will automatically fetch or store additional data from the specified memory area until all data is processed. At that time, an interrupt occurs informing the computer that the data block has been processed.

The table of DMC word pairs are in fixed locations starting at twice the Device Number. For the standard devices, these locations are:

Teletype (Device 2)	0004, 0005
Card Reader (Device 3)	0006, 0007
Paper Tape (Device 4)	0010, 0011
Line Printer (Device 5)	0012, 0013

If useful, the value of the two words can be modified during the I/O process, thereby extending or changing the memory area being processed.

12.1.4 Direct Controlled Input/Output Without Interrupt

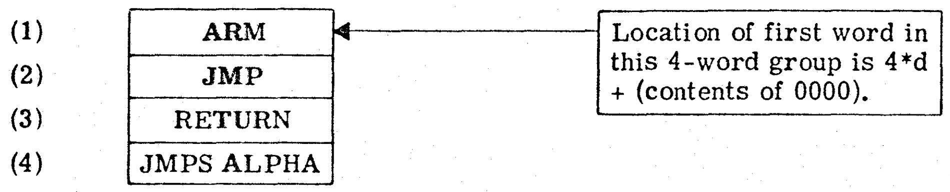
The only difference between this and the previous procedure is that interrupts are not requested.

12.1.4.1 DMA Option

The previous procedures are all possible using a basic SPIRAS-65 Computer. As an additional option, any device can also be equipped with two DMA registers. These registers allow an operation identical to paragraph 11.1.3 except that no memory accesses are required between data words thereby increasing the potential I/O transfer rate.

12.1.5 Input/Output Interrupts

Location 00000 points to the location of an interrupt table. This table consists of four words for each device number and each four word group is used as follows:


(1) ARM
(2) JMP
(3) RETURN
(4) JMPS ALPHA
Location of first word in this 4-word group is 4*d + (contents of 0000).

The computer will set word 1 to an ARM or an ARMF instruction depending if the overflow was set at the time of the interrupt. Word 3 is set to the next location to be executed after interrupt processing is completed. The (short form) instruction in word 4 is then executed.

When an I/O device generates an interrupt signal and interrupts are armed;

1) Interrupts will be disarmed.

2) The program being executed is interrupted after the current instruction is complete. (CALL, CALS, and ARM instructions will execute one additional instruction before interrupting.)

3) The current value of the P register is saved in word 3 of the interrupt table 4-word group for the interrupting device, and an ARM or ARMF instruction is constructed and placed into word 1 of the 4-word group.

4) The instruction in word 4 of the 4-word group is executed.

The instruction executed is probably a Jump to an interrupt handling program which will save any necessary registers, do whatever processing is necessary to service the interrupt, reset the registers (which also resets the register comparison status flags) and jump to word 1 of the Interrupt Table 4-word group. That, in turn, will ARM the interrupts (as well as reset the original overflow status) and do a long jump back to the proper place in the interrupted program.

The location of the 4-word groups depend on the contents of word 0000 (=BASE) and the interrupting device number. For the standard I/O devices, the following addresses are applicable.

	4-WORD GROUP LOC.
Teletype (Device 2)	BASE+0010
Card Reader (Device 3)	BASE+0014
Paper Tape (Device 4)	BASE+0020
Line Printer (Device 5)	BASE+0024

12.2 RESERVED LOCATIONS IN CORE

0000	Location of Interrupt Table
0001	(Used by Floating Point Logic)
0002	Executed when indirect address trap occurs.
0003	Executed by 00041X-00077X Instructions
0004	Device 02 DMC (Teletype)
0005	
0006	Device 03 DMC (Card Reader)
0007	
0010	Device 04 DMC (Paper Tape)
0011	
0012	Device 05 DMC (Line Printer)
0013	
0014	Device 06 DMC
0015	
.	.
.	.
.	.
0176	Device 077 DMC
0177	

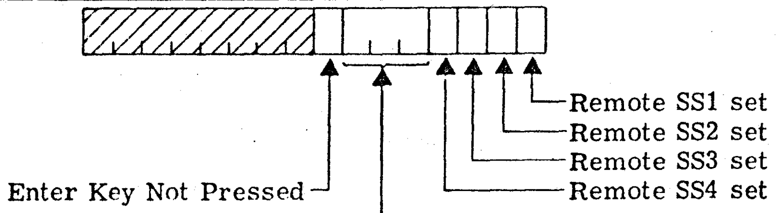
Those words in the above table for which a DMC Device is not attached to the system, may be used for any other purposes. Most standard software packages (Assembler, Fortran, etc.) utilize the memory area starting at location 0074. (In addition, the memory area between 0000 and 0073 is often used for bootstrap loading purposes.)

SECTION 13

I/O STATUS AND CONTROL WORD FORMATS

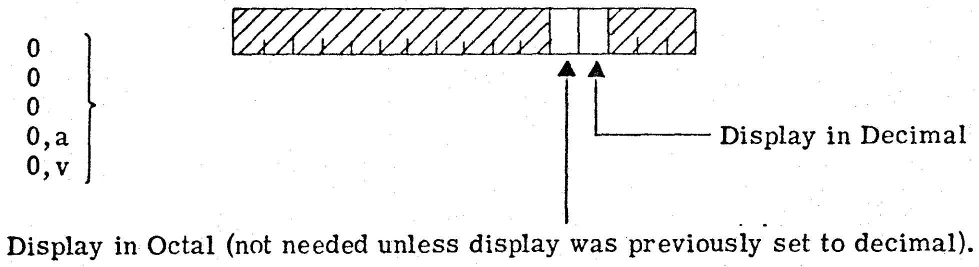
13.1 CONSOLE INPUT/OUTPUT

SENS	0,m
SENA	0
SENB	0
SENX	0
SENM	0,a

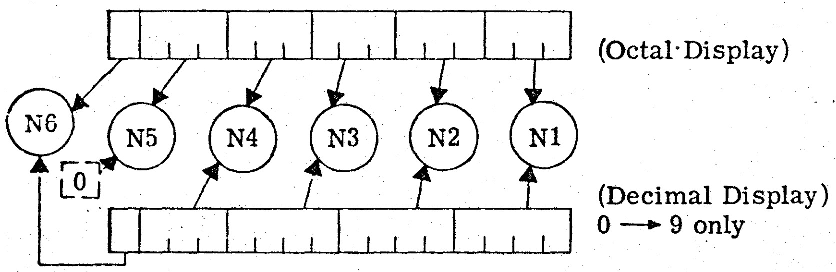


Position of Register Display Switch		
000	=	Memory Data Position
001	=	P Register Position
010	=	X Register Position
011	=	B Register Position
100	=	A Register Position
101	=	Instruction 1 Position
110	=	Instruction 2 Position
111	=	Memory Address Position

EXCA	0
EXCB	0
EXCX	0
EXCM	0,a
EXCI	0,v

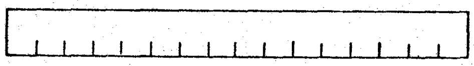

Display in Decimal
Display in Octal (not needed unless display was previously set to decimal).

OTA	0
OTB	0
OTX	0
OTM	0,a
OTI	0,v



Information is displayed only if the computer is in run mode.

INA	0
INB	0
INX	0
CIA	0
CIB	0
CIX	0
CIM	0,a

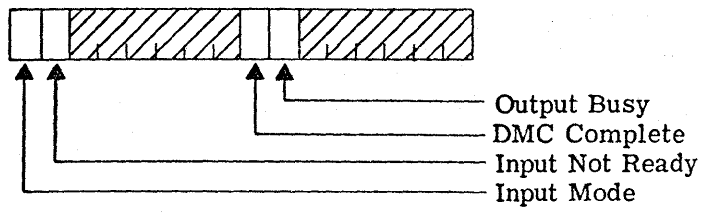


Information currently in the "NIXIE" display buffer is input as 16 bits of data.

13.2 TELETYPE INPUT/OUTPUT

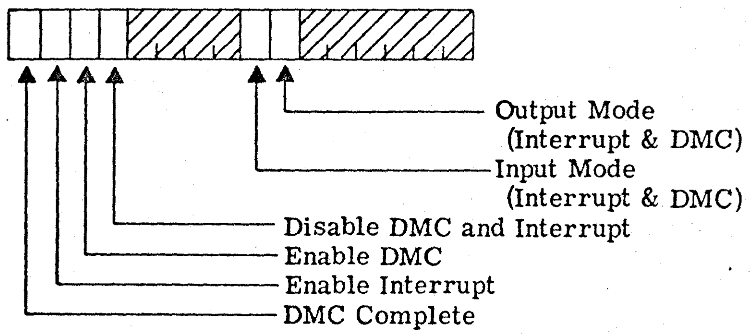
---SENSE---

SENS	2, m
SENA	2
SENB	2
SENX	2
SENM	2, a



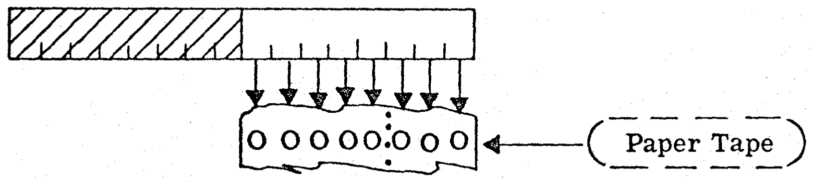
---CONTROL---

EXCA	2
EXCB	2
EXCX	2
EXCM	2, a
EXCI	2, v


Output Mode (Interrupt & DMC)
Input Mode (Interrupt & DMC)
Disable DMC and Interrupt
Enable DMC
Enable Interrupt
DMC Complete

---OUTPUT---

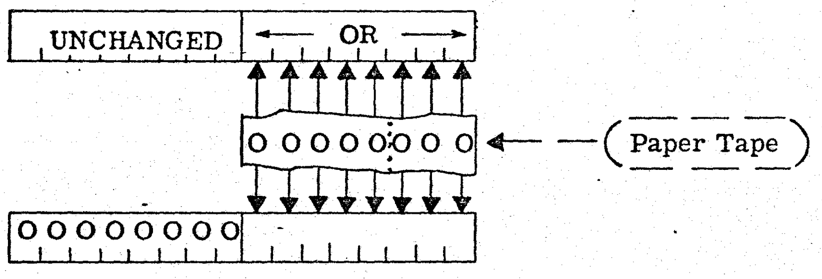
OTA	2
OTB	2
OTX	2
OTM	2, a
OTI	2, v



---INPUT---

INA	2
INB	2
INX	2

CIA	2
CIB	2
CIX	2
CIM	2, a



13.2.1 Teletype Programming Notes

The ASR 33/35 is operated in the full duplex mode on the SPIRAS-65 computer. Full duplex means that it is possible to simultaneously and asynchronously input (keyboard or reader) and output (page printer and punch). The teletypes used on the SPIRAS-65 feature an even parity coding. All SPIRAS-65 system software forces the eight bit to be a logical "1" inside the computer. Either code may be output with equal effectiveness.

The teletypes respond to the tape control characters as follows:

ASC II Code	FUNCTION
021,221	X-ON (Reader On)
022,222	TAPE (Punch On)
023,223	X-OFF (Reader Off)
024,224	~~TAPE~~ (Punch Off)

The punch-on code should always be followed by a RUBOUT (ASCII 377) or an equivalent amount of time before attempting to punch data or a synchronization problem will develop.

There is no method of inhibiting printing on the SPIRAS-65 teletypes. When it is desired to punch without printing; the 4x4 format, which derives its name from the fact that each sixteen bit computer word is represented by four characters, should be used.

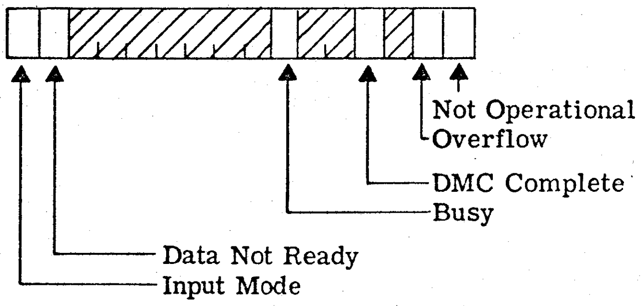
4 BIT CODE	PUNCH CHARACTER
0000	00010.000
0001	00000.001
0010	00000.010
0011	00000.011
0100	00000.100
0101	00010.101
0110	00010.110
0111	00010.111
1000	00011.000
1001	00011.000
1010	00011.010
1011	00011.011
1100	00011.100
1101	00011.101
1110	00011.110
1111	00011.111

When turning the reader on and off under program control it is necessary to allow two extra characters on the tape for every off-on cycle because the teletype does not stop "on character."

13.3 CARD READER INPUT

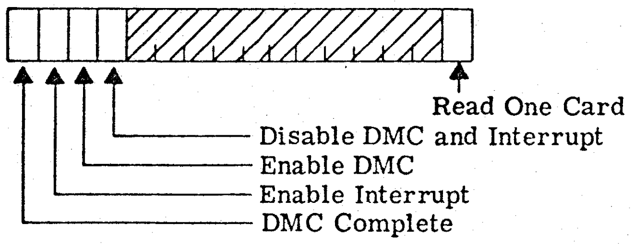
---SENSE---

SENS	3,m
SENA	3
SENB	3
SENX	3
SENM	3,a



---CONTROL---

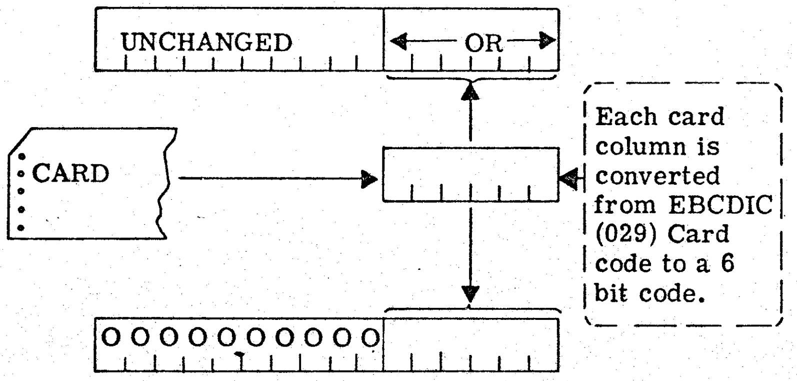
EXCA	3
EXCB	3
EXCX	3
EXCM	3,a
EXCI	3,v



---INPUT---

INA	3
INB	3
INX	3

CIA	3
CIB	3
CIX	3
CIM	3,a

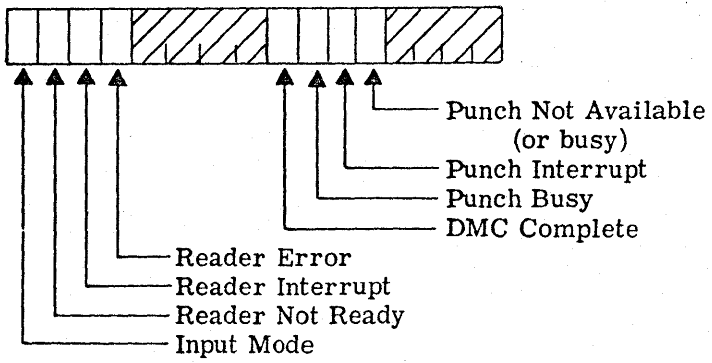


UNCHANGED
OR
CARD
Each card column is converted from EBCDIC (029) Card code to a 6 bit code.
O O O O O O O O O O

13.4 HIGH SPEED PAPER-TAPE INPUT/OUTPUT

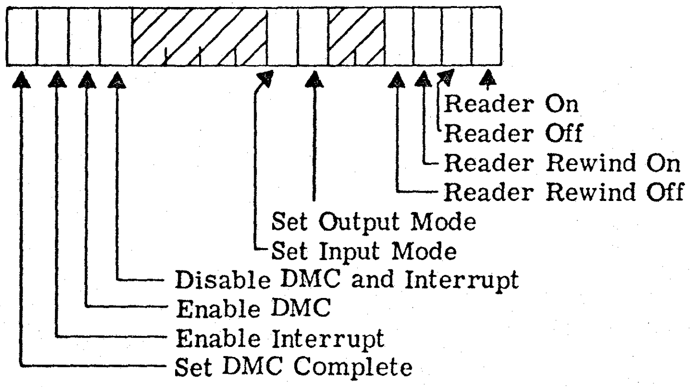
---SENSE---

SENS	4,m
SENA	4
SENB	4
SENX	4
SENM	4,a



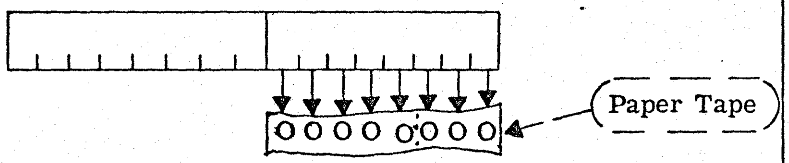
---CONTROL---

EXCA	4
EXCB	4
EXCX	4
EXCM	4,a
EXCI	4,v



---OUTPUT---

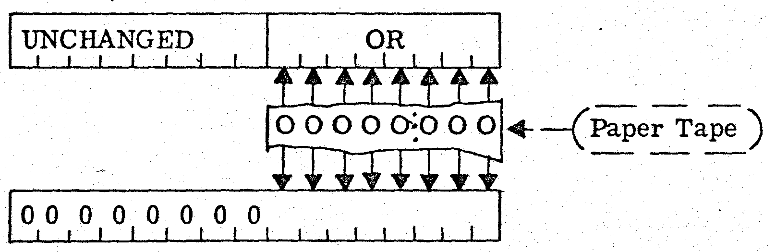
OTA	4
OTB	4
OTX	4
OTM	4,a
OTI	4,v



---INPUT---

INA	4
INB	4
INX	4

CIA	4
CIB	4
CIX	4
CIM	4,a



SECTION 14

SPIRAS-65 ASSEMBLER PROGRAM

14.1 PROGRAM TYPES

The Assembler Program is available in two versions; the primary version which requires a minimum of 8192 words of memory, and a basic version which operates within a 4096 word memory computer. The basic version is a compatible subset of the primary version without any macro or concordance capabilities.

Both versions of the Assembly Program operate under the SPIRAS-65 Operating System. This operating system is tailored to the configuration of the computer and performs all the standard I/O functions required by the Assembler Program, Fortran Compiler, etc.

Except where specified, the assembler characteristics described in the rest of this section apply to both the basic and primary versions of the Assembler.

14.2 ASSEMBLY FORMAT

For documentation purposes, a source statement normally positions its fields as follows:

Label Field-----------	Column 1
Command Field-------	Column 8
Argument Field-------	Column 16
Comments Field ------	Column 32

The assembler, however, actually allows source statements to be "free-form" using the following logical rules:

- A Label Field (if present) must start in Column 1.
- The Command Field starts with the first non-blank character following the Label Field.
- The Argument Field starts with the first non-blank character following the Command Field. If more than 10 blanks follow the Command Field, the Argument Field is presumed vacant.
- The Argument Field may consist of several arguments separated by a comma, a single space, or both. A double blank terminates the Argument Field.
- Any characters following the Argument Field (or following Column 72) are ignored except for listing, and can be used for comments. Teletype listings are terminated at Col. 50.
- An asterisk in Column 1 will cause the rest of that line to be considered as comments.

14.3 SYMBOLIC LABELS

Labels consists of a sequence of characters in which the first character is a letter, and the remaining characters are either letters, digits, dollar sign or the underline character. (It is suggested that the dollar sign ($) be reserved for use by system programs in order to avoid conflicts with system variables and subroutines.) Labels may be any length, but only the first 8 characters are retained by the assembler requiring that all labels be unique within the first 8 characters.

Examples

```
LAB7
VOLTMETER
X10031
F$31
MAX_SIZE
```

14.4 COMMANDS

The Command Field consists of any of the instruction mnemonics described in earlier sections of the manual, or pseudo-op mnemonics described later in this section.

When applicable, the mnemonic may be followed by the letter I if an immediate address is being specified, or by the asterisk character (*) if an indirect address is being specified.

If the Command Field consists of a constant, then this field is processed as if it were the argument Field of a DATA pseudo-operation.

Examples

```
LDA
LDAI
LDA*
CALS
PTR
DATA
0102511
```

14.5 ARGUMENTS

Arguments are made up of symbolic label operands (as described in paragraph 14.3), constant operands, or combinations of operands separated by operators. Tables 14-1 and 14-2 describe the various constant formats and the allowable operator types.

TABLE 14-1

ALLOWABLE CONSTANT TYPES

CONSTANT TYPE	EXAMPLES
Octal (Leading Zero)	Ø177777,Ø3, -Ø77
Integer (no decimal point)	123,32768,-50,+9
ASCII (2 char. max, stored right justified with a leading zero byte if necessary)	'AB', 'X', '12', 'Ø'
Single Precision Floating Point*	12.3, -6E5,.1,+9.,123.4E-5
Double Precision Floating Point*	12.3DØ,-6D5,.1DØ,+12.3D-5
Single Precision Fixed Point*	12.3B5,-6B+15,.1B-2,+1.5B2 12.3E2B10,-6E-10B-20
Double Precision Fixed Point*	12.3BB5,-6BB+15,.1BB-2 12.3D2BB10,-6E-10BB-20
* Not allowed by the basic version of the Assembly Program.	

TABLE 14-2

ALLOWABLE OPERATORS

ARITHMETIC (overflow ignored) +,-,*,/ LOGICAL (bit-by-bit logical operation) .AND.,.OR.,.XOR. RELATIONAL (result is 1 if true, Ø if false) .EQ.,.NE.,.GT.,.LT.,.LE.,.GE. SHIFT (logical shift) .LS.,.RS.
NOTE: The basic version of the Assembler Program allows only the operators + and -.

Operators are executed according to a priority value attached to each operator and according to the depth of parenthese nesting (operators within parentheses will be executed before any operator outside of parentheses). The priority value attached to each operator is shown in Table 14-3.

TABLE 14-3

OPERATOR PRIORITY VALUES

OPERATORS	PRIORITY VALUE
*,/	15+B
+,-	12+B
.EQ.,.NE.,.GT.	9+B
.GE.,.LT.,.LE.	9+B
.AND.	7+B
.OR.	6+B
.XOR.	5+B
.RS.,.LS.	3+B
(	B=B+20
)	B=B-20
(Terminators)	0

Operands are typed as values (or absolute addresses), as multiple-word data, as relative addresses, or as external addresses. Certain combinations of operators and operands are improper. Table 14-4 indicates which combinations are proper (Y=Yes, N=No).

14.6 ADDRESS MODIFIERS

Memory referencing instructions may wish to specify an address modifier (such as an index tag) in addition to the symbolic address. This is done by following the address with a register letter enclosed in parentheses. For example:

```
LDA      ALPHA(X)        Relative to X Register
LDA      A+B+C(A)        Relative to A Register
LDA      ALPHA-1(P)      Relative to P Register
LDA*     ALPHA           Indirect Address
LDA*     Ø12372(X)       Pre-Indexed Indirect Address
LDA*     ALPHA(Y)        Post-Indexed Indirect Address
```

TABLE 14-4

ALLOWABLE OPERATOR/OPERAND COMBINATIONS

		A2		
A1+A2		ABS.	REL.	EXT.
A1	ABS.	Y	Y	Y
	REL.	Y	N	N
	EXT.	Y	N	N
A1-A2				
A1	ABS.	Y	N	N
	REL.	Y	Y*	N
	EXT.	Y	N	Y**
A1 RELATIONAL A2				
A1	ABS.	Y	N	N
	REL.	N	Y	N
	EXT.	N	N	Y**
A1 OTHERS A2				
A1	ABS.	Y	N	N
	REL.	N	N	N
	EXT.	N	N	N

* Result is an absolute value.
** Operands must be in same external group.

```
LDAS*     ALPHA              Short-Form Indirect Address
LDAS      ALPHA+5 (X)        Short-Form Relative to X Register
LDAS      ALPHA (P)          Short-Form Relative to (Advanced) P Register
```

• Symbolic addresses with no modifiers are processed as follows:

1) If the address contains a reference to an externally defined variable, the instruction is passed on to the loader for resolution.

2) Otherwise, the instruction is made Direct (if possible).

3) Otherwise, the instruction is made relative-to-P if P is within range of the address.

4) Those short-form instructions that contain an address that is neither in range of P ($\pm 511_{10}$) or Direct ($0\text{-}1023_{10}$) will be passed on to the loader which will generate an indirect link to the address, modifying the instruction accordingly. (Indirect links are processed by the loader identically the same as literals.)

```
         ORG     Ø17ØØ
GAMMA    LDAS    ALPHA          =LDAS     *+Ø1ØØ(P)
         ORG     Ø2ØØØ
ALPHA    LDAS    Ø5ØØ           =LDAS     Ø5ØØ
         LDAS    ALPHA          =LDAS     *-1(P)
         LDAS    BETA           =LDAS     *+4(P)
         LDAS    GAMMA          =LDAS     Ø17ØØ
         LDAS    BETA (P)       =LDAS     *+2(P)
         LDAS    ALPHA (P)      =LDAS     *-5(P)
BETA     LDAS    Ø5ØØ (P)       (out of range)
         LDAS    GAMMA (P)      =LDAS     *-71(P)
         LDA     KAPPA          =LDA      Ø5ØØØ
         LDA     KAPPA(P)       =LDA      *+Ø2766(P)
KAPPA    EQU     Ø5ØØØ
```

14.7 LITERALS

A literal is any single or double word data value appearing in an argument field and is preceeded by an equals sign. The SPIRAS-65 Loader Program constructs a literal pool such that all identical literals within the group of programs being loaded will share the same location. Because the literal table starts at location 0100, literals may be referred to by either long or short instructions. The assembly program, therefore, does not construct its own literal table but only passes on the literal value to the Loader. Literals may, however, be constructed within the program by using the LIT pseudo-op (see paragraph 14.8.15).

Examples:

```
LDAS     =3
ADDS     ='X'
ANDS     =Ø377
DIV      =1.5B8          )
DLD      =1.234          )
FADD     =9876E5         }  Not in basic version
DADD     =0.0032BB-6     )   of the assembler.
```

14.8 PSEUDO-OPS

14.8.1 ORG

This pseudo-op specifies the location of the next data item generated by the assembler, and also determines the mode of this data (absolute or relative).

The variable field must contain one argument (or expression). If this argument specifies an absolute value, that value becomes the storage location for the next word generated, and the mode of all symbolic labels defined after this line (until another ORG statement is processed) are defined as absolute addresses. If the argument specifies a relative value, that relative location is used for subsequent data storage, and all symbolic labels defined after this line are defined as relative addresses.

Several ORG statements may be present in one program. If none is present, the assembler will presume an ORG to relative location zero has been specified.

Example:

```
ORG    *            Relative Zero Origin
ORG    Ø3ØØØ        Absolute Origin
ORG    *-5          Relative or Absolute (depending on previous mode)
ORG    START+2ØØ    Relative Origin (if START is relative)
```

14.8.2 BOOT

If this pseudo-op is present, it specifies the binary output of the assembler is to be in absolute bootstrap format. The ENT, EXT and COMN pseudo-ops cannot be present within a program containing a BOOT pseudo-op. The ORG pseudo-op may be used but only with absolute addresses. The assembler will presume the LIT pseudo-op has been specified.

14.8.3 EXT

The argument field of this pseudo-op contains the names of all symbolic labels whose location will be externally defined (library subroutines, etc.). Several EXT pseudo-ops may be used in one program if convenient.

Examples:

```
EXT    SIN,COS,SQRT,EXP
EXT    S$IO
```

14.8.4 COMN

The COMN pseudo-op creates storage areas which are to be shared by several sub-programs. The symbol appearing in the location field specifies the COMMON region (a BLANK location field specifies BLANK common). The variable field contains the list of variables (and their sizes) that are to be located within these common regions (assigned in the order of appearance within the variable field). Because of loader limitations, labels used in a COMN statement are limited to 6 characters.

```
       COMN    A1(5),A2(1)
STUDNT COMN    NAME(30),AVERAG(1)
```

14.8.5 ENT

If any of the symbolic labels defined within the program being assembled is to be referenced symbolically by some other program or subroutine, these labels must be specified in the argument field of the ENT pseudo-op. The ENT pseudo-op must preceed all other lines within the program except for comment lines, listing control lines or EXT pseudo-op lines.

Because of Loader symbol table format restrictions, only 1 to 6 character labels may be specified in the argument field of an ENT pseudo-op.

Examples:

```
ENT    POINT1,ENTRY2
ENT    ALT 2
```

14.8.6 EQU

The EQU pseudo-op declares the symbolic label appearing in the label field is to be assigned the same value as the variable or expression appearing in its argument field. Any symbolic names appearing in the argument field must be previously defined.

Examples:

```
TEST2  EQU    ALPHA
ENTRY  EQU    *-1
SIZE   EQU    TEND-TSTART
```

14.8.7 SET

This pseudo-op is the same as EQU except the name in the label field can be redefined without generating error messages. This capability is frequently required within MACROs.

14.8.8 BSS

A block of words is reserved by this pseudo-op starting at the current program location with a size equal to the number of words specified by the value in the argument field.

If a label is present, it is assigned to the first word of the reserved block. If any symbolic names are present in the argument field, they must be previously defined and the argument field must result in an absolute value.

Examples:

```
TABLE    BSS    25
         BSS    SIZE
LIST     BSS    MAX+1
```

14.8.9 PTR

This pseudo-op is used to define an argument pointer (often used when calling subroutines). Its argument field can be any mode of variable or expression whose value does not exceed 32767 (15 bits). In addition, the indirect bit (sign bit) will be set if an asterisk follows the PTR.

Examples:

```
ARG1     PTR     ALPHA
         PTR*    ARG3+2
         PTR     *
ARG2     PTR*    1000
         PTR     SQRT
         PTR     0
```

14.8.10 DATA

Any of the constant types shown in Table 14-1 may be specified in the argument field of a DATA pseudo-op. As many constants as desired may be specified in the argument field separated by commas. If a label is present in the label field it is assigned to the location of the first word of the first constant.

Character strings within the argument field of a DATA pseudo-op may contain one or two characters and are stored right justified (with a leading zero byte if necessary).

Examples:

```
CONVRSN    DATA    Ø,1,2,7,4,9,2
           DATA    12.3,'AB',.1DØ
NOTE       DATA    'EN','DX',Ø173
           DATA    'A'
           DATA    Ø
```

14.8.11 TEXT (Not in The Basic Assembler)

The TEXT pseudo-op is used to specify a data block consisting of ASCII coded 8-bit characters packed 2 per word (with space character added if necessary to fill the last word).

The variable field consists of a string of characters enclosed within quote characters. Certain characters (such as quote, carriage return, colon, etc.) cannot normally be included within the text string. Such characters can be specified by giving their ASCII code as 3 octal digits preceeded by a colon (:). These four characters will be replaced by the specified 8 bits of data within the data block.

Examples:

```
      TEXT     'PART:2475 NAME:272'
DATE  TEXT     'SEPT 23, 1970'
```

14.8.12 VFD (Not in Basic Assembler)

The Variable Field of the VFD pseudo-op (Variable-Field-Data) consist of pairs of arguments. The first argument of the pair is a value that specifies the number of bits (sub-field width) that the second argument should occupy. The second argument is then positioned properly and combined with the values of other argument pairs specified in this same variable field. The resulting data word is formed from left to right with trailing zeros if necessary. An error message results from a field-width total greater than 16, or from any sub-field value that will not fit within its specified sub-field width.

Examples:

```
VFD     3(1),10(0123),1(1)         (=021234)
VFD     3(A),10(X-BASE+2),1(FLAG),2(MODE)
VFD     F1(X),F2(Y),F3(Z)
```

14.8.13 IF, ENDF (Not in Basic Assembler)

The variable field of the IF pseudo-op is evaluated and if it is zero (False), all source statements following this pseudo-op, up to the corresponding ENDF pseudo-op, are treated as comments. IF and ENDF must be used in pairs and these pairs may be nested within each other to any depth. See the example in the next paragraph.

14.8.14 MAC, ENDM (Not in Basic Assembler)

The label field of the MAC pseudo-op specifies the name of the macro about to be defined. The statements that follow the MAC pseudo-op up to the corresponding ENDM pseudo-op define the "Macro Prototype."

Example:

```
*------Move Macro.   Argument 1 specifies the number of words to
*      move, Argument 2 is the name of the 'FROM' list, argument
*      3 is the name of the 'TO' list.
Move   MAC
       IF        [1].LT.2          Skip if not a 1-word move
       LDAS      [2]
       STAS      [3]
       ENDF
       IF        [1].EQ.2          Skip if not a 2-word move
       DLD       [2]
       DST       [3]
       ENDF
       IF        [1].GT.2          Skip if less then 3-words
       LDXI      [1]
       LDA       [2]-1(X)
       STA       [3]-1(X)
       DXS       1
       JMPS      *-5
       ENDF
       ENDM
```

Calls to a Macro would consist of the Macro name in the operator field, and the arguments to the Macro within the variable field separated by commas if necessary.

Example:

```
       MOVE      1, ALPHA, BETA
       MOVE      20, LIST1, TABLE
```

14.8.15 LIT

Cause literals within a relocatable program to be placed within the program just prior to the "END" statement. If no LIT pseudo-op is encountered, literals will be transmitted to the loader for assignment in its literal pool which permits sharing of common literals by all subroutines loaded.

14.8.16 END

The END pseudo-op must be the last statement within the program being assembled. If a label is specified in the argument field, it represents the starting location of the program.

14.9 LISTING CONTROL

The following pseudo-ops control various listing options that may be set or reset as desired throughout the program. (LIST, LIF and NLMC are initially presumed.)

LIST ----	Generate symbolic listing
NLST ----	List only those lines containing errors
LIF ----	List all card images that are not processed with an IF/ENDF area
NLIF ----	Do not list any lines within an IF/ENDF area that are not assembled
LMAC----	List all lines generated by a macro call
NLMC----	Do not list any macro generated lines

(The above pseudo-ops are not in the basic assembler.)

The assembler automatically skips over paper seams and titles and numbers each page. A quote character (') in Column 1 causes the current page to be ejected and the rest of the line is printed on the top of this new page and all following pages.

A double quote character (") in Column 1 causes the current page to be ejected. (The page header is not modified.)

Example:

```
'        DEBUG PROGRAM (VERSION 3)--------8 June 69
```

4.10 ERROR MESSAGES

If an error is detected by the assembly program one or more of the following error codes will be added to the error columns (left 4 columns) of the listing.

CODE	MEANING
A -------	Incorrect address used.
B -------	Incorrect combination of operands used in an expression.
C -------	Incorrect character used. Any of the following conditions can cause this error: 1. First character of statement incorrect. 2. Argument field of a register copy or shift instruction incorrect. 3. The M-Field of a memory reference instruction is incorrect. 4. An incorrect terminator.
D -------	An EQU or SET pseudo instruction does not have a label field.
E -------	The exponent used in a floating point number is too large.
I -------	An I/O error has occurred.
L -------	Incorrect literal usage.
M -------	Multiple symbolic definitions.
N -------	The number used in this instruction is too large.
O -------	The operation field is undefined.
P -------	Parenthesis incorrectly used in an expression
S -------	The scale factor used in a fixed point number is incorrect.
U -------	Undefined symbol referenced.
V -------	The second word of a valued I/O instruction is incorrect.
$ -------	This in an assembler fault. It indicates that the memory locations reserved for the symbol table is full. The remainder of this assembly will be incorrect.

APPENDIX A

INSTRUCTION SUMMARY

A. 1 INSTRUCTION BY MNEMONICS

Mnemonic	Operation Code	Function	Section
ABD	00 0 23 m	Add to B	5
ADD	00 0 04 m	Add to A	5
ADDS	04 m aaa	Add Short From	5
ADX	00 0 24 m	Add to X	5
AND	00 0 15 m	Logical AND with A	7
ANDS	15 m aaa	Logical AND with A Short Form	7
ARM	10 4 002	Arm Interrupt, Set Overflow OFF	11
ARMF	10 4 003	Arm Interrupt, Set Overflow ON	11
ASLA	00 1 000+n	Arithmetic Shift Left of A	10
ASLB	00 1 200+n	Arithmetic Shift Left of B	10
ASLD	00 1 400+n	Arithmetic Shift Left Double	10
ASRA	00 1 100+n	Arithmetic Shift Right of A	10
ASRB	00 1 300+n	Arithmetic Shift Right of B	10
ASRD	00 1 500+n	Arithmetic Shift Right Double	10
CAB	002 s 5 d	Copy A to B and (s) to (d)	6
CABF	002 s 6 d	Copy A to B and (s) to (d) if Overflow set	6
CAS	00 0 20 m	Compare with A and Skip	9
CAX	002 s 6 d	Copy A to X and (s) to (d)	6
CAXF	002 s 6 d	Copy A to X and (s) to (d) if Overflow set	6
CALL	00 0 07 m	Call Unconditionally	8
CALS	07 m aaa	Call Short Form	8
CBS	00 0 21 m	Compare with B and Skip	9
CIA	10 3 1 dd	Clear and Input to A	11
CIB	10 3 2 dd	Clear and Input to B	11
CIM	10 3 r dd	Clear and Input to Memory	11
CIX	10 3 3 dd	Clear and Input to X	11
CP	002 s 2 d	Copy	6
CPC	002 s 4 d	Copy and Complement	6
CPCF	002 s 4 d	Copy and Complement if Overflow set	6
CPD	002 s 1 d	Copy and Decrement	6
CPDF	002 s 1 d	Copy and Decrement if Overflow set	6

Mnemonic	Operation Code	Function	Section
CPF	002 s 2 d	Copy if Overflow Set	6
CPI	002 s 0 d	Copy and Increment	6
CPIF	002 s 0 d	Copy and Increment if Overflow set	6
CPN	002 s 3 d	Copy and Negate	6
CPNF	002 s 3 d	Copy and Negate if Overflow set	6
CXB	002 s 7 d	Copy X to (B) and (s) to (d)	6
CXBF	002 s 7 d	Copy X to (B) and (s) to (d) if Overflow set	6
CXS	000 22 m	Compare with X and Skip	9
DRM	10 4 000	Disarm Interrupt, set Overflow off	11
DRMF	10 4 001	Disarm Interrupt, set Overflow on	11
DCR	00 0 25 m	Decrement and Replace	5
DIV	00 0 27 m	Divide	5
DADD	00 0 32 m	Double Precision Add	5
DLD	00 0 30 m	Double Precision Load	4
DRS	00 0 10 m	Decrement, Replace, Skip if Ø	10
DST	00 0 31 m	Double Precision Store	4
DSUB	00 0 33 m	Double Precision Subtract	5
DXS	ØØ 3 nnn	Decrement X and Skip if Zero	10
EXCA	10 0 1 dd	External Control from A	11
EXCB	10 0 2 dd	External Control from B	11
EXCI	10 0 0 dd	External Control Immediate	11
EXCM	10 0 r dd	External Control from Memory	11
EXCX	10 0 3 dd	External Control from X	11
FADD	00 0 34 m	Floating Point Addition	5
FDIV	00 0 37 m	Floating Point Division	5
FMUL	00 0 36 m	Floating Point Multiply	5
FSUB	00 0 35 m	Floating Point Subtract	5
HLT	00 0 000	Halt	7
INA	10 5 1 dd	Input and Or with A	11
INB	10 5 2 dd	Input and Or with B	11
INR	00 0 26 m	Increment and Replace	5
INX	10 5 3 dd	Input and Or with X	11
IXS	ØØ 6 nnn	Increment X and Skip if Zero	9
JMP	00 0 17 m	Jump Unconditionally	8
JMPS	17 m aaa	Jump Unconditionally Short Form	8
LDA	00 0 11 m	Load A	4

Mnemonic	Operation Code	Function	Section
LDAS	11 m aaa	Load A Short Form	4
LDB	00 0 12 m	Load B	4
LDBS	12 m aaa	Load B Short Form	4
LDX	00 0 13 m	Load X	
LDXS	13 m aaa	Load X Short Form	4
LEA	00 0 41 m	Load Effective Address into X	4
LRLA	00 1 600+n	Logical Rotate Left of A	10
LRLB	00 1 640+n	Logical Rotate Left of B	10
LRLD	00 1 700+n	Logical Rotate Left Double	10
LSLA	00 1 040+n	Logical Shift Left of A	10
LSLB	00 1 240+n	Logical Shift Left of B	10
LSLD	00 1 440+n	Logical Shift Left Double	10
LSRA	00 1 140+n	Logical Shift Right of A	10
LSRB	00 1 340+n	Logical Shift Right of B	10
LSRD	00 1 540+n	Logical Shift Right Double	10
MUL	00 0 01 m	Multiply	5
MULS	01 m aaa	Multiply Short Form	5
NOP	00 2 000	No Operation	7
ORA	00 0 16 m	Logical OR with A	7
ORAS	16 m aaa	Logical OR with A Short Form	7
OTA	10 2 1 dd	Output from A	11
OTB	10 2 2 dd	Output from B	11
OTI	10 2 0 dd	Output Immediate	11
OTM	10 2 r dd	Output from Memory	11
OTX	10 2 3 dd	Output from X	11
OVF	00 1 74 n	Set Overflow	7
RGC	00 2 sss	Register Copy	6
SAN	00 5 140	Skip if A Negative	9
SANN	00 4 140	Skip if A Not Negative	9
SANP	00 5 040	Skip if A Not Positive	9
SANZ	00 5 100	Skip if A Not Zero	9
SAP	00 4 040	Skip if A Positive	9
SAZ	00 4 100	Skip if A Zero	9
SBNZ	00 5 200	Skip if B Not Zero	9
SBZ	00 4 200	Skip if B Zero	9
SENA	10 1 1 dd	Sense Status to A	11

Mnemonic	Operation Code	Function	Section
SENB	10 1 2 dd	Sense Status to B	11
SENM	10 1 r dd	Sense Status to Memory	11
SENS	10 1 0 dd	Sense Masked Status and Skip if Zero	9
SENX	10 1 3 dd	Sense Status to X	11
SKF	00 5 ccc	Skip if Condition False	9
SKIP	00 5 000	Skip Unconditional	9
SKT	00 4 ccc	Skip if Condition True	9
SNOF	00 5 020	Skip if Overflow Not Set	9
SNS1	00 5 001	Skip if Sense Switch 1 Not Set	9
SNS2	00 5 002	Skip if Sense Switch 2 Not Set	9
SNS3	00 5 004	Skip if Sense Switch 3 Not Set	9
SNS4	00 5 010	Skip if Sense Switch 4 Not Set	9
SOF	00 4 020	Skip if Overflow Set	9
SS1	00 4 001	Skip if Sense Switch 1 Set	9
SS2	00 4 002	Skip if Sense Switch 2 Set	9
SS3	00 4 004	Skip if Sense Switch 3 Set	9
SS4	00 4 010	Skip if Sense Switch 4 Set	9
STA	00 0 02 m	Store A	4
STAS	02 m aaa	Store A Short Form	4
STB	00 0 03 m	Store B	4
STBS	03 m aaa	Store B Short Form	4
STX	00 0 06 m	Store X	4
STXS	06 m aaa	Store X Short Form	4
SUB	00 0 05 m	Subtract	5
SUBS	05 m aaa	Subtract Short Form	5
SXNZ	00 5 400	Skip if X Not Zero	9
SXZ	00 4 400	Skip if X Zero	9
XOR	00 0 14 m	Exclusive OR with A	7
XORS	14 m aaa	Exclusive OR with A Short Form	7

A. 2 INSTRUCTIONS ORDERED BY OP-CODE NUMBER

00 0 000	HLT	00 1 000+n	ASLA	00 4 140	SANN	10 2 0 dd	OTI
00 0 01 m	MUL	00 1 040+n	LSLA	00 4 200	SBZ	10 2 1 dd	OTA
00 0 02 m	STA	00 1 100+n	ASRA	00 4 400	SXZ	10 2 2 dd	OTB
00 0 03 m	STB	00 1 140+n	LSRA	00 5 ccc	SKF	10 2 3 dd	OTX
00 0 04 m	ADD	00 1 200+n	ASLB	00 5 000	SKIP	10 2 r dd	OTM
00 0 05 m	SUB	00 1 240+n	LSLB	00 5 001	SNS1	10 3 1 dd	CIA
00 0 06 m	STX	00 1 300+n	ASRB	00 5 002	SNS2	10 3 2 dd	CIB
00 0 07 m	CALL	00 1 340+n	LSRB	00 5 004	SNS3	10 3 3 dd	CIX
00 0 10 m	DRS	00 1 400+n	ASLD	00 5 010	SNS4	10 3 r dd	CIM
00 0 11 m	LDA	00 1 440+n	LSLD	00 5 020	SNOF	10 4 000	DRM
00 0 12 m	LDB	00 1 500+n	ASRD	00 5 040	SANP	10 4 001	DRMF
00 0 13 m	LDX	00 1 540+n	LSRD	00 5 100	SANZ	10 4 002	ARM
00 0 14 m	XOR	00 1 600+n	LRLA	00 5 140	SAN	10 4 003	ARMF
00 0 15 m	AND	00 1 640+n	LRLB	00 5 200	SBNZ	10 5 1 dd	INA
00 0 16 m	ORA	00 1 700+n	LRLD	00 5 400	SXNZ	10 5 2 dd	INB
00 0 17 m	JMP	00 1 740+n	OVF	00 6 nnn	IXS	10 5 3 dd	INX
00 0 20 m	CAS	00 2 000	NOP	00 7 xnn	indexed shift	11 m aaa	LDAS
00 0 21 m	CBS	00 2 sss	RGC			12 m aaa	LDBS
00 0 22 m	CXS	00 2 s0d	CPI	01 m aaa	MULS	13 m aaa	LDXS
00 0 23 m	ADB	00 2 s1d	CPD	02 m aaa	STAS	14 m aaa	XORS
00 0 24 m	ADX	00 2 s2d	CP	03 m aaa	STBS	15 m aaa	ANDS
00 0 25 m	DCR	00 2 s3d	CPN	04 m aaa	ADDS	16 m aaa	ORAS
00 0 26 m	INR	00 2 s4d	CPC	05 m aaa	SUBS	17 m aaa	JMPS
00 0 27 m	DIV	00 2 s5d	CAB	06 m aaa	STXS		
00 0 30 m	DLD	00 2 s6d	CAX	07 m aaa	CALS		
00 0 31 m	DST	00 2 s7d	CXB	10 0 0 dd	EXCI		
00 0 32 m	DADD	00 3 nnn	DXS	10 0 1 dd	EXCA		
00 0 33 m	DSUB	00 4 ccc	SKT	10 0 2 dd	EXCB		
00 0 34 m	FADD	00 4 001	SS1	10 0 3 dd	EXCX		
00 0 35 m	FSUB	00 4 002	SS2	10 0 r dd	EXCM		
00 0 36 m	FMUL	00 4 004	SS3	10 1 0 dd	SENS		
00 0 37 m	FDIV	00 4 010	SS4	10 1 1 dd	SENA		
00 0 40 m	LEA	00 4 020	SOF	10 1 2 dd	SENB		
00 0 41 n		00 4 040	SAP	10 1 3 dd	SENX		
⋮	(trap)	00 4 100	SAZ	10 1 r dd	SENM		
00 0 77 n							

APPENDIX B

TABLES AND CONSTANTS

B.1 TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 921	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 766	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125

B.2 TABLE OF POWERS OF TEN IN OCTAL

10^n	n	10^{-n}
1	0	1.000 000 000 000 000 000 00
12	1	0.063 146 314 631 463 146 31
144	2	0.005 075 341 217 270 243 66
1 750	3	0.000 406 111 564 570 651 77
23 420	4	0.000 032 155 613 530 704 15
303 240	5	0.000 002 476 132 610 706 64
3 641 100	6	0.000 000 206 157 364 055 37
46 113 200	7	0.000 000 015 327 745 152 75
575 360 400	8	0.000 000 001 257 143 561 06
7 346 545 000	9	0.000 000 000 104 560 276 41
112 402 762 000	10	0.000 000 000 006 676 337 66
1 351 035 564 000	11	0.000 000 000 000 537 657 77
16 432 451 210 000	12	0.000 000 000 000 043 136 32
221 411 634 520 000	13	0.000 000 000 000 003 411 35
2 657 142 036 440 000	14	0.000 000 000 000 000 264 11
34 327 724 461 500 000	15	0.000 000 000 000 000 022 01
434 157 115 760 200 000	16	0.000 000 000 000 000 001 63
5 432 127 413 542 400 000	17	0.000 000 000 000 000 000 14
67 405 553 164 731 000 000	18	0.000 000 000 000 000 000 01

B.3 USEFUL MATHMATICAL CONSTANTS IN OCTAL

π = 3.11037 552421	e = 2.55760 521305	γ = 0.44742 147707
π^{-1} = 0.24276 301556	e^{-1} = 0.27426 530661	$\ln \gamma$ = 0.43127 233602
$\sqrt{\pi}$ = 1.61337 611067	$\sqrt{e}$ = 1.51411 230704	$\log_2 \gamma$ = 0.62573 030645
$\ln \pi$ = 1.11206 404435	$\log_{10} e$ = 0.33626 754251	$\sqrt{2}$ = 1.32404 746320
$\log_2 \pi$ = 1.51544 163223	$\log_2 e$ = 1.34252 166245	$\ln 2$ = 0.54271 027760
$\sqrt{10}$ = 3.12305 407267	$\log_2 10$ = 3.24464 741136	$\ln 10$ = 2.23273 067355

APPENDIX C

SPIRAS-65 CODES

CHAR.	TELE-TYPE ASCII CODE	INTER-NAL ASCII CODE	029 CARD CODE	CARD READER CODE
0	060	260	0	00
1	261	261	1	01
2	262	262	2	02
3	063	263	3	03
4	264	264	4	04
5	065	265	5	05
6	066	266	6	06
7	267	267	7	07
8	270	270	8	10
9	071	271	9	11
A	101	301	12-1	61
B	102	302	12-2	62
C	303	303	12-3	63
D	104	304	12-4	64
E	305	305	12-5	65
F	306	306	12-6	66
G	107	307	12-7	67
H	110	310	12-8	70
I	311	311	12-9	71
J	312	312	11-1	41
K	113	313	11-2	42
L	314	314	11-3	43
M	115	315	11-4	44
N	116	316	11-5	45
O	317	317	11-6	46
P	120	320	11-7	47
Q	321	321	11-8	50
R	322	322	11-9	51
S	123	323	0-2	22
T	324	324	0-3	23
U	125	325	0-4	24
V	126	326	0-5	25

CHAR.	TELE-TYPE ASCII CODE	INTER-NAL ASCII CODE	029 CARD CODE	CARD READER CODE
W	327	327	0-6	26
X	330	330	0-7	27
Y	131	331	0-8	30
Z	132	332	0-9	31
!	041	241	11-8-2	52
"	042	242	8-7	17
#	243	243	8-3	13
$	044	244	11-8-3	53
%	245	245	0-8-4	34
&	246	246	12	60
'	047	247	8-5	15
(	050	250	12-8-5	75
)	251	251	11-8-5	55
*	252	252	11-8-4	54
+	053	253	12-8-6	76
,	254	254	0-8-3	33
-	055	255	11	40
.	056	256	12-8-3	73
/	257	257	0-1	21
:	072	272	8-2	12
;	273	273	11-8-6	56
<	074	274	12-8-4	74
=	275	275	8-6	16
>	276	276	0-8-6	36
?	077	277	0-8-7	37
@	300	300	8-4	14
[	333	333	12-8-2	72
\	134	334	0-8-2	32
]	335	335	11-8-7	57
↑	336	336	12-8-7	77
←	137	337	0-8-5	35
(sp)	240	240	(blank)	20

APPENDIX C (Continued)

SPIRAS-65 CODES

CHARACTER	TELETYPE ASCII CODE	INTERNAL ASCII CODE
SOM	201	201
EOA	202	202
EOM(EOF)	003	203
EOT(STOP)	204	204
WRU	005	205
RU	006	206
BELL	207	207
FE	210	210
HORZ TAB	011	211
LINE FEED	012	212
VERT TAB	213	213
FORM	014	214
CAR. RET.	215	215
SO	216	216
SI	017	217
DCO	220	220
X-ON	021	221
P-ON	022	222
X-OFF	223	223
P-OFF	024	224
ERROR	225	225
SYNC	227	227
SPACE	240	240
RUB-OUT	377	377

APPENDIX D
PROGRAM EXAMPLES

```
0001          PROGRAM TO LIST CARDS OR (SPACE) COMPRESSED TAPES

         *      PROGRAM TC LIST CARDS OR (SPACE) COMPRESSED TAPES
         *
         *              **************************************
         *              *                                    *
         *              *  C A R D / T A P E   L I S T E R   *
         *              *                                    *
         *              **************************************
         *
         *------THIS PROGRAM USES THE OPERATING SYSTEM TO READ A RECORD FROM
         *      ONE DEVICE (CARD READER, MAG TAPE, PAPER TAPE, ETC) AND OUTPUT
         *      IT TO A SECOND DEVICE (LINE PRINTER, ASR PRINTER, ETC).  THE
         *      CHOICE OF DEVICES IS MADE WITHIN THE OPERATING SYSTEM.  SENSE
         *      SWITCH 4 WILL CAUSE A PAUSE (FOR OPERATOR ACTION) BEFORE INPUT
         *      OF THE NEXT RECORD.  PAGES WILL BE NUMBERED.
         *
         *
                BOOT                    ASSEMBLE IN BOOTSTRAP FORMAT
                ORG     05C00
         *
000076   IO     EQU     076             ENTRY POINT TO OPERATING SYSTEM
```

```
0002        PROGRAM TO LIST CARDS OR (SPACE) COMPRESSED TAPES

                      "
                      *------OPEN INPUT AND OUTPUT DEVICES
005000: 074076        LIST   CALS*   IO                  OPEN INPUT DEVICE
005001: 001001               DATA    001001
005002: 074076               CALS*   IO                  OPEN OUTPUT DEVICE
005003: 001044               DATA    001044
005004: 003774               DXS     4
005005: 066073               STXS    MAXCNT              SET MAX LINES PER PAGE
005006: 002121               CP      0,A
005007: 026070               STAS    PAGENO              RESET PAGE NUMBER
005010: 176023               JMPS    PAGE                OUTPUT A PAGE EJECT
                      *
                      *------START OF COPY LOOP
005011: 005010        LIST4  SNS4
005012: 000000               HLT                         WAIT FOR OPERATOR ACTION
005013: 000130               LDXI    BUFFER              LOCATION OF BUFFER
005014: 005114
005015: 074076               CALS*   IO                  INPUT 1 RECORD
005016: 000101               DATA    000101              KEYWORD
005017: 000120               DATA    80                  SIZE OF BUFFER
005020: 004100               SAZ
005021: 176006               JMPS    LIST8               INPUT STATUS ERROR
005022: 000130               LDXI    BUFFER
005023: 005114
005024: 074076               CALS*   IO                  OUTPUT 1 RECORD
005025: 010344               DATA    010344
005026: 000120               DATA    80
005027: 004100               SAZ
005030: 000000        LIST8  HLT                         STATUS ERROR PRESENT
                      *
                      *------CHECK IF PAGE EJECT REQUIRED
005031: 000101               DRS     COUNT               DECREMENT LINE COUNT
005032: 005077
005033: 177755               JMPS    LIST4               NOT AT BOTTOM OF PAGE
005034: 000261        PAGE   INR     PAGENO
005035: 005100
005036: 136044               LDXS    =4                  CONVERT PAGE NO. TO ASCII CHARACTERS
005037: 126040               LDBS    PAGENO
```

```
0003          PROGRAM TO LIST CARDS OR (SPACE) COMPRESSED TAPES

     005040: 002121   0055 PAGE2  CP      0,A
     005041: 000270   0056        DIVI    10
     005042: 000012
     005043: 046040   0057        ADDS    =0260
     005044: 000026   0058        STA     BUF+10(X)      STORE INTO PAGE HEADER BUFFER
     005045: 005072
     005046: 003777   0059        DXS     1
     005047: 177770   0060        JMPS    PAGE2
     005050: 000130   0061        LDXI    BUF            HEADER BUFFER LOCATION
     005051: 005060
     005052: 074076   0062        CALS*   IC             EJECT PAGE AND PRINT PAGE NUMBER
     005053: 020444   0063        DATA    020444
     005054: 000017   0064        DATA    15
     005055: 116023   0065        LDAS    MAXCNT         RESET LINE COUNT
     005056: 026020   0066        STAS    COUNT
     005057: 177731   0067        JMPS    LIST4          PROCESS NEXT RECORD
                      0068 *
     005060: 000240   0069 BUF    DATA    ' ',' ',' ',' ',' ',' '
     005061: 000240
     005062: 000240
     005063: 000240
     005064: 000240
     005065: 000240
     005066: 000320   0070        DATA    'P','A','G','E',' '
     005067: 000301
     005070: 000307
     005071: 000305
     005072: 000240
                      0071        BSS     4
                      0072 *
     005077: 000000   0073 COUNT  DATA    0              CURRENT LINE COUNT
     005100: 000000   0074 PAGENO DATA    0              CURRENT PAGE NUMBER
     005101: 000000   0075 MAXCNT DATA    0              MAXIMUM LINE COUNT/PAGE
                      0076 LITRLS BSS     10             ROOM FOR ANY NEEDED LITERALS
                      0077 BUFFER BSS     80             COPY BUFFER
                      0078 *
                      0079        ORG     LITRLS         CAUSE LITERALS TO BE OUTPUT BEFORE BUFFER
     005102:          0080        END
```

```
LITERAL TABLE
    005103: 000004
    0051C4: 00026C



SYMBOL   LOC       CARDS THAT REFERENCE SYMBCL
BUF      005060  CC58 CC61
BUFFER   005114  CC36 CC42
COUNT    005077  CC5C CC66
ID       C00C76  CC23 CC25 0C37 C043 C062
LIST4    C05011  0C51 CC67
LIST8    C05030  CC41
LIST     CC5000
LITRLS   005102  CC79
MAXCNT   005101  CC28 CC65
PAGENO   005100  CC30 CC52 0C54
PAGE2    005040  CC6C
PAGE     005034  CC31



CARDS WITH ERRORS:    NONE
```

```
0001        TYPE OCTAL WORD SUBROUTINE     (20 FEB 70)

        '      TYPE OCTAL WORD SUBROUTINE     (20 FEB 70)
        *
        *      ****************************************
        *      *                                      *
        *      *    T Y P E   O C T A L   W O R D     *
        *      *                                      *
        *      *         S U B R O U T I N E          *
        *      *                                      *
        *      ****************************************
        *
        *------CALLING SEQUENCES   ( (X) REGISTER IS NOT MODIFIED)
        *      (TYPE 6 OCTAL CHARS.)    (TYPE 6 OCTAL CHARS. WITHIN PARENS)
        *      LDBS    VALUE            LDBS    VALUE
        *      CALS    TYPOCT           CALS    TYPIOC
        *      (RETURN)                 (RETURN)
        *
        *
               ENT     TYPOCT,TYPIOC
               EXT     TYPCHR
        *
        *
               ORG     *
```

```
0002        TYPE OCTAL WORD SUBROUTINE     (20 FEB 70)


                        0023 "
                        0024 *-------TYPE OCTAL WORD
    000000: 000171      0025         JMP     0                EXIT
    000001: 000000
            000001      0026 TYPOCT  EQU     *-1              ENTRY LOCATION
    000002: 060015      0027         STXS    TYPOC6+1         SAVE INDEX
    000003: 130033      0028         LDXS    =-6
    000004: 002121      0029         CP      0,A
    000005: 001441      0030         LSLD    1
    000006: 040032      0031 TYPOC4  ADDS    =0260
    000007: 070000      0032         CALS    TYPCHR           TYPE CHARACTER
    000010: 002121      0033         CP      0,A
    000011: 001443      0034         LSLD    3
    000012: 006001      0035         IXS     1
    000013: 170006      0036         JMPS    TYPOC4
    000014: 000130      0037 TYPOC6  LDXI    0                RESET INDEX
    000015: 000000
    000016: 170000      0038         JMPS    TYPOCT-1         RETURN
                        0039 *
                        0040 *-------TYPE OCTAL WORD WITHIN PARENTHESES
    000017: 000171      0041         JMP     0                EXIT
    000020: 000000
            000020      0042 TYPIOC  EQU     *-1              ENTRY LOCATION
    000021: 000110      0043         LDAI    0124240          (SP)(()
    000022: 124240
    000023: 070000      0044         CALS    TYPCHR           TYPE 2 CHARACTERS
    000024: 070001      0045         CALS    TYPOCT           TYPE OCTAL VALUE
    000025: 000110      0046         LDAI    0120251          ())(SP)
    000026: 120251
    000027: 070000      0047         CALS    TYPCHR           TYPE 2 CHARACTERS
    000030: 170017      0048         JMPS    TYPIOC-1         RETURN
                        0049 *
    000031:             0050         END

LITERAL TABLE
    000032: 000260
    000033: 177772


SYMBOL  LOC       CARDS THAT REFERENCE SYMBOL
TYPCHR  177777  0032 0044 0047
TYPIOC  000020  0048
TYPOCT  000001  0038 0045
TYPOC4  000006  0036
TYPOC6  000014  0027
```

```
0001          TYPE CHARACTER SUBROUTINE          (20 FEB 70)


                           0001 *      TYPE CHARACTER SUBROUTINE          (20 FEB 70)
                           0002 *
                           0003 *      *****************************************
                           0004 *      *                                       *
                           0005 *      *     T Y P E   C H A R A C T E R       *
                           0006 *      *                                       *
                           0007 *      *        S U B R O U T I N E            *
                           0008 *      *                                       *
                           0009 *      *****************************************
                           0010 *
                           0011 *------CALLING SEQUENCE  ( (B) AND (X) NOT MODIFIED)
                           0012 *      LDAS    CHAR             (A)= 00000000C1C1C1C1 OR C2C2C2C2C1C1C1C1
                           0013 *      CALS    TYPCHR
                           0014 *      (RETURN)
                           0015 *
                           0016 *
                           0017        ENT     TYPCHR
                           0018 *
                           0019        ORG     *
                           0020 *
                           0021 *------TYPE 1 OR 2 CHARACTERS
    000000: 000171         0022        JMP     0                EXIT
    000001: 000000
            000001         0023 TYPCHR EQU     *-1              ENTRY POINT
    000002: 101002         0024        SENS    2,0100
    000003: 000100
    000004: 170002         0025        JMPS    *-2              WAIT IF PRINTER BUSY
    000005: 102102         0026        OTA     2                TYPE ASCII CHARACTER
    000006: 005140         0027        SAN                      SKIP IF 2 CHARACTERS TO OUTPUT
    000007: 170000         0028        JMPS    TYPCHR-1         RETURN
    000010: 001150         0029        LSRA    8
    000011: 170002         0030        JMPS    TYPCHR+1         TYPE SECOND CHARACTER
                           0031 *
    000012:                0032        END



 SYMBOL  LOC      CARDS THAT REFERENCE SYMBOL
 TYPCHR  000001   0028 0030



CARDS WITH ERRORS:     NONE
```

APPENDIX E

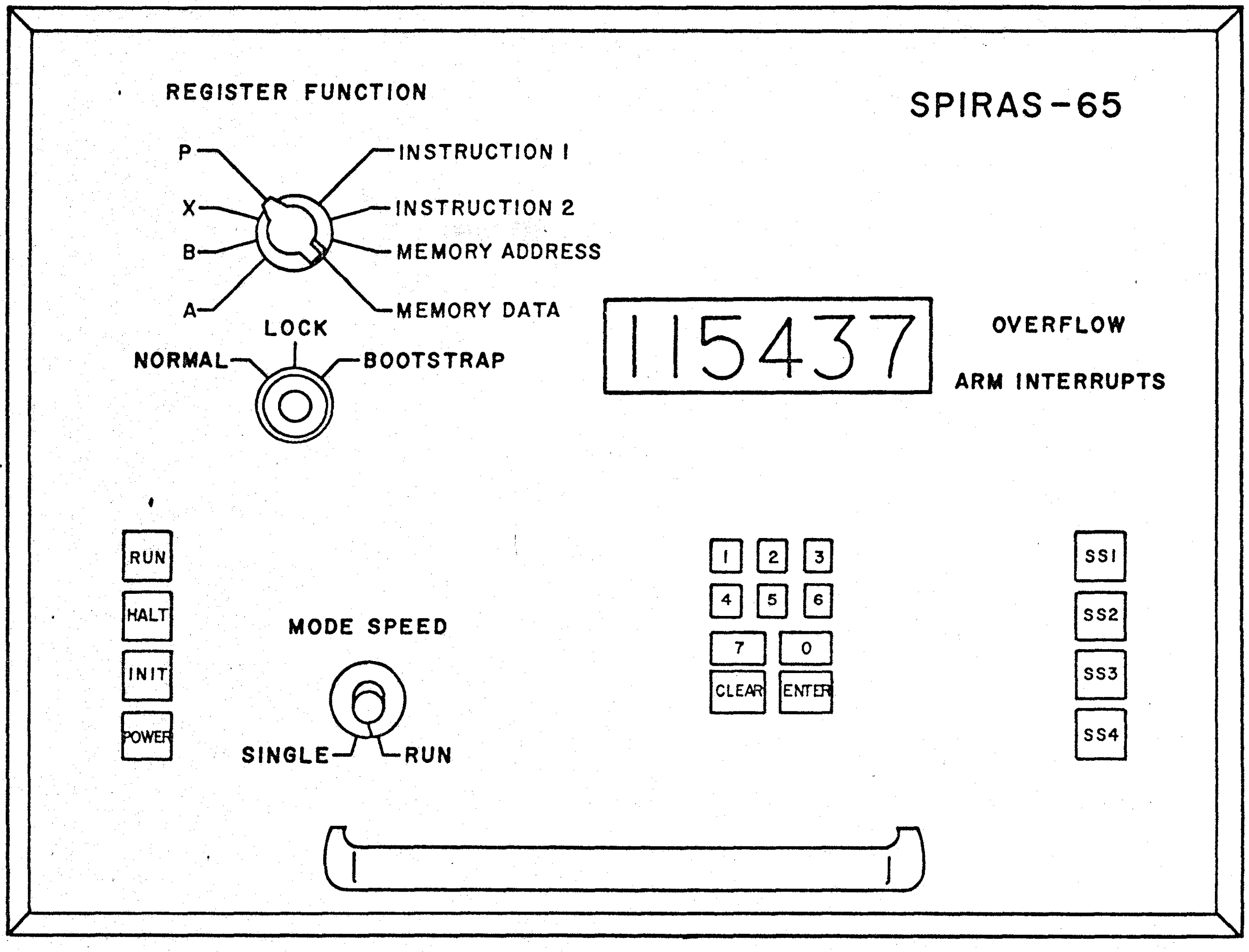
CONSOLE CONTROLS

CONTROL/POSITION	DESCRIPTION
Register Function	Display on NIXIE Tubes the function indicated
A	A register
B	B register
X	X register
P	P register
INSTRUCTION 1	MEMORY (P)
INSTRUCTION 2	MEMORY (P+1)
MEMORY ADDRESS	Address at which "MEMORY DATA" is located
MEMORY DATA	Location determined by "MEMORY ADDRESS"
Mode Speed	
RUN	Processor Runs at full speed when RUN is pressed
SINGLE STEP	Processor executes a single instruction when RUN is pressed
VARIABLE	Processor RUNS at a speed determined by potentiometer setting when RUN is pressed
Keyboard	
CLEAR	Clears NIXIE display and gives control to operator.
NUMERALS	Enters numeral depressed into display
ENTER	Contents of the display replace the contents of the register indicated by the REGISTER FUNCTION switch.
POWER	Alternate action switch controls primary power.
INIT	Momentary switch resets computer control logic. Computer is then in the HALT mode.
HALT	Momentary switch halts the processor after the instruction currently in process is completed.
RUN	Momentary switch causes the processor to run in the mode determined by the MODE-SPEED switch
SS1 - SS4	Alternate action switches that can be tested by skip on sense switch instructions.

Key Lock

NORMAL	The usual operating position where all controls are functional
LOCK	The ENTER, HALT, INIT, and RUN switches are disabled.
BOOTSTRAP	Upon pressing the INIT switch the bootstrap program is entered.

INDICATOR	DESCRIPTION
OVERFLOW	Indicates that the OVERFLOW flag is set.
ARM INTERRUPTS	Indicates that the interrupts are armed (can interrupt program).



REGISTER FUNCTION
SPIRAS-65
P
X
B
A
INSTRUCTION 1
INSTRUCTION 2
MEMORY ADDRESS
MEMORY DATA
LOCK
NORMAL
BOOTSTRAP
115437
OVERFLOW
ARM INTERRUPTS
RUN
HALT
INIT
POWER
MODE SPEED
SINGLE
RUN
1
2
3
4
5
6
7
0
CLEAR
ENTER
SS1
SS2
SS3
SS4

SPIRAS SYSTEMS, INC.
332 Second Avenue
Waltham, Mass.

EHS-69-M145

SUBJECT: PHASE Ø SPIRAS-65

DATE: 24 October 1969

FROM: E. H. Sonn

TO: All SPIRAS-65 Users

The User's Manual, dated October, 1969, is written for the Phase I machines which will be released before the end of 1969. The user should be aware of the OP Code and other operational differences between the two machines.

OP CODES

MNEMONIC	PHASE Ø CODE	PHASE I CODE
LDA	ØØØ1m	ØØ11m
LDAS	Ø1maaa	11maaa
LDB	ØØØ2m	ØØ12m
LDBS	02maaa	12maaa
LDX	ØØØ3m	ØØ13m
LDXS	Ø3maaa	13maaa
STA	ØØ11m	ØØØ2m
STAS	11maaa	Ø2maaa
STB	ØØ12m	ØØØ3m
STBS	12maaa	Ø3maaa
STX	ØØ13m	ØØØ6m
STXS	13maaa	Ø6maaa
MUL	ØØØ6m	ØØØ1m
MULS	Ø6maaa	Ø1maaa
IXS	1Ø6nnn	ØØ6nnn
DXS	1Ø7nnn	ØØ3nnn
INA	1Ø35dd	1Ø51dd
INB	1Ø36dd	1Ø52dd
INX	1Ø37dd	1Ø53dd

ADDRESSING MODES

The long form Addressing Modes have been changed as shown:

MODE CODE	PHASE Ø	PHASE I
Ø	Immediate	Immediate
1	Direct	Direct
2	Indirect	Indirect
3	Indirect Pre-indexed with X	Indirect Pre-indexed with X
4	Indexed with A	Indexed with A
5	Indexed with B	Indirect Post-indexed with X
6	Indexed with X	Indexed with X
7	Relative to P	Relative to P

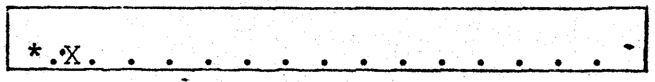
INPUT-OUTPUT INSTRUCTION FORMAT

The Input-Output Memory Reference format has been changed. In the Phase I format, the second word of the instruction is a sixteen bit address whereas the Phase Ø format has an indirect bit, an index bit and a fourteen bit address. The Addressing Modes are now indicated in the first word of the instruction. The new format permits the use of sixteen bit addresses, but limits indexing on indirect to pre-indexing or post-indexing whereas previously, indexing could be accomplished on any level of indirect addressing. See Page 3-2 of the Manual for a more complete description.

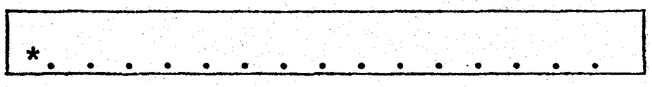
INDIRECT ADDRESS FORMAT

The Indirect Address formats have been changed to eliminate the index bit.

PHASE Ø



PHASE I



BOOTSTRAP OPERATION

The Phase Ø Bootstrap requires that the "Run" button be pressed between reading the secondary Bootstrap and the main body of the tape. This Bootstrap cannot be used for relocatable tapes. A Bootstrap Simulator Program has been provided for reading relocatable tapes from the teletype. Another Bootstrap Simulator has been provided to users of the high-speed paper tape reader. This simulator is necessary to read either absolute or relocatable tapes.

INSTRUCTION SUMMARY

For your convenience, a copy of the Instruction Summary from the previous edition of the manual is attached.

[illegible]

E. H. Sonn

EHS:HL

Attachment

APPENDIX 1: INSTRUCTION SUMMARY

1. INSTRUCTIONS BY MNEMONICS

Mnemonic	Operation Code	Function	Page
ABD	00 0 23 m	Add to B	C1
ADD	00 0 04 m	Add to A	C1
ADDS	04 m aaa	Add Short Form	C1
ADX	00 0 24 m	Add to X	C1
AND	00 0 15 m	Logical AND with A	E1
ANDS	15 m aaa	Logical AND with A Short Form	E1
ARM	10 4 002	Arm Interrupt, Set Overflow OFF	J4
ARMF	10 4 003	Arm Interrupt, Set Overflow On	J4
ASLA	00 1 000+n	Arithmetic Shift Left of A	H4
ASLB	00 1 200+n	Arithmetic Shift Left of B	H4
ASLD	00 1 400+n	Arithmetic Shift Left Double	H4
ASRA	00 1 100+n	Arithmetic Shift Right of A	H5
ASRB	00 1 300+n	Arithmetic Shift Right of B	H5
ASRD	00 1 500+n	Arithmetic Shift Right Double	H5
CAB	002 s 5 d	Copy A to B and (s) to (d)	D2
CABF	002 s 6 d	Copy A to B and (s) to (d) if Overflow set	D2
CAS	00 0 20 m	Compare with A and Skip	G4
CAX	002 s 6 d	Copy A to X and (s) to (d)	D2
CAXF	002 s 6 d	Copy A to X and (s) to (d) if Overflow set	D2
CALL	00 0 07 m	Call Unconditionally	F1
CALS	07 m aaa	Call Short Form	F1
CBS	00 0 21 m	Compare with B and Skip	G4
CIA	10 3 1 dd	Clear and Input to A	J3
CIB	10 3 2 dd	Clear and Input to B	J3
CIM	10 3 4 dd	Clear and Input to Memory	J4
CIX	10 3 3 dd	Clear and Input to X	J3
CP	002 s 2 d	Copy	D2
CPC	002 s 4 d	Copy and Complement	D2
CPCF	002 s 4 d	Copy and Complement if OV set	D2

Mnemonic	Operation Code	Function	Page
CPD	002 s 1 d	Copy and Decrement	D2
CPDF	002 s 1 d	Copy and Decrement if OV set	D2
CPF	002 s 2 d	Copy if Overflow Set	D2
CPI	002 s 0 d	Copy and Increment	D2
CPIF	002 s 0 d	Copy and Increment if OV set	D2
CPN	002 s 3 d	Copy and Negate	D2
CPNF	002 s 3 d	Copy and Negate if OV set	D2
CXB	002 s 7 d	Copy X to (B) and (s) to (d)	D2
CXBF	002 s 7 d	Copy X to (B) and (s) to (d) if Overflow set	D2
CXS	000 22 i	Compare with X and Skip	G4
DRM	10 4 000	Disarm Interrupt, set OV off	J5
DRMF	10 4 001	Disarm Interrupt, set OV on	J5
DCR	00 0 25 m	Decrement and Replace	C4
DIV	00 0 27 m	Divide	C2
DADD	00 0 32 m	Double Precision Add	C2
DLD	00 0 30 m	Double Precision Load	B2
DRS	00 0 10 m	Decrement, Replace, Skip if Ø	G5
DST	00 0 31 m	Double Precision Store	B3
DSUB	00 0 33 m	Double Precision Subtract	C3
DXS	10 7 nnn	Decrement X and Skip if Zero	G5
EXCA	10 0 1 dd	External Control from A	J1
EXCB	10 0 2 dd	External Control from B	J1
EXCI	10 0 0 dd	External Control Immediate	J2
EXCM	10 0 4 dd	External Control from Memory	J1
EXCX	10 0 3 dd	External Control from X	J1
FSUB	00 0 35 m	Floating Point Subtract	C3
HLT	00 0 000	Halt	E2
INA	10 3 5 dd	Input and Or with A	J4
INB	10 3 6 dd	Input and Or with B	J4
INR	00 0 26 i	Increment and Replace	C4
INX	10 3 7 dd	Input and Or with X	J4
IXS	10 6 nnn	Increment X and Skip if Zero	G5
JMP	00 0 17 m	Jump Unconditionally	F1
JMPS	17 m aaa	Jump Unconditionally Short Form	F1
LDA	00 0 01 m	Load A	B1
LDAS	01 m aaa	Load A Short Form	B1

Mnemonic	Operation Code	Function	Page
LDB	00 0 02 m	Load B	B1
LDBS	02 m aaa	Load B Short Form	B1
LDX	00 0 03 m	Load X	B1
LDXS	03 m aaa	Load X Short Form	B1
LEA	00 0 41 m	Load Effective Address into X	B3
LRLA	00 1 600+n	Logical Rotate Left of A	H3
LRLB	00 1 640+n	Logical Rotate Left of B	H3
LRLD	00 1 700+n	Logical Rotate Left Double	H4
LSLA	00 1 040+n	Logical Shift Left of A	H2
LSLB	00 1 240+n	Logical Shift Left of B	H2
LSLD	00 1 440+n	Logical Shift Left Double	H2
LSRA	00 1 140+n	Logical Shift Right of A	H2
LSRB	00 1 340+n	Logical Shift Right of B	H3
LSRD	00 1 540+n	Logical Shift Right Double	H3
MUL	00 0 06 m	Multiply	C2
MULS	06 m aaa	Multiply Short Form	C2
NOP	00 2 000	No Operation	E2
ORA	00 0 16 m	Logical OR with A	E2
ORAS	16 m aaa	Logical OR with A Short Form	E2
OTA	10 2 1 dd	Output from A	J2
OTB	10 2 2 dd	Output from B	J2
OTI	10 2 0 dd	Output Immediate	J3
OTM	10 2 4 dd	Output from Memory	J3
OTX	10 2 3 dd	Output from X	J3
OVF	00 1 74x	Set Overflow	E2
RGC	00 2 sss	Register Copy	D2
SAN	00 4 140	Skip if A Negative	G3
SANN	00 5 140	Skip if A Not Negative	G3
SANP	00 5 040	Skip if A Not Positive	G3
SANZ	00 5 100	Skip is A Not Zero	G3
SAP	00 4 040	Skip if A Positive	G3
SAZ	00 4 100	Skip if A Zero	G3
SBNZ	00 5 200	Skip if B Not Zero	G3
SBZ	00 4 200	Skip if B Zero	G3
SENA	10 1 1 dd	Sense Status to A	J2

Mnemonic	Operation Code	Function	Page
SENB	10 1 2 dd	Sense Status to B	J2
SENM	10 1 4 dd	Sense Status to Memory	J2
SENS	10 1 0 dd	Sense Masked Status and Skip if Zero	G5
SENX	10 1 3 dd	Sense Status to X	J2
SKF	00 5 ccc	Skip if Condition False	G1
SKP	00 5 000	Skip Unconditional	G4
SKT	00 4 ccc	Skip if Condition True	G1
SNOF	00 5 020	Skip if Overflow Not Set	G2
SNS1	00 5 001	Skip if Sense Switch 1 Not Set	G2
SNS2	00 5 002	Skip if Sense Switch 2 Not Set	G2
SNS3	00 5 004	Skip if Sense Switch 3 Not Set	G2
SNS4	00 5 010	Skip if Sense Switch 4 Not Set	G2
SOF	00 4 020	Skip if Overflow Set	G2
SS1	00 4 001	Skip if Sense Switch 1 Set	G2
SS2	00 4 002	Skip if Sense Switch 2 Set	G2
SS3	00 4 004	Skip if Sense Switch 3 Set	G2
SS4	00 4 010	Skip if Sense Switch 4 Set	G2
STA	00 0 11 m	Store A	B2
STAS	11 m aaa	Store A Short Form	B2
STB	00 0 12 m	Store B	B2
STBS	12 m aaa	Store B Short Form	B2
STXS	13 m aaa	Store X Short Form	B2
SUB	00 0 05 m	Subtract	C1
SUBS	05 m aaa	Subtract Short Form	C1
SXNZ	00 5 400	Skip if X Not Zero	G3
SXZ	00 4 400	Skip if X Zero	G3
XOR	00 0 14 m	Exclusive OR with A	E1
XORS	14 m aaa	Exclusive OR with A Short Form	E1

2. INSTRUCTIONS ORDERED BY OP-CODE NUMBER

00 0 000	HLT	00 1 000+n	ASLA	00 4 140	SANN	10 2 0 dd	OTI
00 0 01 m	LDA	00 1 040+n	LSLA	00 4 200	SBZ	10 2 1 dd	OTA
00 0 02 m	LDB	00 1 100+n	ASRA	00 4 400	SKX	10 2 2 dd	OTB
00 0 03 m	LDX	00 1 140+n	LSRA	00 5 ccc	SKF	10 2 3 dd	OTX
00 0 04 m	ADD	00 1 200+n	ASLB	00 5 000	SKF	10 2 4 dd	OTM
00 0 05 m	SUB	00 1 240+n	LSLB	00 5 001	SNS1	10 3 1 dd	CIA
00 0 06 m	MUL	00 1 300+n	ASRB	00 5 002	SNS2	10 3 2 dd	CIB
00 0 07 m	CALL	00 1 340+n	LSRB	00 5 004	SNS3	10 3 3 dd	CIX
00 0 10 m	DRS	00 1 400+n	ASLD	00 5 010	SNS4	10 3 4 dd	CIM
00 0 11 m	STA	00 1 440+n	LSLD	00 5 020	SNOF	10 3 5 dd	INA
00 0 12 m	STB	00 1 500+n	ASRD	00 5 040	SANP	10 3 6 dd	INB
00 0 13 m	STX	00 1 540+n	LSRD	00 5 100	SANZ	10 3 7 dd	INX
00 0 14 m	XOR	00 1 600+n	LRLA	00 5 140	SAN	10 4 000	DRM
00 0 15 m	AND	00 1 640+n	LRLB	00 5 200	SBNZ	10 4 001	DRMF
00 0 16 m	ORA	00 1 700+n	LRLD	00 5 400	SXNZ	10 4 002	ARM
00 0 17 m	JMP	00 1 740+n	CVF	00 6 nnn	(trap)	10 4 003	ARMF
00 0 20 m	CAS	00 2 000	NOP	00 7 snn	indexed shift	10 6 nnn	IXS
00 0 21 m	CBS	00 2 sss	RGC			10 7 nnn	DXS
00 0 22 m	CXS	00 2 s0d	CPI	01 m aaa	LDAS	11 m aaa	STAS
00 0 23 m	ADB	00 2 s1d	CPD	02 m aaa	LDBS	12 m aaa	STBS
00 0 24 m	ADX	00 2 s2d	CP	03 m aaa	LDXS	13 m aaa	STXS
00 0 25 m	DCR	00 2 s3d	CPN	04 m aaa	ADDS	14 m aaa	XORS
00 0 26 m	INR	00 2 s4d	CPC	05 m aaa	SUBS	15 m aaa	ANDS
00 0 27 m	DIV	00 2 s5d	CAB	06 m aaa	MULS	16 m aaa	ORAS
00 0 30 m	DLD	00 2 s6d	CAX	07 m aaa	CALS	17 m aaa	JMPS
00 0 31 m	DST	00 2 s7d	CXB	10 0 0 dd	EXCI		
00 0 32 m	DADD	00 3 nnn	(trap)	10 0 1 dd	EXCA		
00 0 33 m	DSUB	00 4 ccc	SKT	10 0 2 dd	EXCB		
00 0 34 m	FADD	00 4 001	SS1	10 0 3 dd	EXCX		
00 0 35 m	FSUB	00 4 002	SS2	10 0 4 dd	EXCM		
00 0 36 m	FMUL	00 4 004	SS3	10 1 0 dd	SENS		
00 0 37 m	FDIV	00 4 010	SS4	10 1 1 dd	SENA		
00 0 40 m	LEA	00 4 020	SOF	10 1 2 dd	SENB		
00 0 41 n		00 4 040	SAP	10 1 3 dd	SENX		
⋮	(trap)	00 4 100	SAZ	10 1 4 dd	SENM		
00 0 77 n							

SPIRAS-65 INSTRUCTIONS

LOAD/STORE

MNEMONIC		DESCRIPTION	TIMING
LDA	a1	(e)→(A)	[3]
LDAS	a2		[2]
LDB	a1	(e)→(B)	[3]
[illegible]S	a2		[2]
[illegible]	a1	(e)→(X)	[3]
[illegible]XS	a2		[2]
STA	a1	(A)→(e)	[3]
STAS	a2		[2]
STB	a1	(B)→(e)	[3]
STBS	a2		[2]
STX	a1	(X)→(e)	[3]
STXS	a2		[2]
DLD	a1	(e), (e+1)→(A), (B)	[4]
DST	a1	(A), (B)→(e), (e+1)	[5]
LEA	a1	e→(X)	[4]

ARITHMETIC

MNEMONIC		DESCRIPTION	TIMING
ADD	a1	(A)+(e)→(A)	[3]
ADDS	a2		[2]
ADB	a1	(B)+(e)→(B)	[3]
ADX	a1	(X)+(e)→(X)	[3]
SUB	a1	(A)−(e)→(A)	[3]
SUBS	a2		[2]
MUL	a1	(B)*(e)→(A), (B)	[11]
MULS	a2		[10]
DIV	a1	(A), (B)/e→(B), rem→(A)	[15]
DADD	a1	(A), (B) ++ (e), (e+1)→(A), (B)	[5]
DSUB	a1	(A), (B) −− (e), (e+1)→(A), (B)	[6]
FADD	a1	(A), (B).+(e), (e+1)→(A), (B)	[11-28]
FSUB	a1	(A), (B).−(e), (e+1)→(A), (B)	[11-28]
FMUL	a1	(A), (B).*(e), (e+1)→(A), (B)	[60-70]
FDIV	a1	(A), (B)./(e), (e+1)→(A), (B)	[60-70]
INR	a1	(e)+1→(e)	[4]
DCR	a1	(e)−1→(e)	[4]

[illegible]P INSTRUCTIONS

MNEMONIC		DESCRIPTION	TIMING
SKIP		unconditionally skip 1 word	[1]
SAZ		skip 1 word if (A) = 0	[1]
SANZ		skip 1 word if (A) ≠ 0	[1]
SAN		skip 1 word if (A) < 0	[1]
SANN		skip 1 word if (A) ⩾ 0	[1]
SAP		skip 1 word if (A) > 0	[1]
SANP		skip 1 word if (A) ⩽ 0	[1]
SBZ		skip 1 word if (B) = 0	[1]
SBNZ		skip 1 word if (B) ≠ 0	[1]
SXZ		skip 1 word if (X) = 0	[1]
SXNZ		skip 1 word if (X) ≠ 0	[1]
SOF		skip 1 word if (OV) = 0	[1]
SNOF		skip 1 word if (OV) ≠ 0	[1]
SS1		skip 1 word if SS1 on	[1]
SNS1		skip 1 word if SS1 off	[1]
SS2		skip 1 word if SS2 on	[1]
SNS2		skip 1 word if SS2 off	[1]
SS3		skip 1 word if SS3 on	[1]
SNS3		skip 1 word if SS3 off	[1]
SS4		skip 1 word if SS4 on	[1]
SNS4		skip 1 word if SS4 off	[1]
SKT	n	skip 1 word if any conditions True	[1]
SKF	n	skip 1 word if all conditions False	[1]
CAS	a1	IF(R) < (e), don't skip	[3]
CBS	a1	IF(R) = (e), skip 1 word	[3]
CXS	a1	IF(R) > (e), skip 2 words	[3]
DRS	a1	(e) − 1 → (e), skip if (e) = 0	[4]
IXS	n	(X) = (X) + n, skip if (X) = 0 [n=0→511]	[2]
DXS	n	(X) = (X) − n, skip if (X) = 0 [n=0→511]	[2]

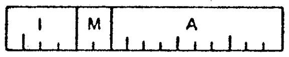
JUMP/CALL INSTRUCTIONS

MNEMONIC		DESCRIPTION	TIMING
JMP	a1	(e)→(P)	[3]
JMPS	a2	(e)→(P)	[2]
CALL	a1	*+2→e, e+1→(P)	[3]
CALS	a2	*+1→e, e+1→(P)	[2]

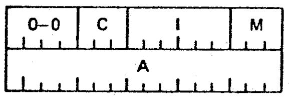
LOGICAL/CONTROL

MNEMONIC		DESCRIPTION	TIMING
AND	a1	(e) and (A)→(A)	[3]
ANDS	a2		[2]
XOR	a1	(e) eor (A)→(A)	[3]
XORS	a2		[2]
ORA	a1	(e) or (A)→(A)	[3]
ORAS	a2		[2]
HLT		Halt	[-]
NOP		No operation	[1.4]
OVF		n→OV,	[1]
SPF		1→Protect Flag	[1]

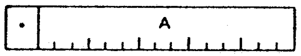
DATA WORD FORMATS



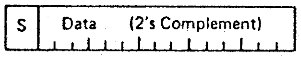
1-Word instruction



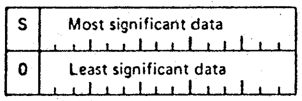
2-Word instruction



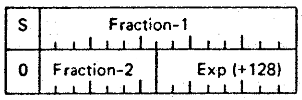
Indirect Address



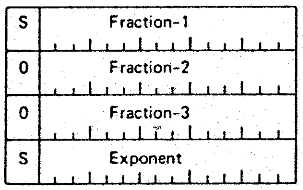
Single Precision



Double Precision

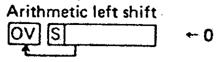
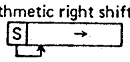
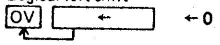
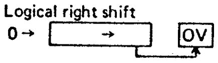
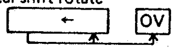
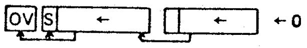
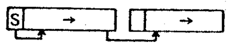
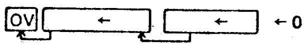
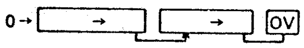
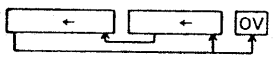


Floating Point



Double Precision Floating Point

REGISTER SHIFTING

MNEMONIC		DESCRIPTION	TIMING
ASLA	n	Arithmetic left shift	[1+.2n]
ASLB	n		
ASRA	n	Arithmetic right shift	[1+.2n]
ASRB	n		
LSLA	n	Logical left shift	[1+.2n]
LSLB	n		
LSRA	n	Logical right shift	[1+.2n]
LSRB	n		
LRLA	n	Logical shift rotate	[1+.2n]
LRLB	n		
ASLD	n		[1+.4n]
ASRD	n		[1+.4n]
LSLD	n		[1+.4n]
LSRD	n		[1+.4n]
LRLD	n		[1+.4n]

REGISTER TRANSFER

MNEMONIC		DESCRIPTION			TIMING
CP	s,d	[CPF]	(s)→(d)	[if OV = 1]	[1.4]
CPI	s,d	[CPIF]	(s)+1→(d)	[if OV = 1]	[1.4]
CPD	s,d	[CPDF]	(s)−1→(d)	[if OV = 1]	[1.4]
CPC	s,d	[CPCF]	$(\bar{s})$→(d)	[if OV = 1]	[1.4]
CPN	s,d	[CPNF]	-(s) →(d)	[if OV = 1]	[1.4]
CAB	s,d	[CABF]	(A)→(B), (s)→(d)	[if OV = 1]	[1.4]
CAX	s,d	[CAXF]	(A)→(X), (s)→(d)	[if OV = 1]	[1.4]
CXB	s,d	[CXBF]	(X)→(B), (s)→(d)	[if OV = 1]	[1.4]
RGC	nnn	Operation depends on bits nnn			[1.4]

s = Zero, A, B or X
d = Zero, A, B, X, AB, AX, BX or ABX

OUTPUT

MNEMONIC		DESCRIPTION	TIMING
EXCA	n	(A) → device n control	[1]
EXCB	n	(B) → device n control	[1]
EXCX	n	(X) → device n control	[1]
EXCM	n,a3	(e) → device n control	[3]
EXCI	n,v	v → device n control	[2]
OTA	n	(A) → device n data	[1]
OTB	n	(B) → device n data	[1]
OTX	n	(X) → device n data	[1]
OTM	n,a3	(e) → device n data	[2]
OTI	n,v	v → device n data	[3]
ARM		arm interrupts, 0 → OV	[1]
DRM		disarm interrupts, 0 → OV	[1]
ARMF		arm interrupts, 1 → OV	[1]
DRMF		disarm interrupts, 1 → OV	[1]

INPUT

MNEMONIC		DESCRIPTION	TIMING
SENA	n	device n status → (A)	[1]
SENB	n	device n status → (B)	[1]
SENX	n	device n status → (X)	[1]
SENM	n,a3	device n status → (e)	[3]
SENS	n,m	skip if device n status (masked by m) = 0	[2]
CIA	n	device n data → (A)	[1]
CIB	n	device n data → (B)	[1]
CIX	n	device n data → (X)	[1]
CIM	n,a3	device n data → (e)	[3]
INA	n	device n data OR (A) → (A)	[1]
INB	n	device n data OR (B) → (B)	[1]
INX	n	device n data OR (X) → (X)	[1]

ADDRESSING

A1

XXX	a(D)	e = a	[a = 0 → 65535]
[illegible]	[illegible]	a = (a)	
[illegible]	a(X)	e = a + (X)	
XXX*	a(X)	e = (a + (X))	
XXX*	a(Y)	e = (a) + (X)	
XXX	a(A)	e = a + (A)	
XXX	a(P)	e = a + (P)	
XXXI	a	operand = a	

A2

XXXS	a(D)	e = a	[a = 0 → 1023]
XXXS*	a	e = (a)	[a = 0 → 1023]
XXXS	a(X)	e = a + (X)	[a = 0 → 1023]
XXXS	a(P)	e = a + (P)	[a = -512 → +511]

A3

XXX	a(D)	e = a	[a = 0 → 65535]
XXX*	a	e = (a)	[a = 0 → 65535]
XXX	a(X)	e = a + (X)	[a = 0 → 65535]
XXX*	a(Y)	e = (a) + (X)	[a = 0 → 65535]