70|25 TRAINING MANUAL

SPECTRA 70

RADIO CORPORATION OF AMERICA • ELECTRONIC DATA PROCESSING

RCA

SYSTEM

70|25

TRAINING MANUAL

SPECTRA 70

RADIO CORPORATION OF AMERICA • ELECTRONIC DATA PROCESSING

SYSTEM 70/25

TRAINING MANUAL

RADIO CORPORATION OF AMERICA

70-25-801

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

First Printing: December, 1964
Second Printing: January, 1965

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

FOREWORD

70/25 TRAINING MANUAL

This manual is designed for use in formal training programs which may vary in length from about 15 classroom hours (with appropriate outside assignments and work sessions) to 45 hours or more, depending upon the experience of the student. People with good and recent programming experience may find the text helpful in self-study.

Principal references which should be used in either formal or self-study situations are:

1. 70/25 Assembly Manual
2. 70/25 System Reference Manual

GENERAL DESCRIPTION

INTRODUCTION

The RCA 70/25 is the intermediate member of the Spectra 70 Data Processing series. It is a powerful small-to-medium scale data processor. Equipped with communications gear, the 70/25 has high-speed, high-volume message switching or remote processing capabilities.

70/25 PROCESSOR

The RCA 70/25 Processor is a general-purpose, stored program, digital machine that includes High-Speed Magnetic Core Memory, Program Control, and the appropriate Input/Output logic for the Spectra 70 Systems standard Interface Unit.

HIGH-SPEED MEMORY

The High-Speed Memory (HSM) is a magnetic core device that provides storage and work area for programs and data. The memory capacity is either 16,384, 32,768, or 65,536 bytes. A byte is the smallest addressable unit in memory, and consists of eight information bits and a parity bit. Each byte location may be accessed with a 16-bit binary address consisting of two parts: a displacement carried in an instruction, and a base address stored in a general register. The sum of the two form an effective memory address.

The Memory Cycle is 1.5 microseconds, which is the time it takes to transfer four bytes from HSM to a memory register and to regenerate the bytes in storage.

PROGRAM CONTROL

The Program Control executes the instructions of the program stored in the HSM. An instruction can be interpreted and executed by the Program Control only after it has been brought out of HSM. The process of interpreting and placing the components of the instruction in the proper registers is called staticizing. An instruction is first staticized and then executed by the Program Control logic.

AUTOMATIC INTERRUPT

The RCA 70/25 can staticize and execute all instructions in one of two programming states; the Processing State and the Interrupt State. The Processing State is the normal mode of operation. A condition that causes interrupt will transfer the computer to the Interrupt State. Interrupt is mechanized in the 70/25 hardware. It automatically senses the presence of interrupt conditions, and transfers control to the Interrupt State.

INSTRUCTION COMPLEMENT

The RCA 70/25 Order Code consists of thirty-one instructions which can be divided into four classes.

1. DATA HANDLING

The data-handling instructions allow for the movement of data fields within HSM. Data may be moved without changing format or it can be packed, unpacked or edited for printing during the movement. A Translate instruction facilitates code conversion and data validation.

2. ARITHMETIC INSTRUCTIONS

This set includes Decimal Add, Subtract, Multiply and Divide instructions, as well as Binary Add and Subtract operations. It also incorporates the ability to perform Boolean Operations on bit structures.

3. DECISION AND CONTROL

The decision and control instructions allow for the comparing of both Decimal and Binary fields, and the branching to a location in HSM according to a Condition Code Indicator. Also included are Branch and Link and Branch-On-Count instructions that simplify subroutine linkage, and control of iterative coding.

4. INPUT/OUTPUT

Read and Write instructions transfer data between the processor and all peripheral equipment on-line to the 70/25. Included are the necessary instructions to control the devices and to recognize and recover from error conditions.

INSTRUCTION FORMAT

There are three basic instruction formats in the 70/25; six-byte, four-byte, and a two-byte instruction. The first byte of every instruction is the operation code. Depending on the instruction, the remaining bytes refer to field lengths, register and storage addresses, or contain peripheral device identification.

DATA FORMAT

The basic unit of storage is the byte, which can

represent, in the unpacked format, one alphabetic or numeric character, or two numeric digits in the packed format. Data is represented in HSM in the Extended Binary-Coded-Decimal Interchange Code (EBCDIC).

INPUT/OUTPUT

The RCA 70/25 communicates with peripheral devices through eight I/O channels.

Each peripheral device contains its own control electronics in order to transmit to the processor the status of the device, and any error conditions generated by an I/O command.

Each channel is a separate simultaneous mode, allowing execution overlap with other channels and the processor. An I/O termination interrupt is included in the system to facilitate efficient use of these powerful overlap capabilities.

HIGH-SPEED MEMORY

INTRODUCTION

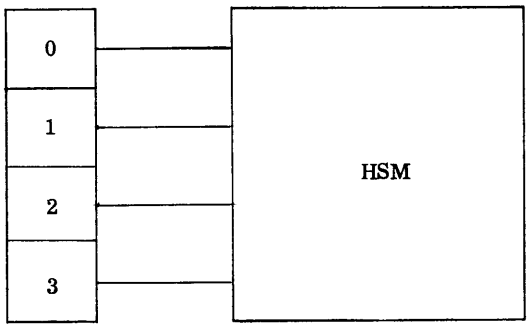
The RCA 70/25 magnetic core High-Speed Memory (HSM) may consist of one, two or four memory planes. Each plane contains $16,384_{(10)}$ byte locations (4 x 64 x 64 bytes). The byte is the smallest addressable unit in memory, and is made up of eight information bits and a parity bit.

BYTE

Bit Identification	P	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit (X = 0 or 1)	X	X	X	X	X	X	X	X	X

Four bytes of HSM may be transferred to a memory register and regenerated in memory within 1.5 microseconds. These four bytes are moved side by side or in parallel.

MEMORY REGISTER



To save processing time, the memory access hardware moves instructions and data in four byte units whenever possible, returning to a byte after byte or serial transfer when necessary to stay within limits defined by a specific operation. These four byte units are called words. The first four bytes of memory, locations 0, 1, 2, and 3, constitute the first word. The second begins with location 4, and the third with 8, etc. Even Word boundary is the term used to describe the initial byte of each word; locations 0, 4, 8, etc. The addresses contained in several 70/25 instructions must begin at even-word boundaries (see page 41).

HSM ADDRESSING

The address of each byte location is expressed as a binary number. Sixteen bits are required to address the highest location of a four plane system (65,536).

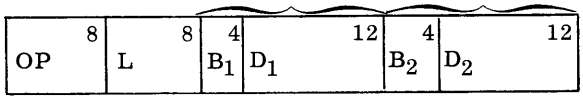
Examples:

	BINARY ADDRESS																DECIMAL EQUIVALENT
	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	25
2	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	109
3	0	0	0	0	0	0	1	1	0	1	1	0	1	1	1	1	879
4	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	4.094
5	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	16.512
6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	65,535

The first example shows the binary representation of HSM location 25. The conversion to decimal requires the adding of the 2^n value of all bits that are one (1).

BINARY	DECIMAL EQUIVALENT
2^0	1
2^3	8
2^4	16
	25

Within the 70/25 instruction format two bytes, 16 bits, are allocated for each memory address.



An address is divided into two parts: (1) a displacement of 12 bits contained in the instruction, and (2) a base address which is pre-stored in one of the fifteen General Registers.

The most significant four bits of each address, the B_1 or B_2 fields, designate the General Register containing the associated base address.

B FIELD

$0001_{(2)}$ - General Register 1

$1000_{(2)}$ - General Register 8

$1111_{(2)}$ - General Register 15

$0000_{(2)}$ - No base address

Assume that General Register One contains $40,000_{(10)}$.

OP	M	B_2	D_2
$47_{(16)}$	$F_{(16)}$	$0001_{(2)}$	$4000_{(10)}$

When an instruction is staticized the displacement is added to the base address. The absolute sum of the two is called the effective address, and is the address value actually used in execution. In the example above,

the displacement, $4000_{(10)}$

is added to the base address in register 1, $40000_{(10)}$

resulting in an effective address of $44000_{(10)}$

This technique makes it unnecessary to carry lengthy addresses within instructions. Each displacement is a fixed length of 12 bits. However, since the 16 least significant bits of general registers may be used for base address values, it is possible to access locations which require 13, 14, 15, or 16 bit addresses.

This addressing concept is a necessary feature in larger members of the Spectra 70 series where addresses may exceed 16-bit lengths.

The maximum value of a displacement is $4095_{(10)}$.

2,048	1,024	512	256	128	64	32	16	8	4	2	1	DECIMAL VALUE
2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	POWER OF TWO
1	1	1	1	1	1	1	1	1	1	1	1	BINARY ADDRESS

1
2
4
8
16
32
64
128
256
512
1024
2048
4095

When addressing locations between $0000_{(10)}$ and $4095_{(10)}$, no base address need be associated with a displacement. The 12-bit address carried in the D_1 or D_2 fields becomes a direct address when the value $0000_{(2)}$ is placed in the corresponding B_1 and B_2 fields.

HEXADECIMAL NUMBERING SYSTEM

The binary system, although efficient for the 70/25, is not a convenient notation for the programmer. The hexadecimal numbering system, which operates on the base sixteen, is a convenient method to express the binary representation of HSM addresses.

The decimal system is a numbering system based upon the number ten. It uses ten single symbols (0-9) to represent the basic digits. By a system of positional notation that indicates multiplication by powers of the base, any value can be expressed. The hexadecimal system requires sixteen symbols to express its basic digits. The alphabetic letters A through F have been assigned to represent the decimal values 10 through 15 in order to maintain single symbols for the digital values of the hexadecimal system.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	. . .
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	. . .

Each symbol in the hexadecimal system can be expressed by four bits in the binary system. Therefore, two hexadecimal marks are required to represent a byte, and four hexadecimal marks can express an HSM address.

HEXADECIMAL	BINARY	DECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Conversion of Hexadecimal to Decimal

The decimal number 472 represents:

$$4 \times 10^2 + 7 \times 10^1 + 2 \times 10^0$$

$$4 \times 100 + 7 \times 10 + 2 \times 1 = (472)_{10}$$

The binary number $(101101)_2$ can be converted to its decimal equivalence by:

$$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$32 + 0 + 8 + 4 + 0 + 1 = (45)_{10}$$

A hexadecimal number is converted to a decimal value by multiplying the hexadecimal characters by the appropriate value of 16^n.

Examples:

1. Convert $(1024)_{16}$ to Decimal

   $1 \times 16^3 + 0 \times 16^2 + 2 \times 16^1 + 4 \times 16^0$

   $4096 + 0 + 32 + 4 = (4132)_{10}$

2. Convert $(3AF)_{16}$ to Decimal

   $3 \times 16^2 + 10 \times 16^1 + 15 \times 16^0$

   $3 \times 256 + 10 \times 16 + 15 \times 1$

   $768 + 160 + 15 = (943)_{10}$

The first example shows the hexadecimal address $(1024)_{16}$ which has a decimal value of $(4132)_{10}$. The actual machine (binary) address is:

0001000000100100

Each hexadecimal character can be represented by four bits. Therefore, hexadecimal is converted to binary by replacing each hexadecimal character with its binary value.

$$(|1|0|2|4|)_{16} = (|0001|0000|0010|0100|)_2$$

$$(0001000000100100)_2 = 4096 + 32 + 4\ (4132)_{10}$$

The second example shows that the hexadecimal address 3AF has a decimal value of 943.

$$(|3|A|F|)_{16} = (|0011|1010|1111|)_2 = (943)_{10}$$

Exercise:

1. A byte consists of_____information bits and a_____bit, and is the_____addressable unit in the 70/25 HSM.

2. An effective HSM address is the absolute sum of a_____ and a_____.

3. Base address values are stored in_____. The _____and/or_____fields of an instruction specify which base address will be used to compute an effective address.

4. The decimal value of a displacement may not exceed_____.

5. Convert following hexadecimal numbers to binary:

   a. $A4E8_{(16)}$

   b. $E82C_{(16)}$

   c. $3D71_{(16)}$

6. Convert following hexadecimal numbers to decimal:

   a. $B5F9_{(16)}$

   b. $F93D_{(16)}$

7. Convert following binary numbers to hexadecimal:

   a. $1100011000001010_{(2)}$

   b. $0000101001001110_{(2)}$

   c. $0010110001100000_{(2)}$

8. Convert following decimal numbers to hexadecimal:

   a. $55067_{(10)}$

   b. $7007_{(10)}$

DATA AND INSTRUCTION FORMAT

DATA FORMATS

When representing data, a byte may store a single character (unpacked format), or two numeric digits (packed format).

UNPACKED FORMAT

A byte in the unpacked format uses all eight bits to represent one alphabetic or numeric character. This format, for example is required for the storage of any characters that are to appear on any type of display output such as the Printer or Typewriter.

Some of the more commonly used characters, and the hexadecimal representation of their bytes are as indicated in the tables below.

ALPHABETIC						NUMERIC	
Char.	Hex.	Char.	Hex.	Char.	Hex.	Char.	Hex.
						0	F0
A	C1	J	D1			1	F1
B	C2	K	D2	S	E2	2	F2
C	C3	L	D3	T	E3	3	F3
D	C4	M	D4	U	E4	4	F4
E	C5	N	D5	V	E5	5	F5
F	C6	O	D6	W	E6	6	F6
G	C7	P	D7	X	E7	7	F7
H	C8	Q	D8	Y	E8	8	F8
I	C9	R	D9	Z	E9	9	F9

SPECIAL CHARACTERS			
Char.	Hex.	Char.	Hex.
BLANK	E0	- (Minus Hyphen)	60
. (Period)	4B	/	61
<	4C	, (Comma)	6B
(	4D	%	6C
+	4E	#	7B
&	50	@	7C
$	5B	' (Quote)	7D
*	5C	=	7E
)	5D	Space	40

A decimal numeric field in unpacked format is assumed to contain a sign in the high-order four bits of the right most byte. All other bytes, in the zone portion, will have the four high-order bits a value of all ones (1111_2).

However, the decimal numeric field must be packed before it may be used as an operand in a decimal arithmetic operation.

PACKED DATA FORMAT

In packed data format, one byte stores two decimal digits except for the rightmost byte which contains the sign in the four low-order bits.

The following example shows the same field in unpacked and packed format. Each location represents a byte shown in hexadecimal format.

UNPACKED	F0	F3	F1	F6	F2	F1	S0

PACKED	03	16	21	0S

It should be noted (as in the example above) that when either packing or unpacking a field the rightmost byte has its zone and numeric portions reversed.

SIGN RECOGNITION

In decimal arithmetic operations the sign of a field is recognized as positive if the sign position contains:

(1) All one bits $(1111)_2$

(2) Or if the rightmost bit is a $(0)_2$ i.e., $(1010)_2$, $(1110)_2$.

If the sign has a low-order bit of $(1)_2$, and at least one of the remaining bits is $(0)_2$, it is considered negative.

After a decimal arithmetic operation the sign of the result is one of the following:

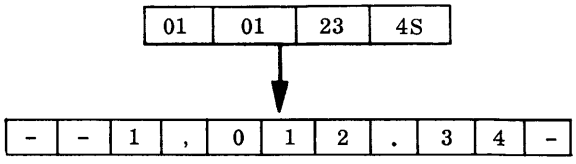
$(1100)_2$ for positive

$(1101)_2$ for negative

Thus, in preparing source card input for numeric data fields, the user may follow existing procedures, i.e., for a negative field an overpunch of the minus (11 punch) in the least significant position generates a zone portion of (1101_2).

EDITED FORMAT

A packed numeric field may be placed in edited format with a single EDIT instruction (see page 33). A field in edited form is unpacked and contains necessary edit symbols.

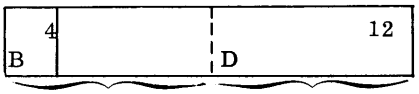


MACHINE INSTRUCTION FORMAT

The 70/25 instruction format is variable in length. An instruction may contain either two, four, or six bytes.

The first byte of each instruction is an operation code. The format of the second byte varies from one instruction to the next. In some instructions it is used as a binary length counter (L). In others, the byte is divided into two length counters of four bits each (L_1, L_2). In still others, it is used to hold a mask (M), or one or more General Register numbers (R), (R_1-R_3). The second byte of a I/O command contains a trunk and device designation.

The third and fourth bytes hold the address displacement (D) and the number of the General Register (B) which contains the base address to be associated with that displacement.

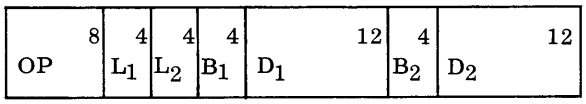


3rd byte 4th byte

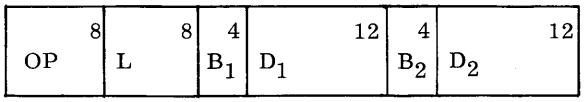
In a two-address instruction the fifth and sixth bytes constitute the B and D field of the second address.

The machine formats and the type of instructions using each format are shown below:

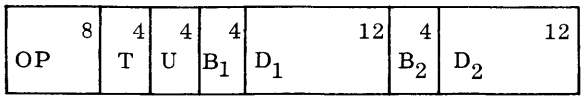
SIX-BYTE INSTRUCTIONS



Binary Arithmetic
Decimal Arithmetic
Decimal Comparison
Packing and Unpacking

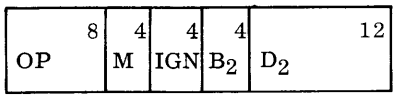


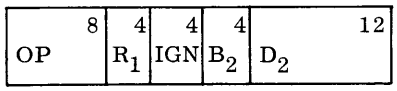
Data Movement
Logical Operations (And, Or, Excl. Or)
Logical Comparison
Data Editing



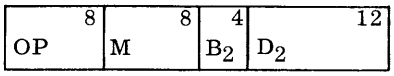
Input/Output

FOUR-BYTE INSTRUCTIONS

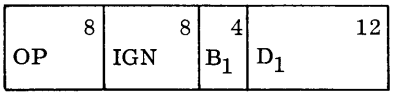




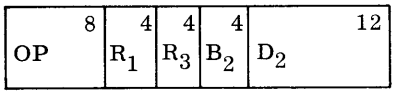
Conditional and Unconditional Branch



Test Under Mask



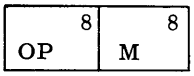
Set P2 Register



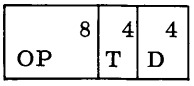
Load Multiple
Store Multiple

IGN: These bits are not used (ignored) by the instruction.

TWO-BYTE INSTRUCTIONS



Halt



Input/Output (Post Status)

True and False Exercise

T F 1. Data Edited for display purposes may be in packed format.

T F 2. A numeric field in unpacked format is assumed to contain an $(F)_{16}$ in the high order four bytes of each byte.

T F 3. When packing or unpacking a field, the rightmost byte has its zone and numeric portions reversed.

T F 4. The values $(1101)_2$ and $(1001)_2$ are valid negative signs.

T F 5. The B_1 or B_2 fields of machine instruction format contain the HSM address of a general register.

T F 6. Each displacement field accommodates a 12 bit address.

T F 7. An instruction is variable in length; either two, three, four, or six bytes.

INTERRUPT

INTRODUCTION

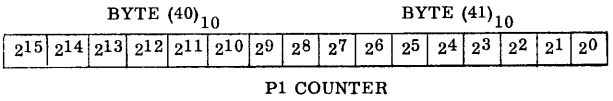
An interrupt facility provides an automatic means for the detection of exceptional conditions, and a method for an immediate program response. The function of sensing for exceptional conditions and the automatic transfer of control to software has been mechanized in the RCA 70/25 hardware. Combining software with the hardware interrupt makes it unnecessary to halt the computer when an error develops, and eliminates program sensing of external demands. This system allows the user to program a response independently of his production processing.

PROGRAMMING STATES

All instructions are executed in one of two states: (1) the Processing State (P1), or (2) the Interrupt State (P2). The Processing State is the normal mode of operation. An interrupt causes the computer to transfer from the Processing State to the Interrupt State where it remains until instructed to return to the original Processing State.

PROCESSING STATE

During the execution of instructions in the P1 state the address of the next instruction to be executed is stored in the P1 counter (reserved HSM forty $(28)_{16}$ and forty-one $(29)_{16}$.

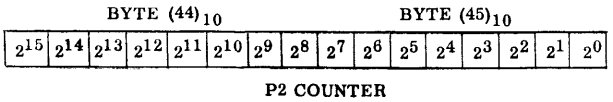


P1 COUNTER

Each time an instruction is staticized in the P1 state the contents of the P1 counter is updated to contain the address of the next instruction. All thirty-one instructions may be executed in the P1 state. The computer remains in this state until an interrupt occurs.

INTERRUPT STATE

When an exceptional condition is detected, and an interrupt initiated, the hardware transfers control to the instruction whose address is stored in the P2 counter (reserved HSM forty-four $(2C)_{16}$ and forty-five $(2D)_{16}$).



P2 COUNTER

The system is now in the Interrupt State. Each time an instruction is staticized the contents of the P2 counter is updated to contain the address of the next instruction to be executed. All thirty-one instructions may be executed in the P2 state, and the computer remains in this state until a STPP2 instruction (see page 39) is executed. The STPP2 instruction resets the P2 counter to its original value, and returns Control to P1. The Interrupt State is not interruptable. Any interrupt attempted will be "PENDING" until the computer returns to the Process State.

Interrupt occurs only after the termination of an instruction. Therefore, when the system returns to the Process after interrupt, the P1 counter holds the address of the instruction that immediately follows the point where interrupt took place. This automatic linkage permits the user to disregard interrupt considerations when programming his process.

TYPES OF INTERRUPT

There are four conditions that can interrupt the Processing State:

1. I/O Device (Manual or Termination)
2. Operation Code Trap
3. Arithmetic Overflow or Divide Exception
4. Elapsed Timer Overflow

I/O INTERRUPT

An interrupt occurs after the termination of each Input/Output Command. A termination interrupt indicates one of two possible terminating conditions:

1. The I/O instruction was not completed successfully (ERROR). In this case, the Secondary Indicator bit in the Standard Device Byte is $(1)_2$ (see page 53).
2. The channel and device that executed the instruction is now free, and ready to receive the next command (NORMAL TERMINATION). In this event, the 2^6 bit of the Standard Device Byte is set to $(1)_2$, (see page 53).

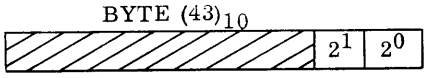
The purpose of normal termination interrupt is to notify system software that an I/O channel is available. With this knowledge, the software can use

efficiently the overlap capabilities of a system containing eight I/O Channels.

A communications device request, or a request for control by the operator at the console typewriter also generates an I/O Interrupt. Console request interruption is distinguished by the fact that the 2^7 bit of the Standard Device Byte is set to $(1)_2$.

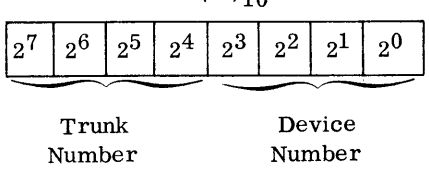
Prior to entering the P2 state, the computer automatically:

1. Stores the state of the Condition Code Indicator, The present value of the Condition Code is stored in the 2^0 - 2^1 bits of the reserved HSM location forty-three $(2B)_{16}$.



The Condition Code Indicator is then set to $(00)_2$.

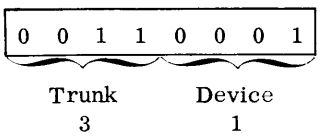
2. Stores the identification (Trunk and Device Number) of the interrupting device in the reserved HSM location forty-seven $(2F)_{16}$. The Device Number is stored in the 2^0- 2^3 bits, and the Trunk Number is stored in the 2^4 - 2^7 bits.



3. Stores the Standard Device Byte for the Interrupting device in the reserved HSM location forty-six $(2E)_{16}$.

See page 53 for a description of the Standard Device Byte.

The P2 counter contains the address of the first instruction of a routine to be executed when interrupt occurs. This routine tests the Condition Code (with a Branch On Condition instruction). A setting of $(00)_2$ indicates that interrupt had been caused by an I/O device. The Trunk and Device Number have been stored in a reserved area of HSM, allowing the routine to identify the device that caused the interrupt. For example, if the Console Typewriter is Device one on Trunk three, and the Interrupt button had been depressed, then HSM location forty-seven would contain:



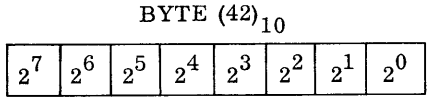
OPERATION CODE TRAP

If an instruction is staticized in which the Operation code is not one of the thirty-one legitimate codes, an interrupt is initiated. This interrupt is called an Operation Code Trap.

Prior to entering the P2 state, the computer automatically:

1. Stores the state of the Condition Code Indicator in the 2^0 - 2^1 bits of location forty-three $(2B)_{16}$.

2. Stores the illegal operation code that caused the interrupt in the reserved HSM location forty-two $(2A)_{16}$.



The two high-order bits of the Operation Code indicate the length of the instruction.

00	=	two-byte instruction
01 or 10	=	four-byte instruction
11	=	six-byte instruction

3. Sets the Condition Code to $(01)_2$. The interrupt routine tests the Condition Code. A setting of $(01)_2$ indicates that the interrupt was caused by an illegal operation code in the instruction previously staticized in the P1 state. Depending on the situation, the illegal operation could actually be an error, or an intentional interrupt. In the latter case, the interrupt could simulate an instruction that is not part of the 70/25 order code. For example, the 70/45 operation code $(4E)_{16}$ for Convert Decimal would cause an interrupt on the 70/25. However, the decimal conversion could be simulated by instructions in the P2 state.

ARITHMETIC OVERFLOW AND DIVIDE EXCEPTION

A carry out of the high-order position of the first operand during the execution of an Add Decimal $(FA)_{16}$ or a Subtract Decimal $(FB)_{16}$ instruction causes interrupt. If the operands of a Divide Decimal $(FD)_{16}$ operation are not properly edited, an interrupt occurs.

Hardware stores the state of the Condition Code in the 2^0 - 2^1 bits of reserved location forty-three $(2B)_{16}$, and resets the code to $(10)_2$, before transferring to the Interrupt State.

ELAPSED TIMER INTERRUPT

General Register Zero serves as an elapsed time clock. Every 16-2/3 milliseconds (using 60 cycle power) the power supply generates a $(1)_2$ bit that is added to the contents of Register Zero. When the register overflows, interrupt takes place. The time intervals between interrupts is controlled by the value pre-stored in the register (see page 13).

Before transfer to the P2 state, the current setting of the Condition Code is stored in the 2^0 - 2^1bits of reserved location forty-three $(2B)_{16}$, and the code reset to $(11)_2$.

INHIBITING INTERRUPT

All interrupts except the Operation Code Trap may be inhibited. Reserved HSM location forty-nine $(31)_{16}$ allows the user to inhibit interrupt on all or selected I/O channels. The user places a mask into the eight rightmost bit positions of the reserved location. The bit positions, $(2^0$-$2^7)$, correspond to the eight I/O channels, 0-7. A $(1)_2$ bit permits interrupt and a $(0)_2$ bit inhibits it.

LOCATION 49 $(31)_{16}$

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

A mask of 10010110 allows channels one, two, four and seven to interrupt, and inhibits interrupt from channels zero, three, five, and six.

If an interrupt on an I/O channel is inhibited, the channel remains busy until a Post Status instruction, addressed to that channel, is executed (see page 53).

Three bit positions $(2^2$-$2^0)$ in reserved location forty-eight $(30)_{16}$ allow the user to inhibit the Elapsed Timer, Arithmetic Overflow, and MULTIPLEX CHANNEL interrupts.

	T	O F	M P X
	2^2	2^1	2^0

2^1=Overflow
2^2=Timer
2^0=Multiplex Channel

A mask of 101 in the 2^2-2^0 allows Timer and Multiplex interrupt, but inhibits interrupt caused by arithmetic overflow or divide exception.

INTERRUPT PRIORITIES

Op Code Trap -	immediate
I/O	1
Elapsed Timer	2
Overflow and Divide Exception	3

Exercise:

T F 1. Only fifteen of the thirty-one 70/25 instructions can be executed in the Interrupt State.

T F 2. The main program is executed in the Processing State.

T F 3. The Processing State is not interruptible.

T F 4. The Interrupt State is not interruptable.

T F 5. The Condition Code is stored prior to changing states.

T F 6. The Condition Code is always set to 00 prior to going into the P2 state.

T F 7. The two program counters are stored in the reserved area of memory.

T F 8. The Processing State uses only one counter to indicate the address of the next instruction.

T F 9. The P1 counter is destroyed by the interrupt.

T F 10. The computer remains in the P2 state until another interrupt occurs.

T F 11. The operation code is stored on an Operation Code Trap.

T F 12. The Standard Device Byte is stored on an Operation Code Trap.

T F 13. Interrupt from any I/O device is the only interrupt that can be inhibited.

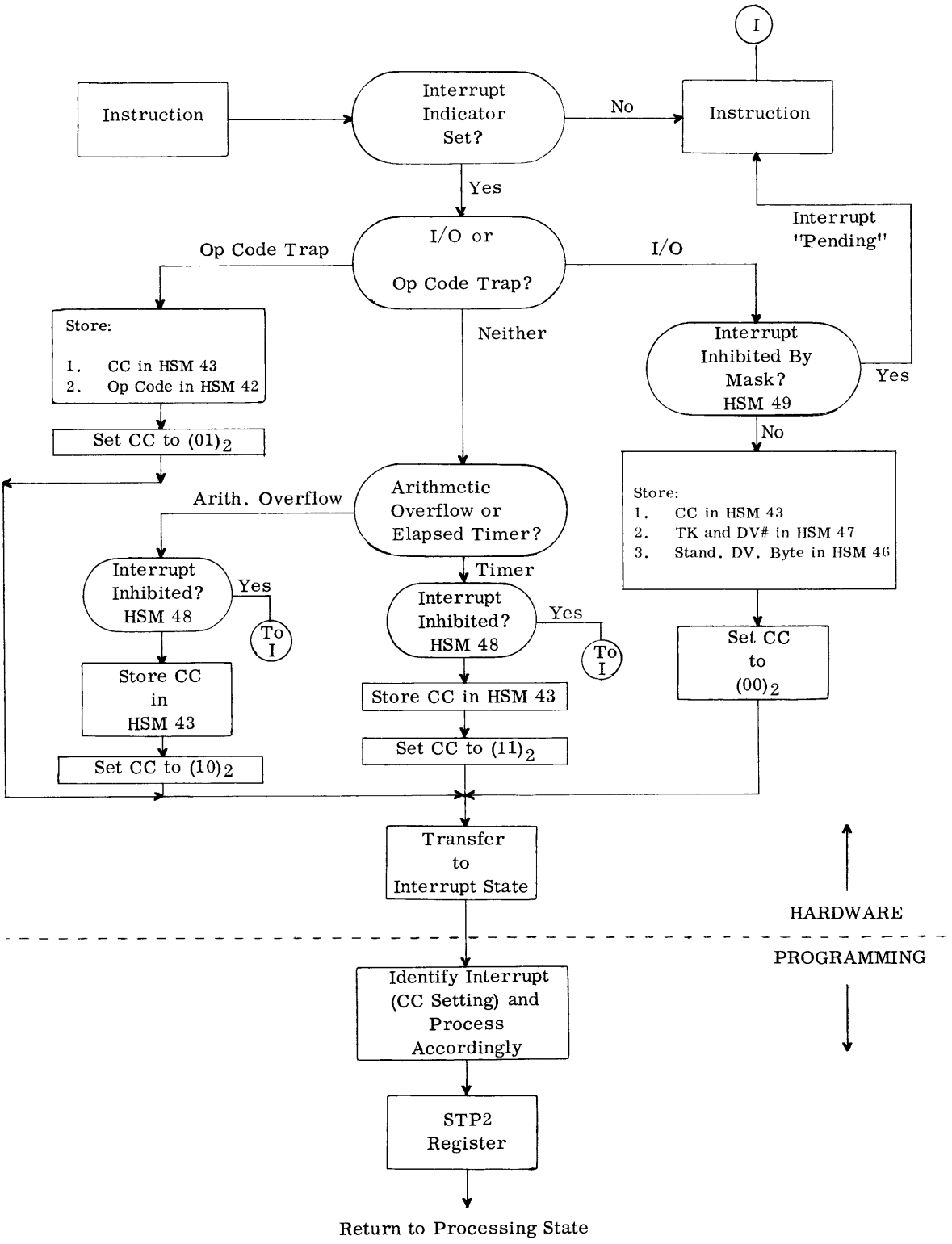
14. Describe the use of HSM location 49.

15. Describe two uses of the Operation Code Trap.

16. Write the masks necessary to inhibit all possible interrupts. Where must they be stored?

17. Describe what is stored in reserved memory when each of the four types of interrupt takes place.

SUMMARY OF 70/25 INTERRUPT LOGIC



ELAPSED TIME CLOCK

The least significant 24 bits of Register Zero, the first General Register, may serve as an elapsed time clock. The 70/25 power supply generates a $(1)_2$ bit every 16-2/3 milliseconds (60 cycle power). This bit is added to the contents of Register Zero. When register overflow develops, an interrupt is initiated (see page 11). The programmer may control the time interval between these interrupts by the selection of the value stored in the register.

A $(1)_2$ is added to the low order bit of the register as follows:

50 CYCLE POWER	60 CYCLE POWER
1 ADD EVERY 20 MILLISECONDS	1 ADD EVERY 16-2/3 MILLISECONDS
50 ADDS EVERY SECOND	60 ADDS EVERY SECOND
3000 ADDS EVERY MINUTE	3600 ADDS EVERY MINUTE
180000 ADDS EVERY HOUR	216000 ADDS EVERY HOUR

If the Timer is set to a value of all one bits $(16,777,215)_{10}$, the first add causes overflow. If the Timer contains all zeros, overflow will take place approximately 93 hours later, using 50 cycle power, or 77 hours later using 60 cycle power.

The number of adds required to clock off more meaningful time intervals are indicated below:

60 CYCLE POWER

16-2/3 MILLISECONDS	=	1 ADD
1 SECOND	=	60 ADDS
30 SECONDS	=	1800 ADDS
1 MINUTE	=	3600 ADDS
30 MINUTES	=	108,000 ADDS
1 HOUR	=	216,000 ADDS

The overflow value of the 24 bit Timer is $16,777,216_{10}$. Let us assume we wish to generate an interrupt every minute. By subtracting 3600_{10}, the number of adds executed in a minute, from the overflow value, we can determine the amount to be stored in the register.

$$\begin{array}{r} 16,777,216_{(10)} \\ -\ \ 3,600_{(10)} \\ \hline 16,773,616_{(10)} \end{array} \qquad \text{to} \qquad \text{FFF1EF}_{(16)}$$

It should be remembered that the timer contents is reduced to zero at the point of overflow. As long as the initial value is added to the register contents before the computer returns to the Processing State, no time loss results.

Register Zero may not be used for general storage purposes. Even though interrupt has been inhibited (the 2^2 bit of reserved HSM location 48 is $(0)_2$), the addition of $(1)_2$ bits to the register contents continues.

Exercise:

If we want interrupt after 5 minutes and 30 seconds, what value should be stored in register zero?

INTRODUCTION TO THE RCA 70/25 ASSEMBLY LANGUAGE

FORMAT REQUIREMENTS

The RCA 70/25 Assembly is an automatic programming system designed to translate a symbolic machine-oriented program into a machine-coded program for subsequent execution on the RCA 70/25 system. The source language consists of one-line statements written on the RCA Spectra 70 Assembly Program Form. Each single-line statement performs one of the following functions:

1. Generates an object program instruction.
2. Allocates data areas or constants.
3. Notifies the assembler to perform a specific function.

OPERATION FIELD

Every statement, except a line used solely for an output listing comment, must have an entry in the OPERATION field (Cols. 10-14) specifying one of the above three functions.

NAME FIELD

The NAME field (Cols. 1-6 only) may be used when it is desired to symbolically identify the leftmost location of the field generated by the statement. The NAME entry symbol must consist of at least one alphabetic (A-Z) character followed by any combination of alphabetic and/or numeric (0-9) characters that do not exceed a total of six characters. The only exception to the symbol entry above is that an asterisk may appear in Col. 1 if the statement line is to be used for an output listing comment.

OPERAND FIELD

The OPERAND field has entries as required by the OPERATION field. Thus, if the OPERATION field specifies that a constant is being defined, the OPERAND field entry is the value of the constant. If an instruction Operation Code appears, the OPERAND field must follow the prescribed format for that particular instruction.

COMMENTS FIELD

A comment may appear in any statement line following the OPERAND entry. It must be separated from the required OPERAND entry by at least one blank column. The entire statement line (to Col. 71) may be used for a comment if an asterisk appears in Column 1.

IDENTIFICATION FIELD

The contents of the IDENTIFICATION field has two functions. In the START statement, the first four positions, columns 73-76, may contain a name to be assigned to the object program. If the last four positions, columns, 77-80, are numeric, the contents is used as the initial setting of the Assembly sequence counter. If not numeric, the counter starts at all zeros. Each object instruction has a sequence number either derived from the value in columns 77-80 or from zeros.

ADDRESSING

A symbolic name is the most frequently used means of addressing and referencing a location. When a symbol has been used in the NAME field to define a location, it may be referenced as frequently as desired in the OPERAND field. The value assigned is the address of the left end of the data field or instruction on the 'NAMEd' line of assembly coding.

As stated previously, the symbol may be any combination of the alphabetics (A-Z) or numerics (0-9). There are two restrictions: (1) no name may exceed six characters, (2) the first character must be alphabetic.

The following are examples of valid and invalid symbols:

VALID

```
A1
STKNK1
C
IN1
```

INVALID

OPN	(Space invalid character)
BEGINERR	(Too many characters)
1A	(First character not alphabetic)
IN.1	(Period invalid character)

The Assembler builds a table containing all the symbolics that appear in the name field. A specific HSM address is assigned to each symbolic. The LOCATION COUNTER, a program counter maintained by the Assembler, generates these addresses and makes assignments. Assume a routine is to begin at HSM location 2000.

The Controlling Code,

NAME	OPERATION	OPERAND
	START	X'7DØ'

places the initial value of $(2000)_{10}$ in the Location counter. As memory is allocated for fields and instructions, the counter is incremented so that it always contains the address at the next location available. If a statement contains a name, that name is placed in the table, and assigned an address equal to the current value in the counter.

Consider the following examples:

ASSEMBLY STATEMENT	CONTENTS OF LINE	SYMBOL (TAG) ASSIGNED	ADDRESS ASSIGNED	FOR THIS LINE ADVANCE LOCATION COUNTER TO:
1	A 5-byte field	Work	2000	2000
2	A 2-byte field	ADDR	2005	2005
3	A 10-byte Constant	WCON	2007	2007
4	A 6-byte Instruction	STRT	2018	2018 *
5	A 4-byte Instruction			2024 **

*Note that the Location Counter is advanced one byte location by the Assembler to orient the instruction to an even address.

**NO SYMBOL (TAG) ASSIGNED.

IMPLICIT BASE ADDRESS SYSTEM

The User may explicitly state base register values in his Assembly Statements, or he may ask the Assembler to assign base addresses and compute displacements. This latter method is called the implicit Base Address System. Base values are considered to be implied whenever they are not explicitly stated.

NAME	OPERATION	OPERAND	
	MVC	$D_1(L, B_1)$, D_2 (B_2)	Explicit Base Addresses
	MVC	$S_1(L)$, S_2	Implied Base Addresses

We tell the Assembly what base addresses to use, for implicit assignment, through a series of Using Statements.

NAME	OPERATION	OPERAND	
	START	X'064'	Set Location Counter to $100_{(10)}$
	USING	*,9	Using $100_{(10)}$ in General Register 9
	USING	4195,10	Using $4195_{(10)}$ in General Register 10
	USING	*+8190,11	Using $8290_{(10)}$ in General Register 11

With this information, the Assembly selects the base-address that gives the least displacement, and computes the B_1-D_1 (or B_2-D_2) field in the object instruction.

Assume the two names ABLE and BAKER have been assigned the addresses $3850_{(10)}$ and $8173_{(10)}$ in the symbol table. The assembly subtracts a smaller base value from the effective (Symbol Table) Address. The difference is the displacement.

3850	8173
0100	4195
3750	3978

If the displacement exceeds 4095, the statement is flagged.

If we move ABLE to BAKER the object result is:

OP	L	B_1	D_1	B_2	D_2
$D2_{(16)}$	$00_{(16)}$	$9_{(16)}$	$3750_{(10)}$	$A_{(16)}$	$3978_{(10)}$

Additional rules for implicit address generation:

1. If more than one register produces a valid displacement (not over $4095_{(10)}$), the register whose contents produce the smallest displacement is used.

2. If two or more registers produce the same displacement the highest numbered register is used.

3. If no register produces a valid displacement the object instruction contains an OP code and zeros. The statement is flagged.

4. The Using Statement is an Assembly Controlling Code. The User must write additional instructions and constants to physically load and manipulate register contents.

5. Address values to be stored in general registers for base address purposes should be defined with an address constant controlling code (DC, A option). If other means are used, the program block will not be relocatable. Float factors are added to address constants, not to the displacement values.

RELATIVE ADDRESSING

As mentioned above, a symbol appearing in the NAME field has an address assigned by the Location Counter. The assignment will be the address of the leftmost byte of the defined field.

The programmer may reference any location to the right or left of this address by indicating a plus (+) or minus (-) value.

As an example, assume a field (WARE) has been assigned as follows:

WARE

	00 01 02 03
30	

The programmer may refer (in the OPERAND field) to the right-end (3003) of this field as:

WARE+3

SELF-RELATIVE ADDRESSING

The asterisk, as the first character of an operand, specifies the current value of the location counter as the address. The address is always the leftmost byte generated by the statement line. Thus, the asterisk, with a plus or minus value, can address a position to the right or left of the first byte generated by the statement line.

Assuming the location counter value is 2000 for a given statement line, *+6 generates an address of 2006 and *-3 furnishes an address of 1997. An asterisked address is relocatable.

SELF-DEFINING VALUES

In the previous example, a self-defining value of 3 incremented a symbolic address.

Self-defining values may be in three forms; decimal, hexadecimal, and character. They may modify addresses, express masks and lengths, and represent I/O trunk and device numbers. Self-defining values may also be used for location addresses. When used for this purpose, they should not exceed $4095_{(10)}$.

DECIMAL

A one to six decimal digit number may be used. The Assembler converts it to the binary equivalent.

Example:

OPERAND
ABLE(4),

Four (4) used to define length of ABLE

OPERAND
0049,

Forty-nine to address location 49_{10} (Interrupt Mask)

HEXADECIMAL

Up to six hexadecimal digits may be written as a self-defining value by enclosing the digits in single quote marks preceded by an X. This option is used to represent binary configurations such as masks.

Examples:

OPERAND
X'3F',

Represents the binary configuration 0011 1111.

CHARACTER

A character may be specified by enclosing it in single quote marks preceded by a C.

Example:

OPERAND
C'A',

The character A (or in binary 1100 0001) is desired.

Example: The three statements below generate the same value:

OPERAND		
C'A'	CHARACTER	All will generate $1100\ 0001_2$
X'C1'	HEXADECIMAL	
192,	DECIMAL	

EXPRESSIONS

An expression is a symbol or a self-defining value, or a combination of the two, written in the operand field of an Assembly statement. A simple expression contains one factor.

NAME	OPERATION	OPERAND
	START	X'ABC'
	MVC	ABLE(3), BAKER
	AP	X1(6), 120(4)

The compound expression is made up of two or three simple expressions.

NAME	OPERATION	OPERAND
	MVC	*⊖55(3), BAKER*ABLE
	AP	SUM1⊖SUM2+66(6), *+TANG⊖5

Expressions are further divided into two additional classifications, absolute and relocatable. An absolute expression generates an object machine address that is fixed, and may not be legitimately changed. The address generated by a relocatable expression is relative to the starting point of the program segment and may be altered when coding blocks are relocated in memory.

Assembly rules for the formation of compound expressions must be followed closely. Otherwise, absolute addresses may be generated where relocatable ones are required.

SUMMARY OF RULES FOR FORMING EXPRESSIONS

A simple expression is a single symbol, or one self-defining value used as an operand.

A compound expression is an arithmetic combination of at least two, but not more than three simple expressions. The expressions may be compounded with addition (+), subtraction (-), or multiplication (*).

*+50 ABLE⊜STARTP 15*6+3

Compound expressions must not begin or end with an arithmetic operator. Simple expressions within compound ones must be separated with one and only one operator.

The following are incorrect:

⊜ROUT ABLEBAKER SIZE++PRICE

An expression becomes absolute if it contains only absolute symbols and/or self-defining values. It is also absolute if it has one of the following forms:

R_1-R_2

R_1-R_2+A

R_1-R_2-A

R1, R2 = relocatable symbols

A = absolute symbol or self-defining value

Thus, the following are all absolute expressions:

X'3X', 168, R1-R2⊜603, 5*5+1, R1-R2+37, 39*x'H4'+2

Relocatable expressions must conform to the rules stated below:

1. An expression must contain either one or three relocatable symbols.
2. If there is one relocatable symbol, it must not be preceded by a subtraction (-) operator.
3. If three relocatable symbols are present, one and only one may be preceded by a subtraction operator.
4. Relocatable symbols may not be compounded with the multiplication operator. Only absolute expressions are legitimate operands in multiplication.

The following are examples of correct relocatable expressions:

DOG, DOG⊜103, *⊜10.

DOG+CAT⊜FIGHT, ABLE+437*6

Illegal Expressions:

1.	DOG+CAT	Contains two relocatable symbols
2.	50*HOPE	Multiplication of relocatable symbol
3.	DOG+CAT+HOPE	No subtraction operator
4.	176 - DOG	Single relocatable symbol preceded by
5.	DOG - CAT - HOPE	Two subtraction operators

IMPLIED LENGTHS

The length of an operand may be implied by omitting any reference to length in an Assembly statement.

NAME	OPERATION	OPERAND	
	MVC	ABLE(3), BAKER	Explicit Length
	MVC	ABLE, BAKER	Implied Length

In line two of the example above, the number of bytes moved from location BAKER to ABLE is equal to the number allocated when the name BAKER was defined. If BAKER is the name of a 3 byte

NAME	OPERATION	OPERAND
BAKER	DS	CL3

storage area, the implied length is three.

An implied length that exceeds the value permitted in an instruction is flagged, and the object length field is set to zeros.

If a name is defined in a statement using an asterisk or a self-defined-value, the implied length is one.

When a compound expression is used as an operand,

NAME	OPERATION	OPERAND	
	MVC	ABLE, A+B-C	Implied Length is Length of A

the implied length is the length assigned to the left-most factor in the expression.

ASSEMBLER CONTROLLING CODES

The DS (Define Storage) code allocates and reserves working storage and input/output areas.

The number of units of memory to be reserved, followed by the letters C, H, or F (byte, halfword, or full word), appears in the OPERAND field.

A symbol appearing in the NAME field is assigned the address of the leftmost byte of the reserved area.

The Location Counter may be set to any desired value with the ORG code. This code sets the location counter to the value appearing in the operand field . The operand may be a symbol or an asterisk (incremented or decremented) or a self-defining value; if a symbol, then it must have been previously defined in the NAME field. With this tool, areas may be allocated beginning at any desired location, or may be re-allocated to accommodate varying formats.

NAME	OPERATION	OPERAND
READIN	DS	80C

The character "C", following the area length 80, tells the Assembler to allocate 80 consecutive one-byte fields beginning at the current position of the location counter.

The statements

	NAME	OPERATION	OPERAND
1.	READIN	DS	20F
2.	READIN	DS	40H

allocate the same amount of memory. However, the first example (20F) instructs the Assembler to advance the location counter to the next word boundary. and then allocate 20 fields of four bytes each; whereas the second example (40H) advances the location counter to the next even byte address, and then allocates 40 fields of two bytes each.

Example (ORG and DS Codes)

Assume a read-in area is allocated for a file that contains transactions in three different formats. The maximum size record is 80 characters. We want to name the area, and to name every field within each format.

NAME	OPERATION	OPERAND	
INAR	DS	80C	
	ORG	INAR	
NACN	DS	10C	
NCOD	DS	2C	
NDAT	DS	4C	New Account
NCUS	DS	25C	Transaction
NADR	DS	30C	
NTYP	DS	2C	
NAMT	DS	7C	

Example (ORG and DS Codes) Cont'd

NAME	OPERATION	OPERAND	
	ORG	INAR	
PACN	DS	10C	
PCOD	DS	2C	
PDAT	DS	4C	Payment
PAMT	DS	7C	Transaction
PTYP	DS	2C	
RACN	ORG	INAR	
RCOD	DS	10C	
RDAT	DS	2C	Receipt
RAMT	DS	4C	Transaction
	DS	7C	
	ORG	INAR+80	Reset Location Counter to value after INAR above

Areas allocated by the DS code are not cleared. The NAME field may be left blank for areas that must be allocated but are not referenced directly.

CONSTANTS

The DC (Define Constant) code both allocates memory for, and stores the value of a constant. The value is written in the OPERAND field, and is expressed in one of three forms, as a character, hexadecimal or address constant. The length of each constant is implied by the value in the operand field.

CHARACTER CONSTANTS

A constant not exceeding 16 characters may be written on one statement line. Each character is converted to a byte. The value, enclosed in single quote marks, is preceded by a C.

NAME	OPERATION	OPERAND
EOF	DC	C'END OF RUN'
CODS	DC	C'012AB'

HEXADECIMAL CONSTANTS

A hexadecimal constant must be used in lieu of a character constant when one or more of the bytes cannot be expressed by a character value. The value is written in an even number of hexadecimal digits not exceeding thirty-two. Each pair of hexadecimal digits (starting from the left end of the expressed value) is used to generate a byte.

OPERATION	OPERAND
DC	X'4020206B20204B202060'

An explanation of the above example is given in DATA EDITING (page 33).

ADDRESS (EXPRESSION) CONSTANT

An address may be stored as a two-byte constant. There must be a separate statement line for each constant of this type. The constant is enclosed in parenthesis and preceded by an A as in the following examples.

	OPERATION	OPERAND
1.	DC	A(*-6)
2.	DC	A(STRT)
3.	DC	A(256)

Explanation

1. Stores the current value of the Location Counter - 6 as a constant (RELOCATABLE).

2. Stores the value of STRT as a constant (RELOCATABLE).

3. Stores the binary equivalent of 256 as a constant (NOT RELOCATABLE).

DEFINING REPETITIVE CONSTANTS

Constants in character and hexadecimal form may be defined in repetitive fashion.

gC'cv'
gX'cv'

In the format above, g represents the actual number of constants to be generated, and 'cv' represents the constant value.

The statements,

NAME	OPERATION	OPERAND
CON1	DC	3C'ABC'
CON2	DC	2X'3F4'

generate ABCABCABC and 03F403F4.

The g factor may not be properly incorporated in an address constant statement.

The implied length of CON1 and CON2, in the above example, is three and two.

DEFINING EXPLICIT CONSTANT LENGTHS

The define constant format may be expanded to include specific references to length.

gCLn'cv'
gXLn'cv'
ALn'cv'

L may not exceed 256_{10}
n = number of bytes
n may not exceed $(16)_{10}$

The explicit length takes precedence if it does not correspond to the physical length of the 'cv' value.

NAME	OPERATION	OPERAND	GENERATED
	DC	2CL3'ABC'	ABCABC
	DC	2XL5'3F6'	00000003F600000003F6
	DC	2CL2'1AB'	1A1A

In the third example above, the B, rightmost character of the constant 1AB, was truncated because a length of two was specified. In like manner, the constant defined on the second line was padded-out to a length of five.

If an explicit length is not included when defining an Expression Constant, a length of 4 bytes is assumed, and the generated constant is aligned on an even word boundary.

When a length other than four (4) is specified, the constant is not aligned and is not relocatable. If the value of the expression exceeds the assigned number of bytes, the high-order bits of the value are truncated.

Absolute expressions may have negative values which are generated in 2's complement form.

Example:

NAME	OPERATION	OPERAND	Generated Binary Constant
CON1	DC	AL1(-2)	11111110
CON2	DC	AL1(-5)	11111011

PROGRAM LINKING CODES

The two codes, ENTRY and EXTRN, provide communication between two programs that have been assembled independently. The ENTRY code specifies the location(s) addressed by another program. The EXTRN code defines a symbol in another program.

ENTRY CODE

A separate ENTRY verb must appear for each entry point in the program (see START code for exception). ENTRY appears in the OPERATION field and a symbol must be used in the OPERAND field. The NAME field is not used on this line.

EXTRN CODE

Reference to a symbol in another program is defined by the EXTRN Code. A separate statement must appear for every symbol appearing in another program. EXTRN appears in the OPERATION field and a symbol must be used in the OPERAND field. The NAME field is not used on this line.

EXAMPLE OF ENTRY AND EXTRN

One use of the ENTRY and EXTRN codes is to link a program to a subroutine. Assume that a SINE-COSINE subroutine has two ENTRY points; SINE and COS, and one EXTRN point; RTN. Programs using the SINE-COSINE routine can BRANCH to either SINE or COS depending on which function is to be computed. The SINE-COSINE routine BRANCHES to RTN after computing the function.

SINE-COSINE ROUTINE

NAME	OPERATION	OPERAND
SINE	START	
	ENTRY	COS
	EXTRN	RTN
	- -	
	- -	
	- -	
	B	RTN
COS	- -	
	- -	
	- -	
	B	RTN
	END	

MAIN PROGRAM

NAME	OPERATION	OPERAND	
BGN	START		
	ENTRY	RTN	
	EXTRN	COS	
	- -		
	- -		
	- -		
	B	COS	COS is an ENTRY point in the SINE-COSINE Routine
RTN	- -		
	- -		
	- -		
	END		

The main program defines RTN as an ENTRY point that allows the SINE-COSINE routine to BRANCH to RTN.

RUN AND SEGMENT CONTROLLING CODES

The first and last statements of the source program must be a START and END statement, respectively. If a program contains sections which are to be loaded individually, the second and succeeding sections must begin with a CSECT control code.

START CODE

In addition to flagging the start of the source program, a START code can set the location counter to an initial value and identify an entry into the program.

START must appear in the OPERATION field. A self-defining value, written in the OPERAND field, sets the location counter. A symbol appearing in the NAME field is considered an ENTRY point into the program (see ENTRY code).

NAME	OPERATION	OPERAND	
BEGIN	START	S'064'	Set Location counter to $100_{(10)}$. Establish "BEGIN" as an entry point.

END CODE

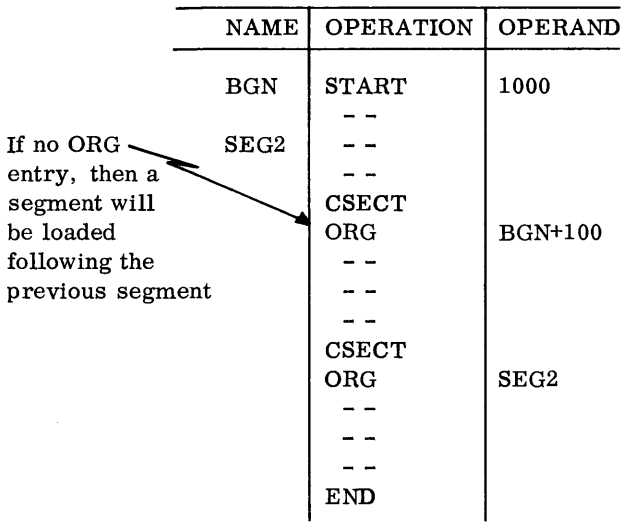
The END code informs the Assembler that all source input statements have been processed. The OPERAND field specifies the starting address of the object program. A symbol, self-defining value, or asterisked address may appear in the OPERAND field.

NAME	OPERATION	OPERAND	
	END	STARTP	STARTP is the name of the program starting address

CSECT CODE

The CSECT code identifies both the beginning of a new section (segment) and the termination of the previous section. The START code identifies the first section, therefore CSECT should be used for the second and succeeding sections only. The NAME field may contain a symbol or be left blank. A symbol used in one segment can be referenced by any other segment in the program.

Example of CSECT



NAME	OPERATION	OPERAND
BGN	START	1000
	- -	
SEG2	- -	
	- -	
	CSECT	
	ORG	BGN+100
	- -	
	- -	
	- -	
	CSECT	
	ORG	SEG2
	- -	
	- -	
	- -	
	END	

The above example illustrates a program consisting of three segments. The second segment is loaded at HSM location 1100_{10}. The third segment will overlay the first two segments at the location named SEG2.

EQUATE CODE

The Equate (EQU) command assigns symbolic names to values, or assigns the same address to two symbolics. The statement,

NAME	OPERATION	OPERAND
XYZ	EQU	6

enters the name XYZ in the symbol table, and to it, assigns the value 6. One name may be equated to another in similar fashion.

NAME	OPERATION	OPERAND
BTAG	EQU	ATAG

The above statement causes the Assembly to enter the name BTAG in the table, and assign to it the address previously assigned to ATAG. The symbol appearing in the Operand field must have been previously defined. It must have appeared in the name field of a prior statement.

BASE REGISTER CONTROLLING CODES

The USING Code indicates to the Assembler what base address values will be in specific general registers at object time. The Assembler creates a table of these values, and then uses them to assign registers and to compute displacements when base references are not included in an expression. (See page 3).

The first operand denotes the value assumed to be in a register. The second operand specifies the register.

NAME	OPERATION	OPERAND
	USING	STARTP+6, 5
	USING	*-10, 6

The first operand may be a relocatable expression while the second is simple and absolute.

The USING code does not load a general register. A Load Multiple, Move Character or Branch and Link instruction places the actual base address in a general register.

The assumed value of a register may be altered by incorporating another USING statement at any place in the source program.

NAME	OPERATION	OPERAND
	USING	*+500, 5

The only function of the DROP code is to delete a base value from the Assembly table.

NAME	OPERATION	OPERAND
	DROP	6

It is not necessary to drop a base address from the table prior to changing it with a second USING statement.

EXTENDED MNEMONIC INSTRUCTIONS

The assembler provides a simplified method of defining the various options available via the Branch on Condition instruction. The pseudo operation codes listed below replace the BC and the associated mask.

PSEUDO OP	FUNCTION	BC EQUIVALENT
B	Branch Unconditional	BC X'F', S_1
NOP	No Operation	BC X'0', S_1
BH	Branch on High	BC X'2', S_1
BL	Branch on Low	BC X'4', S_1
BE	Branch on Equal	BC X'8', S_1
BO	Branch on Overflow (Arithmetic) or Branch if Ones (after TM)	BC X'1', S_1
BP	Branch on Plus	BC X'2', S_1
BM	Branch on Minus (Arithmetic) or Branch if Mixed (after TM)	BC X'4', S_1
BZ	Branch on Zero (Arithmetic) or Branch if Zero (After TM)	BC X'8', S_1

Examples:

OPERATION	OPERAND
B	BEGIN
BH	ERR
BZ	ZERO

Exercise:

1. What does the symbol * (asterisk), in column 1 of the Assembler Coding form, tell the Assembler program?

2. What is a redundant name?

3. If a statement has a symbol in the name field and an * (asterisk) as its first operand, which of the following is true?

   a. Name address and asterisk address are the same.
   b. Asterisk address is name address plus one.
   c. Asterisk address is name address plus two.

4. Indicate the type of the following self-defining value.

Self-Defining Value	Type
C'9'	
X'4645'	
C'A'	
246	
C'A9'	

5. Indicate whether the following statements are true or false.

   T F a. A DS code reserves a portion of memory and clears the reserved area to blanks.

   T F b. Three types of constants may be defined with the DC instruction.

   T F c. The length of a constant must be explicit.

   T F d. If the length of a constant defined by a single DC statement is 17 bytes the rightmost byte will be truncated.

   T F e. A hexadecimal constant consisting of an odd number of hexadecimal digits will have a zero digit padded at the left end and the statement will be flagged in the listing.

6. Describe the use of the EXTRN and ENTRY codes.

7. How many external symbols may be identified by an EXTRN statement?

8. Explain the restrictions imposed by the Assembler on the placing of ENTRY statements in the program.

9. Using the Assembler coding form, write the Assembler and machine instructions necessary to:

   a. Set the Location Counter to an initial setting of 1000.

   b. Provide linkage to a subroutine called ABLE which is not a part of this program.

   c. Set aside a working storage area of 75 bytes.

   d. Provide an address constant of 3000.

10. Define the following abbreviations as used by the machine instructions.

    a. L

    b. L_1

    c. L_2

    d. S

    e. S_1

10. (Cont'd)

    f. S_2

    g. M

    h. T

    i. D

11. Briefly define the following terms:

    a. compound expression

    b. implied length

    c. implied base address

    d. relocatable expression

12. Write a statement generating six repetitions of the hexadecimal constant 'XYZ'. Give the constant an explicit length of 6.

    What would the statement generate?

13. Write a statement allocating a work area of 100 bytes. Align the left-hand end of the area on an even word boundary.

14. How are 32 bit relocatable expression constants defined?

15. Using the EQU command, write statements equating

    a) the address of the name BEGIN to the name ENDP.

    b) the name NINE to the value 9.

16. Describe the purpose of the USING and DROP statements. How are general registers loaded with base addresses?

INSTRUCTION COMPLEMENT

The RCA 70/25 Order Code consists of thirty-one instructions that can be divided into four classes.

1. DATA HANDLING

The data handling instructions allow for the movement of data fields within HSM. Date may be moved without changing format or it can be packed, unpacked, or edited for printing during the movement. A translate instruction facilitates code conversion and data validation.

2. ARITHMETIC INSTRUCTIONS

This set includes Decimal Add, Subtract, Multiply, and Divide instructions, as well as Binary Add and Subtract operations. It also incorporates the ability to perform Boolean Operations on bit structures.

3. DECISION AND CONTROL

The decision and control instructions allow for the comparing of both Decimal and Binary fields, and the branching to a location in HSM according to a Condition Code Indicator. Also included are Branch and Link and Branch on Count instructions which simplify subroutine linkage, and control of iterative coding.

4. INPUT/OUTPUT

This set reads and writes data between the processor and all peripheral equipment on-line to the 70/25. It includes the necessary instructions to control the devices, and to recognize and recover from error conditions.

DATA MOVEMENT INSTRUCTIONS

Data may be moved from one point in memory to another with or without change. The changes that can occur during a moving operation are to pack, unpack, or to unpack and edit.

MOVE CHARACTER INSTRUCTION

The Move Character instruction transfers one byte or one word at a time from a sending to a receiving field. The number of bytes transferred is controlled by the L Register. When possible the hardware accesses a word at a time. The L character (sent to the L Register in the staticizing process) is one less than the number of characters to be transferred (in machine format) because (1) the first character is transferred before the L Register is decremented and (2) the L Register is compared after decrementing to FF_{16} (1 less than 00_{16}) to terminate the execution of the instruction.

Example:

Construct an output record by transferring selected fields from the input area.

Assume that an input record area (INP) is located in memory at 2000-2099 and an output record area (OUP) is in 2200-2299. An account number, 8 characters, is the first field in each record area.

The following instruction would move the account number to output record area.

OPERATION	OPERAND
MVC	OUP(8),INP

or in machine format as:

OP	L	B_1	D_1	B_2	D_2
$D2_{16}$	7	1_{10}	2200_{10}	1_{10}	2000_{10}

General Register one contains 0000

It should be noted that in assembly format as in machine format, the second field is the address of the left end of the sending area. The first address is the left end of the receiving area.

Based on the example above, assume the field INP contains the value as shown below.

INP

	00	01	02	03	04	05	06	07
20	3	7	0	1	4	9	6	5

The field OUP would be filled with characters from the INP area as shown below.

OUP

	00	01	02	03	04	05	06	07
22	3	7	0	1	4	9	6	5

A data field may be filled with a given character or cleared by the Move Character instruction by overlapping the receiving field so that it begins one position to the right of the sending field. Thus the first

character transferred is generated in each position of the receiving area.

As an example, assume 120-character area is to be filled with blanks. It is known that a blank $(40)_{16}$ appears in the first position.

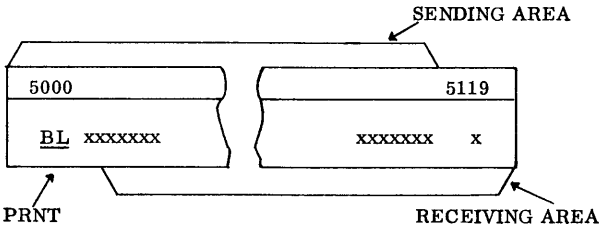
The area is allocated as follows:

NAME	OPERATION	OPERAND
PRNT	ORG DC DS	5000 X'EO' 119C

The name PRNT is assigned the address 5000. The area is cleared by the following instruction:

NAME	OPERATION	OPERAND
HSKP	MVC	PRNT+1(119),PRNT

The following diagram illustrates the overlapping of the sending area by the receiving area.



Upon completion of the execution of the instruction the area 5000 to 5119 will be filled with the Blank $(EO)_{16}$ character.

Exercise:

For the purpose of this exercise, assume memory to be allocated as follows:

NAME	OPERATION	OPERAND
	ORG	2000
WORK	DS	5C
NAME	DS	10C
BAL	DS	6C
DATA	DS	5C
	ORG	2100
WA1	DS	26C

PART I

Write the instructions to perform the following operations. Place your answers in the space provided.

1. Move 'Work' to 'Data'

2. Zero fill the 'Bal' field. (Assume the first byte of 'Bal' is a zero.)

3. Clear the 'Work' and 'Name' fields to blanks. (Assume the first byte of 'Work' is a blank.)

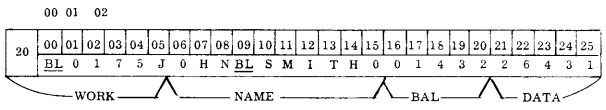
ANSWERS

	NAME	OPERATION	OPERAND
1.			
2.			
3.			

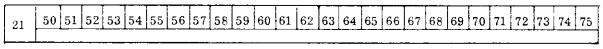
PART II

Assume areas allocated as shown on Line 1. How would the area called WA1 appear after execution of the four instructions below?

Line 1



Line 2



	NAME	OPERATION	OPERAND
4.		MVC	WA1(4),WORK+2
5.		MVC	WA1+20(6),NAME+5
6.		MVC	WA1+4(1),WORK+1
7.		MVC	WA1+S(15),WA1+4

PART III

Each record on an input tape contains the following fields of data.

ITEM	NO OF CHARS
Account No.	8
Name	25
Type Account	3
Street Address	20
City State Code	2
Credit Code	5
Balance	8
Total Purchases	8
Total Returns	8

Write a routine that will construct one output record in the following format.

Account No.
Credit Code
Balance
Total Returns
Total Purchases
Name
Street Address
City State Code

Allocate memory for the input record at location 2000 and for the output record at location 2100. Coding is to begin at 2300. Assume the record is present in memory.

PACKING AND UNPACKING DATA

The previous section discussed the movement of a data field from one to another area of memory using the Move Character instruction. This instruction moved byte(s) without changing their structure.

As outlined in the section on Data Format, data must be in packed format before decimal arithmetic operations may be performed. Data must be in unpacked format before any type of display output (such as printing) may be performed. The Pack and Unpack instructions enable the user to perform these operations as the data is moved.

PACK INSTRUCTION (PACK)

To illustrate use of the Pack instruction assume that an area must be allocated for input transactions (Unpacked) which are in the format as follows:

STOCK NO.	8
CODE	2
AMOUNT	8

One area is allocated for reading in the transaction, and another (a work area) for packing the Amount field prior to updating the Master Record balance field.

The input area is allocated as follows:

NAME	OPERATION	OPERAND
	ORG	3000
STNO	DS	8C
CODE	DS	2C
AMT	DS	8C

The work area for packing the Amount (AMT) field is as follows:

NAME	OPERATION	OPERAND
WAMT	DS	5C

It should be noted that the unpacked field (AMT) can be packed into a much smaller field (WAMT).

The least significant byte of the unpacked field is the only byte that fully occupies a byte position in the packed field. All other bytes are stripped of the zone portion before transfer to the packed field. Therefore, a quick way of determining the number of bytes necessary in the packed field is to divide by two the size (in bytes) of the unpacked field and add 1. Thus, (Unpacked field) $\frac{8}{2} + 1 = 5$ (number of bytes for packed result).

Assuming the amount (AMT) field contained the value as indicated below, the instruction to pack the field in WAMT and the resulting packed field are shown:

AMT

UNPACKED (SENDING FIELD)

	10	11	12	13	14	15	16	17
30	Z0	Z0	Z0	Z2	Z1	Z4	Z9	S7

ASSEMBLY INSTRUCTION

OPERATION	OPERAND
PACK	WAMT(5), AMT(8)

GENERATED INSTRUCTION

OP	L_1	L_2	B_1	D_1	B_2	D_2
$F2_{16}$	4	7	1_{10}	3050_{10}	1_{10}	3010_{10}

General Register one contains 0000

WAMT

PACKED (RECEIVING) FIELD

	50	51	52	53	54
30	00	00	21	49	7S

The receiving field is considered the controlling field for terminating the execution of the instruction.

If the receiving field is not large enough to contain all the digits in the unpacked (sending) field, then truncation of the high-order digits takes place.

If the receiving field is larger than necessary to contain all digits in the sending field, the high-order half-bytes of the packed field are filled with zero digits.

The programmer must be sure that he is dealing with valid fields for both the packing and unpacking operations. There is no hardware check, for example,

that valid numeric characters (or a sign) exist. The first byte position processed in the sending field merely has its half bytes transposed and sent to the receiving (packed) field. Each successive byte in the sending field has its zone portion stripped and the numeric portion forms successive half-bytes in the packed field.

UNPACK INSTRUCTION (UNPK)

The unpacking operation is the reverse of the packing operation.

The first byte in the sending (packed) field is processed by having its zone (sign) and numeric portions reversed and sent to the receiving (unpacked) field.

Each successive half-byte in the packed field forms a byte in the unpacked field with a zone portion of $F16$ $(1111)_2$ being generated by hardware during execution of the instruction.

As an example of the Unpack instruction assume that it is desired to print a balance field as a part of an output record.

The balance field is a packed field that has accumulated the transaction amounts. Assume it is in packed format and contains the following value:

PACKED (SENDING) FIELD

	BAL			
40	50	51	52	53
	00	17	24	3S

An area for printing the balance field has been allocated as follows:

NAME	OPERATION	OPERAND	ASSUMED HSM ALLOCATION
PBAL	DS	7C	5037-5043

The instruction as shown below would unpack BAL with the result in the field PBAL.

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	UNPK	PBAL(7), BAL(4)

GENERATED INSTRUCTION

OP	L_1	L_2	B_1	D_1	B_2	D_2
$F3_{16}$	6	3	2_{10}	0941_{10}	1_{10}	4050_{10}

General Register one contains 0000
General Register two contains 4096_{10}

UNPACKED (RECEIVING) FIELD

	PBAL							
50	37	38	39	40	41	42	43	*
	F0	F0	F1	F7	F2	F4	S3	

* $F = F_{16} = 1111_2$
S = Sign

The receiving (unpacked) field can be considered the controlling field. If it is larger than necessary, high-order bytes are filled with the numeric character zero FO_{16}.

If it is not large enough to receive all digits in the sending (packed) field, the high-order digit(s) of the sending field are truncated.

To determine the size of the unpacked field necessary to receive the digits in a packed field, the size (in bytes) of the packed field should be doubled and one should be subtracted for determining the number of bytes necessary.

Exercise:

Assume that the following allocations have been made:

NAME	OPERATION	OPERAND
	ORG	2000
BAL	DS	5C
	DS	3C
WDAT	DS	5C
	ORG	2150
DATA	DS	3C
	DS	6C
WBAL	DS	4C

and that the allocated areas contain the hexadecimal values as shown below:

BAL WDAT

	00	01	02	03	04	05	06	07	08	09	10	11	12
20	F0	F1	F2	F5	C7	F0	C3						

DATA WBAL

	50	51	52	53	54	55	56	57	58	59	60	61	62
21	00	24	5C										

Answer each of the following questions by writing the assembly instruction in the space provided and showing the result of the instruction in the blank locations above.

1. Pack 'BAL' in 'WBAL'

2. Unpack 'DATA' in 'WDATA'

ANSWERS (Assume that General Register one contains 0000.)

	NAME	OPERATION	OPERAND
1. →			
2. →			

Show the generated instruction for each of the assembly instructions that follow and the results of each instruction in the blank locations above.

	OPERATION	OPERAND
3.	PACK	WDAT-1(1), BAL+6(1)

OP	L_1	L_2	B_1	D_1	B_2	D_2

	OPERATION	OPERAND
4.	UNPK	DATA+4(5), DATA(3)

OP	L_1	L_2	B_1	D_1	B_2	D_2

	OPERATION	OPERAND
5.	PACK	DATA+3(1), BAL(5)

OP	L_1	L_2	B_1	D_1	B_2	D_2

A Master Inventory File is in the following format with all fields in unpacked format.

ITEM NO.		NO. OF CHARS.
1	STOCK NO.	9
2	AGENCY CODE	3
3	ACTIVITY CODE	1
* 4	FORECAST REQUIREMENT	8
5	MANUFACTURER	15
6	MFGR'S. ADDRESS	20
7	MFGR'S. CITY, STATE	15
* 8	STOCK ON HAND	7
* 9	RESERVE REQUIREMENT	6
*10	DUE IN	6
11	REVIEW DATE	6

Items preceded by an asterisk (*) are signed numeric fields (Unpacked).

Requirement No. 1

Allocate memory for the Input Record beginning at 3000 and for the output record beginning at 3100. The output record area is allocated in the same format sequence as the Input record area, however, asterisked items are in packed format and of a minimum size to contain all digits in the input items.

Requirement No. 2

Write a routine with coding to begin at 3200 that will construct an output record. Assume for the purposes of writing your routine that the input record is present in memory.

DECIMAL ARITHMETIC INSTRUCTIONS

Decimal Add and Subtract

The RCA 70/25 has four decimal arithmetic instructions, Add Decimal (AP) Subtract Decimal (SP), Multiply Decimal (MP), and Divide Decimal (DP).

All require operands to be in packed format. The rightmost byte in each field is assumed to contain the sign in the low-order four bits.

A sign is generated in the least significant byte of the result field.

The sign (rightmost four bits of the result operand) is a C_{16} $(1100)_2$ for a positive field or a D_{16} (1101_2) if the result field is negative.

The Condition Code Indicator is set following execution of the instruction based on whether the result field is zero, positive (greater than zero), negative (less than zero), or if overflow has occurred. Overflow interrupt can occur after add and subtract operation, but not after multiply or divide. Overflow, if present, overrides the setting for a positive or negative result.

As an example, assume the following fields are in memory:

BAL

50	06	07	08	09
	23	87	23	1+

AMT

51	50	51	52
	23	47	5+

and the following instruction is issued:

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	AP	BAL(3), AMT(3)

GENERATED INSTRUCTION	OP	L_1	L_2	B_1	D_1	B_2	D_2
	FA_{16}	2	2	2_{10}	0911_{10}	2_{10}	1054_{10}

General Register two contains 4096_{10}

the result field will appear and the Condition Code Indicator will be set as follows:

BAL

50	06	07	08	09
	23	10	70	6+

CONDITION CODE = 3 (overflow)

When overflow occurs, the position to the left of the result field (HSM 5006 above) is not affected by the 1 carry out of the MSD of the result. The overflow setting (Condition Code 3) overrides the positive result setting which would otherwise be set (Condition Code 2).

The operands being added (or subtracted) do not have to be of equal length. The first (and result) operand, however, should be the longer operand if they are unequal. The first operand can be considered the controlling operand.

If the second operand is shorter in length, high-order zeros are generated by hardware until the leftmost digit of the first operand has been reached.

If the second operand is longer, its high-order excess bytes do not affect the result.

It should be noted that this condition does not necessarily set the overflow condition. Overflow is set only by a 1 carry from the most significant digit of the result.

For example, assume HSM contains a field with the following value:

BAL

30	00	01	02
	75	23	4+

and an Amount field contained the following values:

EXAMPLE 1

AMT

30	20	21	22	23	24
	00	05	21	67	5+

EXAMPLE 2

AMT

30	20	21	22	23	24
	00	03	31	84	2+

If an attempt were made to add the amount to the balance with the following instruction:

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	AP	BAL (3), AMT (5)

	OP	L_1	L_2	B_1	D_1	B_2	D_2
GENERATED INSTRUCTION	FA_{16}	2	4	1_{10}	3000_{10}	1_{10}	3020_{10}

General Register one contains 0000

the result in the balance BAL would be as follows and the Condition Code would be set as indicated.

EXAMPLE 1 RESULT

	BAL		
30	00	01	02
	96	90	9+

CC = 2 (POSITIVE RESULT)

EXAMPLE 2 RESULT

	BAL		
30	00	01	02
	07	07	6+

CC = 3 (OVERFLOW)

Note that in Example 2, the Condition Code of 3 (overflow) is not an indication that truncation has occurred as truncation also has occurred in Example 1. The overflow setting is based on the 1 carry from the MSD of the Result field.

There is no hardware check or error indication for invalid or incorrectly addressed fields. It is the responsibility of the programmer to be sure that he has addressed valid packed fields.

A field may be added to (or subtracted from) itself if desired.

As an example, assume that a field has been used for accumulation and it is desired to zero fill it with a valid sign in the rightmost byte position. The field WBAL is as follows:

WBAL

61	43	44	45	46
	02	15	24	3+

The following instruction would zero fill and preserve the sign position:

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	SP	WBAL (4), WBAL (4)

	OP	L_1	L_1	B_1	D_1	B_2	D_2
GENERATED INSTRUCTION	FB_{16}	3	3	2_{10}	2047_{10}	2_{10}	2047_{10}

General Register two contains 4096_{10}

WBAL

RESULT FIELD

61	43	44	45	46
	00	00	00	0+

MULTIPLY DECIMAL

Multiplication may be performed on two packed operands. The result (product) replaces the first operand (multiplicand) following execution of the instruction.

The number of leading zeros in the multiplicand should equal the number of significant digits in the multiplier. If the multiplicand field has an insufficient number of leading zeros, the high-order product digits are truncated.

The programmer must ensure that fields are valid numerics as the hardware performs no check of this type. The Condition Code is not affected by the execution of this instruction.

Example #1

In this example assume the two fields are present in memory and have been assigned the following values:

PRCE

21	07	08	09
	21	57	2+

NUNT

22	30	31
	02	7+

The item PRCE, to be multiplied by NUNT, is not of sufficient length to receive the product. It could be moved to a zero-filled work area and multiplication performed as follows:

HSM BEFORE EXECUTION

	WPRC				
22	00	01	02	03	04
	00	00	21	57	2+

HSM BEFORE AND AFTER EXECUTION

22	30	31
	02	7+

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	MP	WPRC(5) ,NUNT (2)

GENERATED INSTRUCTION	FC_{16}	4_{10}	1_{10}	2_{10}	200_{10}	2_{10}	230_{10}

GENERAL REGISTER 1 CONTAINS 2000_{10}

HSM AFTER EXECUTION

22	00	01	02	03	04	
	00	05	82	44	4+	

It should be noted, based on the above example, that with only two significant digits in the multiplier the leftmost zero-filled byte in the product was not needed. However, if the field NUNT contained three significant digits the added zero byte would be required as in the following example:

Example #2

WPRC

		00	01	02	03	04
HSM BEFORE EXECUTION	22	00	00	21	57	2+

		30	31
HSM BEFORE AND AFTER EXECUTION	22	87	5+

ASSEMBLY INSTRUCTION

GENERATED INSTRUCTION

See Example 1 above

WPRC

		00	01	02	03	04
HSM AFTER EXECUTION	22	01	88	75	50	0+

DIVIDE DECIMAL

Division is performed on two packed operands with the quotient and remainder replacing the first operand (dividend) following execution of the instruction.

Two conditions cause a divide-exception interrupt (assuming permit mask):

1. Dividend does not contain at least one significant (leading) zero.

Example:

5 BYTE DIVIDEND FIELD

VALID	02	17	54	31	8+
INVALID	12	17	54	31	8+

2. The divisor digits, when aligned with the digits one position to the right of the leading zero of the dividend, have a lesser or equal value.

Example:

VALID	01	75	42	1+	DIVIDEND
	41	7+			DIVISOR
INVALID	01	75	42	1+	DIVIDEND
	04	1+			DIVISOR

A divide-exception interrupt is avoided by positioning the dividend with sufficient leading zero digits. For example, a two-byte divisor containing from one to three significant digits would not cause a divide exception if two or more of the most significant bytes of the dividend were zero.

Example:

DIVS

		00	01
HSM BEFORE AND AFTER EXECUTION	22	00	7+

DIVD

		30	31	32	33
HSM BEFORE EXECUTION	22	00	00	49	0+

ASSEMBLY INSTRUCTION	NAME	OPERATION	OPERAND
		DP	DIVD(4), DIVS(2)

GENERATED INSTRUCTION							
	FD	3_{10}	1_{10}	2	0030_{10}	2	0000_{10}

General Register 2 = 2200_{10}

DIVD

		30	31	32	33
HSM AFTER EXECUTION	22	07	0+	00	0+

Following the execution of a Divide instruction, the remainder appears right justified in the dividend field and is equal in length to the divisor. The length of the quotient, therefore, is $L_1 - L_2$.

Exercise:

1. Assume HSM has been allocated as indicated on Line 1 and that each location contains the values as shown.

		AMT 1 ↓		AMT 5 ↓				AMT 3 ↓			AMT 2 ↓		AMT 4 ↓						
Line 1	21	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
		01	25	6+	00	12	47	5+	04	21	2+	00	24	3-	00	12	47	83	21	47	9+	00

Line 2	21	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20

For each of the assembly instructions listed below, show the result of the instruction on Line 2 above and the Condition Code as set following execution of the instruction. (Consider each question independently based on the contents of Line 1.)

	OPERATION	OPERAND	CONDITION CODE 0	1	2	3
1.	AP	AMT 1 (3), AMT 2 (2)				
2.	AP	AMT 3 (2), AMT 1 (3)				
3.	SP	AMT 2 (2), AMT 3 (2)				
4.	SP	AMT 4 (6), AMT 5 (3)				
5.	AP	AMT 5 + 2 (1), AMT 4 + 5 (1)				

2. Write an assembly statement multiplying the contents of area TOTAL by the contents of area PRICE.

	1000	1001	1002	1003	1004	1005
TOTAL	8 3	6 0	4 7	8 4	4 3	0 -

	1200	1201	1202
PRICE	4 1	0 4	6 4

NAME	OPERATION	OPERAND

Show the contents of TOTAL area after execution.

	0997	0998	0999	1000	1001	1002	1003	1004	1005
TOTAL									

3. Assume six values stored in HSM. Write the assembly statements necessary to compute the average value. Place the most significant digit of the result in location 15000.

Assumed values:

10000	10001	10002
0 8	9 3	7 +

10006	10007	10008
9 3	7 8	9 +

10003	10004	10005
0 0	0 0	3 +

10009	10010	10011
1 3	6 7	6 +

10012	10013	10014
4 3	0 0	1 +

10015	10016	10017
0 0	6 7	9 +

NAME	OPERATION	OPERAND

Result:

15000	15001	15002	15003	15004	15005	15006	15007	15008	15009	15010

DATA EDITING

In previous sections we have seen that data may be moved from one area to another either unchanged in byte structure or with packing or unpacking being performed; or, data may be edited as it is moved.

Editing is very much like unpacking data except that two additional functions are performed as the data is unpacked. The editing instruction can (1) suppress leading zeros to a predetermined location in the edited field and (2) insert editing characters as the data is moved to the edited field.

A data field to be edited is assumed to be in valid packed format, i.e., each half-byte is a valid numeric (0-9) except the rightmost half-byte which is a sign.

Data is moved from this packed field to a receiving field that controls the insertion of the numeric digits (half-bytes). The numeric digits are unpacked as they are transferred to the edited field.

The receiving field (edit mask) consists of characters to be inserted as editing symbols such as the comma, decimal point, and asterisk, for example. In addition, the following characters are control characters in the edit mask: (Hexadecimal format of byte shown.)

X'20' - DIGIT SELECT

This character is placed in the edit mask where it is desired to insert a digit from the packed field. The digit is inserted unless it is a leading insignificant zero and a Significance Start character has not been encountered previously.

X'21' - SIGNIFICANCE START

This character serves the same function as the Digit Select character with one added function; it specifies that all of the following digits are to be inserted from the packed field even if one or more leading zeros are still present.

X'22' - FIELD SEPARATOR

This character is used for editing multiple fields; it specifies the end of one and the start of another field and resets the edit operation for the beginning of another field.

To illustrate the editing functions, assume that a packed field has the following format and value:

AMT

	00	01	02	03
20	00	02	37	8+

and that the field is to be edited so that leading zeros will be suppressed.

To do this, allocate an edit mask as follows:

NAME	OPERATION	OPERAND
MASK	DC	X'E02020202020202060'

Hexadecimal characters are used because some of the bytes cannot be represented by a character constant.

The first character of the mask is a fill character; it replaces digit select (X'20') and editing symbols in the mask until one of the following conditions takes place:

1. The first non-zero numeric digit is encountered in the packed (sending) field.

2. A Significance Start character has been encountered in the edit mask (receiving) field.

The fill character also replaces all remaining positions in the edit mask when a plus sign is encountered in the packed (sending) field unless processing multiple packed fields.

To illustrate the above example, assume the edit mask above has been assigned the following memory allocation:

	00	01	02	03	04	05	06	07	08
29	_	d	d	d	d	d	d	d	⊖

where: _ = BLANK
d = DIGIT SELECT
⊖ = MINUS SIGN

HSM BEFORE AND AFTER EXECUTION

AMT

	00	01	02	03
20	00	02	37	8+

MASK

HSM BEFORE EXECUTION

29	00	01	02	03	04	05	06	07	08
	_	d	d	d	d	d	d	d	⊖

ASSEMBLY INSTRUCTION

OPERATION	OPERAND
ED	MASK (9), AMT

GENERATED INSTRUCTION

OP	L	B_1	D_1	B_2	D_2
DE_{16}	8	1_{10}	2900_{10}	1_{10}	2000_{10}

General Register one contains 0000

HSM AFTER EXECUTION

29	00	01	02	03	04	05	06	07	08
	-	-	-	-	2	3	7	8	+

SIGNIFICANCE TRIGGER SETTING* → 0 ——→ 1 ——→ 0

*To determine when to insert the fill character in the Edit Mask, the hardware employs a Significance Trigger. This trigger is set to zero initially. The zero setting specifies the fill character in the edit mask positions. The trigger retains a zero setting until either:

1. A Digit Select character in the mask references the first non-zero numeric digit in the packed (sending) field,

OR

2. A Significance Start character has been encountered in the Edit Mask field.

The trigger is set to 1 after either of these conditions. The 1 setting specifies insertion of the digit (regardless of value) from the packed field in the Edit Mask where a Digit Select character is present. The 1 setting also specifies insertion of editing symbols present in the Edit Mask.

The setting of 1 is retained until either a plus sign is encountered in the packed field or a field separator character is encountered in the Edit Mask. Either of these conditions resets the trigger to zero.

The Edit instruction sets the Condition Code to zero if the packed field has a zero value, to one if the value is negative, and to two if the value is positive.

Example #1

The mask that would edit the previous field (AMT) with a decimal point and a comma (if the value were 1,000.00 or higher) would be as follows:

NAME	OPERATION	OPERAND
EDMK	DC	X'E020206B2020204B202060'

AMT

HSM BEFORE AND AFTER EXECUTION

20	00	01	02	03
	00	02	37	8+

EDMK

HSM BEFORE EXECUTION

29	00	01	02	03	04	05	06	07	08	09	10
	-	d	d	,	d	d	d	.	d	d	⊖

ASSEMBLY INSTRUCTION

OPERATION	OPERAND
ED	EDMK (11), AMT

GENERATED INSTRUCTION

OP	L	B_1	D_1	B_2	D_2
DE_{16}	10	1_{10}	2900_{10}	1_{10}	2000_{10}

General Register one contains 0000

EDMK

HSM AFTER EXECUTION

29	00	01	02	03	04	05	06	07	08	09	10
	-	-	-	-	-	2	3	.	7	8	-

SIGNIFICANCE TRIGGER SETTING 0 ——→ 1 ——→ 0

CONDITION CODE = 2

Example #2

Editing with Decimal Point and at least two zeros present.

AMT

HSM BEFORE AND AFTER EXECUTION		00	01	02	03
	20	00	00	00	0+

MASK

HSM BEFORE EXECUTION		00	01	02	03	04	05	06	07	08	09	10
	29	-	d	d	,	d	d	S	.	d	d	-

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	ED	MASK (11), AMT

MASK

HSM AFTER EXECUTION		00	01	02	03	04	05	06	07	08	09	10
	29	-	-	-	-	-	-	-	-	.	0	0	-

0 ——→ 1 ——→ 0

CONDITION CODE = 0

Examples 3 and 4

Same Mask - Result after positive and negative field.

HSM BEFORE AND AFTER EXECUTION		EXAMPLE 3			EXAMPLE 4		
	20	00	01	02	00	01	02
		01	23	4+	04	27	5⊖

HSM BEFORE EXECUTION		00	01	02	03	04	05	06	07	08	09
	21	-	d	d	d	.	d	d	⊖	C	R

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	ED	MASK (10), AMT

	Example 3										Example 4									
HSM AFTER EXECUTION	21	00	01	02	03	04	05	06	07	08	09	21	00	01	02	03	04	05	06	07	08	09
		-	-	1	2	.	3	4	-	-	-		-	-	4	2	.	7	5	⊖	C	R
TRIGGER →		0 → 1 → 0 →											0 → 1 →									
		CONDITION CODE = 2											CONDITION CODE = 1									

Note that in Example 3 the significance trigger is set to zero by the plus sign in the packed sending field. In Example 4, however, the minus sign in the packed field does not set the trigger back to zero.

Example 5

Editing multiple fields.

AMTS

HSM BEFORE AND AFTER EXECUTION		00	01	02	03	04	05
	20	01	23	7+	00	29	5-

MASK

HSM BEFORE EXECUTION		00																			21
	21	-	d	d	S	.	d	d	C	R	f	f	f	-	d	d	S	.	d	d	C	R	f

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	ED	MASK(22), AMTS

MASK

HSM AFTER EXECUTION		00																			21
	21	-	-	1	2	.	3	7	-	-	-	-	-	-	-	-	2	.	9	5	C	R	-
SIGNIFICANCE TRIGGER		0 → 1 → 0 → 1 → 0																					

CONDITION CODE = 1 (Based on last field processed)

It should be noted that the field separator character resets the significance trigger to zero, so that proper suppression of unwanted characters will take place in the next field.

As can be seen in the previous examples the value in the Edit Mask controls execution of the instruction and the insertion of digits from the packed field.

Exercise:

	VAL		ACC		BOH				DEST							
	↓		↓		↓				↓							
21	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
	01	24	7+	00	00	0+	00	00	47	21	5⊖	15	0+	27	50	1+

Based on the packed format and symbolic values assigned as above, show the result of each instruction in the locations provided and based on the mask as shown in Column II.

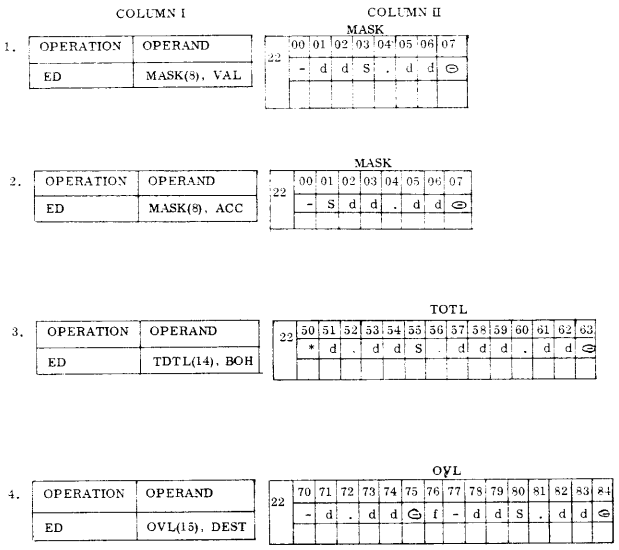
Symbols representing characters in the mask are as follows:

- = BLANK s = SIGNIFICANCE START

⊖ = MINUS f = FIELD SEPARATOR

d = DIGIT SELECT , . = INSERTION CHARACTERS

* = ASTERISK



This exercise requires the preparation of an edited output record from an input record in the following format:

ACCOUNT NO.	8	CHARS
TOTAL DEPOSITS	7	CHARS
TOTAL CHECKS	7	CHARS
PREVIOUS BALANCE	7	CHARS

The following processing steps are required:

a. Add the Total Deposits to the Previous Balance
b. Subtract the Total Checks from the Previous Balance
c. Prepare an output record in the edited format.

The output record is in the format:

	ACCOUNT NO.	8	CHARS
	(BLANKS)	4	CHARS
*	TOTAL DEPOSITS	12	CHARS
	(BLANKS)	4	CHARS
*	TOTAL CHECKS	12	CHARS
	(BLANKS)	4	CHARS
*	PRESENT BALANCE	12	CHARS
	(BLANKS)	76	CHARS

* EDIT FORMAT
$-ZZ,ZZZ.DDS

- = BLANK
Z = SUPPRESSED ZERO (BLANK) OR DIGIT
D = DIGIT
S = SIGN

Prepare assembler statements for allocating storage memory and constants. Routine coding does not include input or output instructions.

COMPARISON AND BRANCHING

There are two instructions that test the relative value of two operands. The Compare Logical instruction tests the relative binary value of two operands. The Compare Decimal instruction tests the relative algebraic value of two operands that are in packed format.

Both instructions set the Condition Code based on the relative value of the operands.

COMPARE LOGICAL (CLC) INSTRUCTION

The Compare Logical instruction tests the relative binary value of two equal-length operands. The two operands may be in either packed or unpacked format. The instruction operates from left to right comparing the bit values in a byte from each field. The instruction terminates when either inequality is found or, if both operands are equal in value, when the last byte in each field has been compared.

The values of the operands remain unchanged in memory.

Example:

(Comparison of Key Criteria Fields)
(Character values shown)

HSM BEFORE AND AFTER EXECUTION

MACN

	00	01	02	03	04	05	06	07
27	7	5	8	4	3	1	2	F

TACN

	00	01	02	03	04	05	06	07
28	7	5	8	4	3	1	2	D

ASSEMBLY INSTRUCTION

OPERATION	OPERAND
CLC	MACN(8), TACN

GENERATED INSTRUCTION

OP	L	B_1	D_1	B_2	D_2
$D5_{16}$	7	1_{10}	2700_{10}	1_{10}	2800_{10}

General Register one contains 0000

CONDITION CODE = 2 (FIRST OPERAND HIGH)

Example:

(Comparison of Address Fields)

HSM BEFORE AND AFTER EXECUTION

	ADR1		ADR2	
	00	01	02	03
20	4007_{10}		4017_{10}	
	$OFA7_{16}$		$OFB1_{16}$	

ASSEMBLY INSTRUCTION

OPERATION	OPERAND
CLC	ADR1(2), ADR2

GENERATED INSTRUCTION

OP	L	B_1	D_1	B_2	D_2
$D5_{16}$	1	1_{10}	2000_{10}	1_{10}	2002_{10}

General Register one contains 0000

CONDITION CODE = (FIRST OPERAND LOW)

COMPARE DECIMAL (CP) INSTRUCTION

The Compare Decimal instruction tests the relative algebraic value of two packed operands. The operands may be of unequal length. However, the first operand should be longer if the operands are unequal. If the second operand is longer than the first, the excess bytes do not enter into the comparison. If the second operand is shorter in length it is assumed to contain high-order zeros.

The instruction operates from right to left. As the rightmost half-byte contains the sign, these respective half-bytes are compared first. If the signs are unlike, the Condition Code is set to reflect the relative algebraic value of the operands and the execution of the instruction is terminated.

If the signs are alike, the execution of the instruction is terminated when the leftmost byte of the first operand has been compared with the actual (or zero-extended) relatively positioned byte of the second operand. The Condition Code setting, in this case, is also based on the relative algebraic values of the operands.

Example:

HSM BEFORE AND AFTER EXECUTION

	AMT				VAL			
	00	01	02	03	20	21	22	
50	01	39	64	2⊖	51	02	34	5+

INSTRUCTION

OPERATION	OPERAND
CP	AMT(4), VAL(3)

GENERATED INSTRUCTION

OP	L_1	L_2	B_1	D_1	B_2	D_2
$F9_{16}$	3	2	2_{10}	0904_{10}	2_{10}	1024_{10}

General Register two contains 4096_{10}

CONDITION CODE = 1 (FIRST OPERAND LOW)

Example:

HSM BEFORE AND AFTER EXECUTION

	CHK				BAL			
	02	03	04		20	21	22	23
40	12	39	4+		41	09	12	39	4+

INSTRUCTION

OPERATION	OPERAND
CP	CHK(3), BAL(4)

	OP	L_1	L_2	B_1	D_1	B_2	D_2
GENERATED INSTRUCTION	FC_{16}	2	3	1_{10}	4002_{10}	2_{10}	0024_{10}

General Register one contains 0000
General Register two contains 4096_{10}

CONDITION CODE = 0 (OPERANDS EQUAL)*

* Note that because the second operand was longer than the first operand the Condition Code does not reflect the true relative value of each field.

Had the operands been reversed, i.e., BAL (4), CHK (3), the Condition Code would have been set to 2 (first operand high).

BRANCH ON CONDITION INSTRUCTION

The Branch On Condition (BC) instruction transfers control based on the setting of the Condition Code indicator.

The BC is a four-byte instruction with the second byte being a mask specifying in the four high-order bits the Condition Code setting(s) upon which the transfer of control depends.

A 1 bit in the respective bit positions below generates a transfer of control if the Condition Code Indicator is set to the position shown.

2^4 = Condition Code 3
2^5 = Condition Code 2
2^6 = Condition Code 1
2^7 = Condition Code 0

The least significant four bits of the mask (2^0 to 2^3) must be zero.

In assembly language, however, the mask is specified as one hexadecimal digit and the four least-significant zero bits will be generated.

Example:

In the following example, assume that the BC instruction follows a decimal subtract instruction and the programmer wants to transfer control to an error routine (ERRT) if overflow has occurred or to an overdraft (OVDF) routine if the result of the subtraction is negative. For a positive or zero result, he enters a process (PRCS) routine.

The coding would be:

NAME	OPERATION	OPERAND	
	SP	BAL (4). AMT (3)	SUBTR. AMT. FROM BAL.
CC3	BC	X'1'. ERRT	BR. TO ERROR RTN
CC1	BC	X'4'. OVDF	BR. TO OVERDR. RTN
PRCS			ENTER PROC. RTN.

An unconditional transfer of control takes place if all the high-order bits have a value of 1.

The Branch (B) operation code simplifies the writing of this instruction. A mask of X'FO' (11110000_2) is generated automatically.

Thus, each of the following generates an unconditional transfer to STRT.

OPERATION	OPERAND
BC	X'F', STRT

OPERATION	OPERAND
B	STRT

BRANCH AND LINK INSTRUCTION

This instruction performs an unconditional branch and stores a return address in a specified register.

It is a four-byte instruction with the four high-order bits of the second byte specifying a general register used for storage of the P Register. (The P Register after staticizing, contains the address of the next instruction.) The P Register address is stored before branching takes place.

The register that stores the P Register may be used for return linkage as in the example shown below.

Example:

In the following example, a BRANCH AND LINK instruction transfers control to a routine at location EDIT.

NAME	OPERATION	OPERAND
TRFED RETRN	BAL	10, EDIT

The P Register (containing the address of RETRN) is stored in General Register 10. At the conclusion of the edit routine, the same register may be used for return as follows:

NAME	OPERATION	OPERAND
EDIT	BC	X'F', 0(10)

BRANCH ON COUNT INSTRUCTION

This instruction allows a program to loop through a routine a given number of times.

A general register holds the count of the number of times the loop is to be executed. Each time the Branch on Count is executed, the register is decremented by one (binary count). When it has been decremented to zero, the branch address is ignored and the next sequential instruction is executed.

Example:

In the following section of a program the routine EDTB is to be executed seven times before proceeding on to the next routine CONT.

NAME	OPERATION	OPERAND
EDTB		
	BCT	9, EDTB
CONT		

Prior to entering the routine (EDTB), General Register 9 was loaded with a binary value of six (see LOAD MULTIPLE Instruction in this section).

The routine (EDTB) is executed once and repeated the number of times specified by the binary count in General Register 9. When the count is decremented to zero the Branch Address is ignored and the next sequential instruction (CONT) is executed.

SET P2 REGISTER (STP2) INSTRUCTION

This instruction transfers control from the Interrupt State to the Processing State. It sets the P2 Register with the desired value and transfers to the address contained in the P1 Register (Reserved Locations 40 and 41).

The Condition Code Indicator is also reset to the Condition Code that existed at the time the Processing State was interrupted. In addition, the hardware interrupt register is reset by the interrupt mask in reserved memory.

Example:

Assume the following values are stored in HSM immediately before execution of the instruction,

HSM BEFORE EXECUTION

		P1 COUNTER		P2 COUNTER	
00	43	00	40 41	00	44 45
	03_{16}		2300_{10}		4000_{10}

and that ENTR had been assigned a value of 3800_{10} by the Assembler.

The following instruction transfers control to the P1 state, and stores 3800_{10} in the P2 counter.

ASSEMBLY INSTRUCTION

OPERATION	OPERAND
STP2	ENTR

GENERATED INSTRUCTION

OP	M	B_1	D_2
82_{16}	00	1_{10}	3800_{10}

General Register one contains 0000

HSM AFTER EXECUTION

	P1 Counter		P2 Counter	
00	40 41	00	44 45	
	2300_{10}		3800_{10}	

TRANSFER CONTROL TO 2300_{10}

CODE RESET TO 3

Exercise:

	BAL		VAL		AMT		NUM				COST			UNIT				
	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68
24	00	12	4+	12	74	5⊖	98	21	2+	98	42	17	0+	00	00	1+	00	12	4⊖

Using the memory layout above, indicate which Condition Code would be set after execution of the following instructions.

COMPARE DECIMAL

	OPERATION	OPERAND	COND. CODE
1.	CP	BAL (3), UNIT + 1 (2)	_____
2.	CP	VAL (3), COST (3)	_____
3.	CP	AMT (3), NUM (4)	_____

	DEF ↓				GHI ↓		MNO ↓						PQR ↓			JKL ↓		ABC ↓			
25	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
	F0	F1	F3	F7	AA	AA	AB	CD	EF	01	24	57	AB	CD	EF	AA	A0	F0	F1	F4	F7

Based on the above, show which Condition Code would be set following execution of the three instructions below. (Hexadecimal values of bytes shown.)

COMPARE LOGICAL — COND. CODE

	OPERATION	OPERAND	
4.	CLC	ABC (4), DEF	_____
5.	CLC	GHI (2), JKL	_____
6.	CLC	MNO (8), PQR	_____

	ONE ↓		TWO ↓			THREE ↓			FOUR ↓				FIVE ↓			SIX ↓		
24	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68
	00	12	4+	12	74	5⊖	98	21	2+	98	42	17	0+	00	00	1+	00	12	4⊖

Based on the above, indicate which instruction would be executed next.

NI = NEXT SEQUENTIAL INSTRUCTION

7.

OPERATION	OPERAND
CP	SIX + 1 (2), ONE (3)
BC	X'B', UPD

NI _____ UPD _____

8.

OPERATION	OPERAND
CP	FIVE (3), TWO (3)
BC	X'7', NEWD

NI _____ NEWD _____

9.

OPERATION	OPERAND
CP	THRE (3), FOUR + 1 (3)
BC	X'D', MIST

NI _____ MIST _____

LOAD AND STORE REGISTER

There are two instructions that enable the programmer to address registers so that he may either load a value in the register(s) selected, or store the contents of the register(s) in memory.

The four-byte memory locations used for either storing or loading registers must be word-oriented.

LOAD MULTIPLE

This instruction places designated value(s) into one or more consecutively numbered general registers.

The following instruction:

NAME	OPERATION	OPERAND
	LM	3, 3, ALPHA

loads General Register 3 with the value stored in the word-oriented four-byte addresses by ALPHA.

The instruction below loads General Registers 2, 3, 4, 5, 6, and 7 with the values stored in the word-oriented 24 bytes addressed by ALPHA.

NAME	OPERATION	OPERAND
	LM	2, 7, ALPHA

Example:

HSM BEFORE AND AFTER EXECUTION

ALPHA

24	00	04	08	12	16	20
	1000_{10}	2000_{10}	3000_{10}	5000_{10}	6000_{10}	7000_{10}

GENERAL REGISTERS 1-8 BEFORE EXECUTION

1	2
0000_{10}	4096_{10}
3	4
8192_{10}	12288_{10}
5	6
16384_{10}	20480_{10}
7	8
24576_{10}	28672_{10}

ASSEMBLY INSTRUCTION

NAME	OPERATION	OPERAND
	LM	2, 7, ALPHA

GENERATED INSTRUCTION

OP	R_1	R_3	B_2	D_2
98_{16}	2_{10}	7_{10}	1_{10}	2400_{16}

GENERAL REGISTER 1 = 0000_{10} (AS ABOVE)

GENERAL REGISTERS 1-8 FOLLOWING EXECUTION

1	2
0000_{10}	1000_{10}
3	4
2000_{10}	3000_{10}
5	6
5000_{10}	6000_{10}
7	8
7000_{10}	28672_{10}

STORE MULTIPLE

This instruction places the contents of one or more consecutively numbered general registers into memory. The locations that store the register(s) must be word-oriented. A full word is used for the storage of the contents of each register.

The Operand format is the same as for the LOAD MULTIPLE instruction.

Example:

GENERAL REGISTERS 1-8 BEFORE AND AFTER EXECUTION

1	2
0000_{10}	1000_{10}
3	4
2000_{10}	3000_{10}
5	6
5000_{10}	6000_{10}
7	8
7000_{10}	28672_{10}

HSM BEFORE EXECUTION

ALPHA

24	00	04	08	12	16	20
	0000_{10}	0000_{10}	0000_{10}	0000_{10}	0000_{10}	0000_{10}

ASSEMBLY INSTRUCTION

NAME	OPERATION	OPERAND
	STM	2, 7, ALPHA

GENERATED INSTRUCTION

OP	R_1	R_3	B_2	D_2
90_{16}	2_{10}	7_{10}	1_{10}	2400_{10}

GENERAL REGISTER 1 = 0000_{10} (AS ABOVE)

HSM AFTER EXECUTION

24	00	04	08	12	16	20
	1000_{10}	2000_{10}	3000_{10}	5000_{10}	6000_{10}	7000_{10}

BINARY ARITHMETIC INSTRUCTIONS

In the two binary arithmetic instructions, Add Binary and Subtract Binary, the length of both operands is controlled by the length of the first.

As in decimal arithmetic operations, if the operands are unequal, the second operand is truncated, if longer, or, is extended with zero-value bytes, if shorter.

Both instructions operate from right to left. The instruction is terminated when the left-end byte has been processed.

The Condition Code Indicator is set based on the result as follows:

Condition Code	Add Binary	Subtract Binary
0	Result is Zero	
1	Not Used	Difference Less Than Zero
2	Result is Greater Than Zero	
3	Overflow	Not Used

Example:

Assume an input tape that contains a block of five 80-character records.

For processing, the programmer moves each record to a separate processing area. The Add Binary instruction increments the second address of the instruction that moves a record to the processing area.

The Subtract Binary instruction, with a branch to read if the input area has been exhausted, is used to determine when the last record is processed.

The Input Block, Record Processing, and constant areas could be allocated as follows:

ALLOCATION OF INPUT AND RECORD PROCESSING AREA

NAME	OPERATION	OPERAND	HSM ALLOCATION
	ORG	3000	
INP	DS	400C	3000-3399
RPR	DS	80C	3400-3479
	ORG	RPR	
ACCT	DS	8C	3400-3407
NAME	DS	25C	3408-3432
ADR	DS	30C	3433-3462
AMT	DS	10C	3463-3472
FILL	DS	7C	3473-3479

ALLOCATION OF CONSTANTS

NAME	OPERATION	OPERAND	STORED VALUE OF CONSTANTS (HEX. FORMAT)
RDIN	DC	A(INP)	3480-3481 0B B8
INCR	DC	A(80)	3482-3483 00 50
TLY	DC	X'0505'	3484-3485 05 05
CTR	DC	X'01'	3486

The functional chart on the following page shows the matching coding steps. Only the steps pertinent to the Binary Arithmetic Instructions are shown.

Example:

Add Binary of Previous Instruction Binary Value

HSM BEFORE AND AFTER EXECUTION

	INCR			
	34	82	83	0000 0000 0101 0000
		00	50	

HSM BEFORE EXECUTION

	IN4+4			
	40	16	17	0000 1011 1011 1000
		0B	B8	

ASSEMBLY INSTRUCTION

NAME	OPER	OPERAND
IN5	AB	IN4+4(2),INCR(2)

GENERATED INSTRUCTION

OP	L_1	L_2	B_1	D_1	B_2	D_2
F6	1	1	1_{10}	4016_{10}	1_{10}	3482_{10}

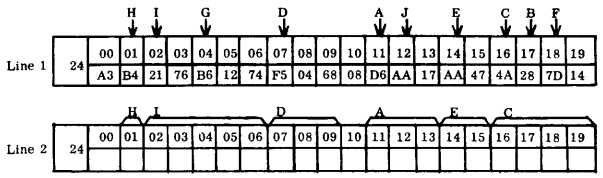
General Register one contains 0000

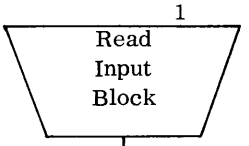
HSM AFTER EXECUTION

	40	16	17	0000 1100 0000 1000
		0C	08	

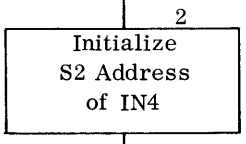
CONDITION CODE = 2

Exercise:

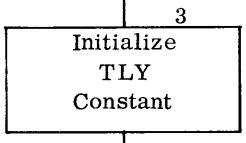




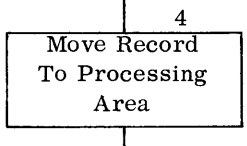
NAME	OPERATION	OPERAND
IN2	MVC	IN4 + (2), RDIN



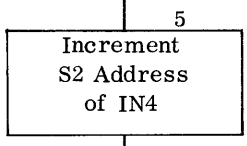
NAME	OPERATION	OPERAND
IN3	MVC	TLY (1), TLY + 1



NAME	OPERATION	OPERAND
IN4	MVC	RPR (80), INP

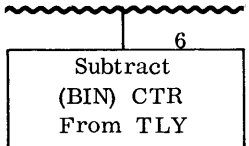


NAME	OPERATION	OPERAND
IN5	AB	IN4 + 4 (2), INCR (2)

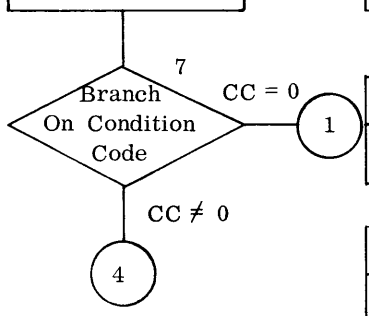


RECORD PROCESSING STEPS

RECORD PROCESS CODING NOT SHOWN



NAME	OPERATION	OPERAND
IN6	SB	TLY (1), CTR (1)



NAME	OPERATION	OPERAND
IN7	BC	X'8', IN1

NAME	OPERATION	OPERAND
	B	IN4

Indicate the results of each instruction (Column I below) in the locations on Line 2, page 42. Show the results in hexadecimal format. Consider each question independently based on the contents of the locations of Line 1. In Column II show the Condition Code that will be set following the execution of each instruction.

Column I

ADD BINARY

	OPERATION	OPERAND
1.	AB	A(3), B(3)
2.	AB	C(4), D(4)
3.	AB	E(2), F(2)

Column II

CONDITION CODE

_____

_____

_____

SUBTRACT BINARY

	OPERATION	OPERAND
4.	SB	D(3), G(3)
5.	SB	H(1), I(1)
6.	SB	I(5), J(2)

_____

_____

_____

LOGICAL INSTRUCTIONS

INTRODUCTION

The Logical Instructions perform operations on the individual bits of a byte. The operation works from the left to right on equal length operands (256 maximum). Proper parity is generated for each byte based upon the eight least-significant bits.

The three principal logical operations are AND (result is one if and only if both bits are one), OR (result is one if either or both bits are one), EXCLUSIVE OR (result is one if either but not both bits are one). One additional logical operation is a test comparison with a specified mask.

AND INSTRUCTION

The rule of the AND instruction is that a 1 bit in the same relative bit position of both operands produces a 1 bit in the same position of the result. Any other combination of bits produces a zero bit in the result.

0	+ 0	= 0
0	+ 1	= 0
1	+ 0	= 0
1	+ 1	= 1

Example:

	AD1		BIT CONFIGURATION
HSM BEFORE EXECUTION	30	00 01	0000 0000 1001 1010
		00 9A	

INC

HSM BEFORE AND AFTER EXECUTION	31	00 01	1111 1111 1111 1010
		FF FA	

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	NC	AD1(2), INC

GENERATED INSTRUCTION	OP	L	B_1	D_1	B_2	D_2
	$D4_{16}$	1	1_{10}	3000_{10}	1_{10}	3100_{10}

General Register one contains 0000

	AD1		
HSM AFTER EXECUTION	30	00 01	0000 0000 1001 1010
		00 9A	

CONDITION CODE = 1

OR INSTRUCTION

This instruction may be used to insert 1 bit(s) in any bit position (s) of a byte.

The rule of OR is that a 1 bit in the same relative position of either field produces a 1 bit in the same position of the result.

0	+ 0	= 0
0	+ 1	= 1
1	+ 0	= 1
1	+ 1	= 1

Example:

	CH		BIT CONFIGURATION
HSM BEFORE AND AFTER EXECUTION	37	00 01 02	0000 0000 0000 0000 0000 0001
		00 00 01	

BAL

HSM BEFORE EXECUTION	40	80 81 82	0000 0111 1000 1001 1000 1100
		07 89 8C	

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	OC	BAL(3), CH

GENERATED INSTRUCTION	OP	L	B_1	D_1	B_2	D_2
	$D6_{16}$	2	1_{10}	4080_{10}	1_{10}	3700_{10}

General Register one contains 0000

BAL

HSM AFTER EXECUTION	40	80 81 82	0000 0111 1000 1001 1000 1101
		07 89 8D	

CONDITION CODE = 1

EXCLUSIVE OR INSTRUCTION

The Exclusive Or instruction extracts 1 bit(s) in specified bit position(s) of one or more bytes.

The Exclusive Or may also be used to alternate designated bits so that they will have a value of 1 the first time and 0 the second time the Exclusive Or instruction is performed. In this case the modifying mask has a one bit in the bit positions where the interchange is desired.

The rule of Exclusive Or may be considered the same as binary addition. However, there is no carry from one bit to the next.

0	+	0	=	0
1	+	0	=	1
0	+	1	=	1
1	+	1	=	0

Example:

	INH		BIT CONFIGURAGIONS
HSM BEFORE AND AFTER EXECUTION	31	10 / 3F	0011 1111
HSM BEFORE EXECUTION	00	49 / 3F	0011 1111

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	XC	INH(1), X'0031'

GENERATED INSTRUCTION	OP	L	B_1	D_1	B_2	D_2
	$D7_{16}$	0	1_{10}	3110_{10}	1_{10}	0049_{10}

General Register one contains 0000

HSM AFTER EXECUTION *	00	49 / 00	0000 0000

CONDITION CODE = 0

Note that this example has set the Interrupt Mask to prohibit interrupt from any I/O channel.

The same mask applied again will set the Interrupt Mask (location 0049) to allow interrupt from any channel.

USE OF LOGICALS

There are many programming situations where the Logical instructions are useful. For example, a program switch may be a Branch On Condition instruction. Following the BC instruction is a section of coding which is bypassed if the Branch takes place. When such a condition is desired, a Logical instruction may be used to insert all one bits in the mask of the BC making it an Unconditional Branch. When execution of the coding following the BC is desired, a logical instruction that inserts all zero bits in the mask may be used. This makes the BC a 'no-Op' instruction.

Logical instructions can alter the value of a field. A logical instruction may change the sign of a packed field from a plus sign $(1100)_2$ to a minus sign $(1101)_2$. This feature is useful when editing the packed field. The minus sign allows the insertion of editing symbols to the right of the digits in an edited field. Thus a field may be made pseudo-negative for fields of a prescribed value. For example, if an asterisk is desired to the right of any edited balance field below $100.00, the packed field sign position could be altered to a negative sign. (See OR example.)

The Condition Code Indicator is set by the Logical instructions. It is set to zero if all of the bits in the result field are zero. It is set to one if any of the result bits are one.

TEST UNDER MASK INSTRUCTION

This instruction compares the relatively positioned bits of a byte with a mask byte and indicates the result by a setting of the Condition Code Indicator.

The mask byte is written as the second byte of the TM instruction. The S1 address is the location of the byte to be tested.

A one bit in the mask tests the presence of a one bit in the corresponding bit position of the byte addressed.

The Condition Code Indicator is set to zero if all of the selected bits are zero (or if the mask is all zeros). The setting is one if the selected bits are a mixture of zeros and ones. Condition Code three is set if the selected bits are all ones. Condition Code two is not set by this instruction.

Example #1

HSM BEFORE AND AFTER EXECUTION	60	10 / $0101\ 1100_2$

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	TM	LOC, X'OF'

GENERATED INSTRUCTION	OP	M	B_1	D_1
	91_{16}	$0000\ 1111_2$	2_{10}	1914_{10}

General Register two contains 4096_{10}

CONDITION CODE = 1

Example #2

HSM BEFORE AND AFTER EXECUTION	00	49 / 0011 1101

ASSEMBLY INSTRUCTION	OPERATION	OPERAND
	TM	X'31', X'02'

GENERATED INSTRUCTION	OP	M	B_1	D_1
	91_{16}	$0000\ 0010_2$	1_{10}	0049_{10}

General Register one contains 0000

CONDITION CODE = 0

DATA TRANSLATION

The Translate instruction provides the user with the ability to accept data in 'foreign' code, and translate and process it in the 'native' code of the computer being used. The data may then be translated back to the 'foreign' code by the Translate instruction, if desired, prior to the output operation.

The Translation process employs a table that is addressed by each data character to be translated. The binary value of the data character being translated is added to the address of the 256-byte translation table. The character in that position of the table replaces the data character.

The Translate instruction may be used for preserving the security of information being transmitted over communication lines. Data may be translated in an unintelligible code by the use of a program and one or a series of translation tables at the sending location, and sent to the receiving location where a similar program and table(s) re-translates the data to its intelligible form.

Another use of the Translate instruction is to validate data. As an example, assume that a ten-character amount field must contain all numeric (unpacked) characters.

The numeric characters, hexadecimal F0 through F9, have binary values equal to 240_{10} to 249_{10}. Thus, the table could be constructed as follows with all zero bit-filled bytes in these positions such as the following:

NAME	OPERATION	OPERAND
EDTB	DS	240C
	DS	X'00000000000000000000'
	DS	6C

The first 240 and the last 6 bytes of the table would be filled with 1 bits. Thus, any character in the data field except an unpacked numeric can address a byte filled with one bits.

If the original data field must be preserved, the translation may be performed from a Work area. Assume the data field has been transferred to a work area and appears as follows: (shown in hexadecimal)

WAMT

	00 01 02 03 04 05 06 07 08 09
22	F0 F0 F2 F7 F6 F3 C3 F2 F8 F1

It should be noted that all characters are numeric (unpacked) except the hexadecimal C3 in position 2006. Thus, after translation this byte will be filled with one bits and the others filled with zero bits.

The Translate instruction is written as in the following format. (Assume the translation table begins at 2200.)

WAMT

HSM BEFORE EXECUTION

	00 01 02 03 04 05 06 07 08 09
20	F0 F0 F2 F7 F6 F3 C3 F2 F8 F1

ASSEMBLY INSTRUCTION

NAME	OPERATION	OPERAND
	TR	WAMT(10), EDTB

GENERATED INSTRUCTION

OP	L	B_1	D_1	B_2	D_2
DC_{16}	9_{10}	2_{10}	0000_{10}	2	0200

General Register 2 = 2000

WAMT

HSM AFTER EXECUTION

	00 01 02 03 04 05 06 07 08 09
20	00 00 00 00 00 00 FF 00 00 00

The presence of any 1 bits in the translated field indicates that the original data field contained an invalid character. The field could be tested against a zero-bit-filled field with a Compare Logical instruction as follows:

NAME	OPERATION	OPERAND
	CLC	WAMT(10), EDTB+239
	BC	X'7', INV
VAL		

The Compare Logical instruction tests the field for all zero bits. The Branch instruction transfers to a routine for handling a field that is invalid (INV). If the field is all zero, the routine for handling a field that is valid (VAL) will be entered.

As another example of the Translate instruction for validation, assume a field that consists of either alphabetic or numeric characters. Any other character would make the field invalid.

The binary values of all alphabetic and numeric characters range as follows:

A to I inclusive = 193_{10} to 201_{10}

J to R inclusive = 205_{10} to 217_{10}

S to Z inclusive = 226_{10} to 233_{10}

0 to 9 inclusive = 240_{10} to 249_{10}

The table could be allocated by the DC (Define Constant) code as follows:

NAME	OPERATION	OPERAND	
TABLE	DC	193x'FF'	
	DC	9x'00'	A to I
	DC	7x'FF'	
	DC	9x'00'	J to R
	DC	8x'FF'	
	DC	8x'00'	S to Z
	DC	6x'FF'	
	DC	10x'00'	0 to 9
	DC	6x'FF'	

Then, using instructions similar to those in the preceding example, a field could be tested for alpha or numeric content.

Any other character will translate into a 1 bit-filled byte. For example, assume the field contained a period, an invalid character. A period (.) has a binary value of 01001011_2 or 75_{10}. It would address the 75th position of the table and be replaced by a 1 bit-filled byte.

The previous examples have illustrated the Translate instruction for validation. A table of this type is fairly easy for the programmer to construct using the DC and DS codes.

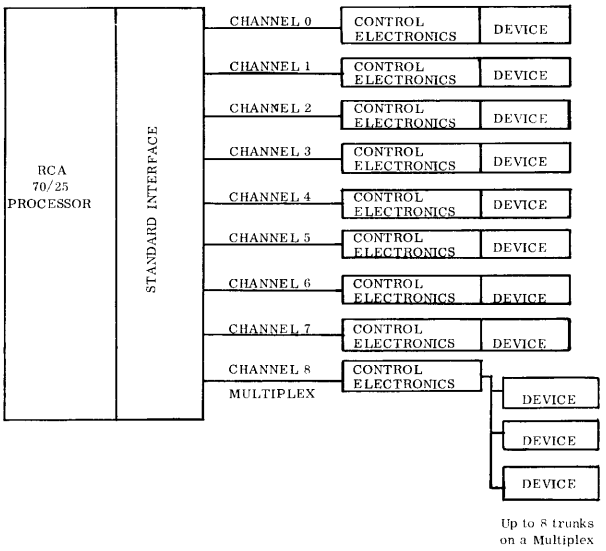
For translating from one code to another, such as from the ASCII to the EBCDIC code a prepared table could be used which would be entered into memory from an input device.

INPUT/OUTPUT

INTRODUCTION

The user may elect to control Input/Output by Assembly I/O instructions or by software I/O subroutines. This section describes the eight Assembly I/O instructions, and the methods of error recognition and recovery.

Up to nine channels connect the RCA 70/25 mainframe to I/O devices. Each channel has its own Control Electronics, and is capable of the independent execution of I/O commands. A feature called MULTIPLEXING makes it possible to connect many devices, printers, card readers, etc., to a single channel without the loss of simultaneous capabilities.



Two of the eight channels may be used for high-speed transmission, and the remaining six for medium speed, or all eight may be used for the medium speed. The high-speed channels must be the first two (0 and 1) channels.

To these channels may be added the ninth channel, a MULTIPLEX that accommodates up to eight trunks.

The total theoretical data rate available on the 70/25 is 667KB. Because the medium-speed channels and multiplexor take more time to process each byte than the high-speed channels, devices operating on the slower channels must have their peak data rates multiplied by a weighting factor in the determination of total equivalent data rate. These weighting factors are 2.5 for a device operating on a medium-speed channel, and 4.5 for a device operating in multiplex mode on the multiplexor channel. (The multiplexor, however, cannot exceed a total data rate of 111KB.)

The following configuration is assumed for all examples in this section.

CHANNEL	UNIT	DEVICE
1	1	CARD READER
2	1	CARD PUNCH
3	1	PRINTER
4	1,2,3,4	TAPE STATIONS

READING DATA

Data is read into High-Speed Memory (HSM) by two Read Commands. The Read Device Forward can address all peripheral equipment on line to the system. The Read Device Reverse command can be issued only to tape stations.

READ INSTRUCTIONS

OPERATION	OPERAND
RDF	T(D), $D_1(B_1)$, $D_2(B_2)$
RDR	T(D), $D_1(B_1)$, $D_2(B_2)$

These commands select a Channel and Device, and indicate what HSM area is to receive the data.

Assuming the two HSM areas:

INPT (HSM 1000-1100)
OUTP (HSM 1200-1319)

The instruction,

OPERATION	OPERAND
RDF	1(1), INPT, INPT + 79

reads a card from the Card Reader (Trunk 1) into HSM location starting at 1000 (INPT) and ending at HSM 1079 (INPT+79). Base addresses are implied.

The same instruction with the Trunk number changed:

OPERATION	OPERAND
RDF	4(1), INPT, INPT + 79

reads a block from magnetic tape filling eighty characters of HSM (1000-1079).

The Read Device Reverse (RDR) can be issued only to magnetic tape. The instruction:

OPERATION	OPERAND
RDR	4(1), INPT + 79, INPT

causes the magnetic tape to be moved in a reverse direction. The first byte read is placed in INPT+79 (HSM 1079), the second byte is placed in INPT+78, etc., and the last byte read is placed in INPT (HSM 1000). Note that the addresses had to be reversed.

Once staticized and accepted by the channel Control Electronics, I/O instructions are executed in a simultaneous manner, leaving the processor free to staticize the next instruction in sequence.

The 70/15 Read Auxiliary instruction (RDA) is recognized and executed on the 70/25 as a Read Device Forward, and does not initiate an Operation Code Trap interrupt.

Notice that both the RDF and RDR instructions are terminated by:

1. Reaching a gap on magnetic tape or reading one card, or
2. Reading the amount of data specified by the address operands.

WRITING DATA

The Write instruction (WR) transfers data from HSM to the selected device.

The instruction:

OPERATION	OPERAND
WR	3(1), OUTP, OUTP + 79

prints the contents of HSM 1200-1279 (OUTP area) to the Printer. Base addresses are implied.

The same command with channel number changed:

OPERATION	OPERAND
WR	4(1), OUTP, OUTP + 79

writes a block of eighty bytes to tape 1 on channel 4, or

OPERATION	OPERAND
WR	2(1), OUTP, OUTP + 79

punches a card.

The Erase (WRE) instruction is a Write command; however, it can only erase a section of magnetic tape. The instruction,

OPERATION	OPERAND
WRE	4(2), OUTP, OUTP + 50

erases an area of tape equal to the length of 51 bytes.

CONTROLLING PERIPHERAL DEVICES

The instructions discussed above serve the function of moving data in and out of HSM. The Write Control (WRC) command has the function of communicating control information (rewind tape, paper advance, pocket select, etc.) to a specific device. The instruction transmits a byte from HSM to the Control Electronics of the selected device. The bit configurations of these control bytes are illustrated below.

CARD READER Control Byte

2^0	Select Output Stacker #1
2^1	Select Output Stacker #2
2^2	Translate mode
2^3	Binary mode
2^4-2^7	NOT USED

MAGNETIC TAPE Control Byte

2^0	NOT USED
2^1	NOT USED
2^2	Reread
2^3	Unwind one gap
2^4	Rewind one gap
2^5	Rewind and Disconnect
2^6	Unload without Rewind (cartridge Tape Only)
2^7	Rewind to BT Marker

PRINTER Control Byte

2^0 2^1 2^2 2^3 2^4	=	COUNT	Advance the paper up to fifteen single spaces, or selection of Paper Tape Loop Channel (1-11).
2^5		NOT USED	
2^6	0 =		Paper advance following next print action.
	1 =		Paper advance immediately.
2^7	0 =		Paper advance by 2^0-2^2 count.
	1 =		Paper advance by Paper Tape Loop as selected by the 2^0-2^2 count.

Examples of Write Control

Assuming the following bytes in memory:

CTL1	00000001	$(01)_{16}$
CTL2	00001000	$(08)_{16}$
CTL3	01000010	$(C2)_{16}$

The instructions: (Base Addresses implied)

NAME	OPERATION	OPERAND
	WRC	1(1), CTL1, CTL1
	WRC	4(3), CTL2, CTL2
	WRC	3(1), CTL3, CTL3

will:

1. Select Output Stacker #1 on the Card Reader
2. Unwind one gap on Magnetic Tape 3
3. Advance the paper on the printer two lines immediately.

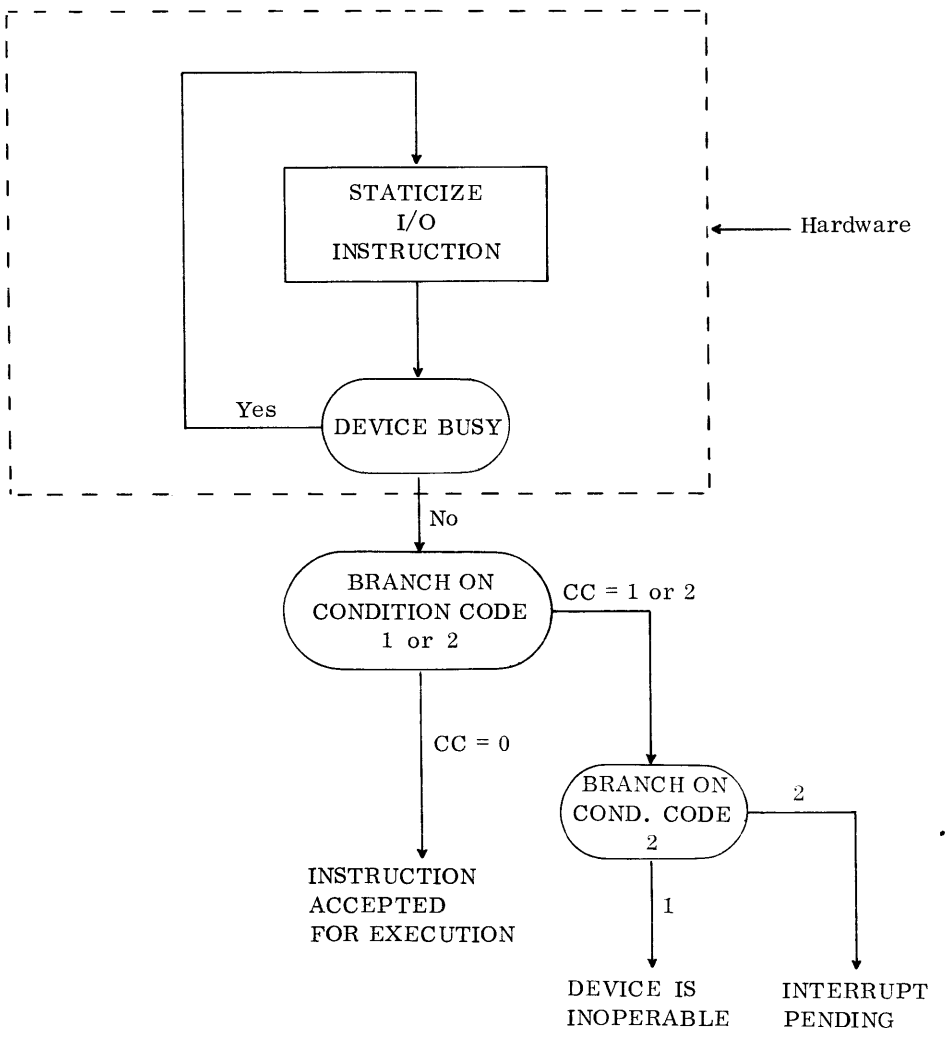
ERROR RECOGNITION

An error condition that develops during the execution of an I/O instruction does not halt the computer.

If the selected device is busy the instruction is re-staticized until the device is available. I/O instructions set the Condition Code to one of three conditions. If the device is inoperable, the instruction terminates and the Condition Code is set to one (1). If the instruction has been accepted for execution by the Control Electronics, the Condition Code is set to zero (0), and the normal mode is free to staticize the next instruction in sequence. If an interrupt is pending on a device, and an instruction selects that device, the instruction terminates and the Condition Code is set to (2).

Note: If interrupt is inhibited on an I/O channel, the termination interrupt generated by the end of a command using that channel, will be pending and the channel will remain busy until a Post Status Instruction resets the busy bit in the Control Electronics.

The Flow Chart (below) indicates both hardware and programming logic for basic I/O.



STATUS INFORMATION

As previously stated, an I/O error does not stop the computer. At the termination of each command, either a software or a user routine determines if the command was completed successfully. The Control Electronics of each channel maintains a STANDARD DEVICE BYTE. This byte contains information about the current status of the channel.

STANDARD DEVICE BYTE

BIT

2^0 ILLEGAL OPERATION - Improper command code for this device. i.e., Read from Card Punch.

2^1 INOPERABLE - The device is unusable until condition is cleared. i.e., No power, jammed, interlock open, etc.

2^2 SECONDARY INDICATOR SET - Indicates that a bit in the "SENSE BYTE" is set. An I/O SENSE operation must be executed to determine the particular condition.

2^3 DEVICE END - Set when device terminates. Indicates that device is available.

2^4 CONTROL BUSY - Channel is engaged in previously initiated operation.

2^5 DEVICE BUSY - Device is engaged in previously initiated operation.

2^6 TERMINATION INTERRUPT -

2^7 MANUAL INTERRUPT PENDING - Interrupt button on Interrogating Typewriter depressed.

When an I/O instruction has been completed, the bit configuration of the Standard Byte indicates how the command was terminated.

- If the 2^2 bit (Secondary Indicator) is set, an error or exceptional condition developed during execution.
- If the 2^6 bit is set, a normal termination is indicated. The instruction was completed successfully.
- The 2^7 bit (Manual Interrupt) differentiates between a termination interrupt and an external interruption initated by the operator at the console typewriter.

The Standard Device Byte is transferred from the Control Electronics into HSM using one of two methods.

One - If I/O Interrupt is not inhibited, hardware automatically stores the Standard Byte in reserved memory location $(46)_{10}$ when interrupts occurs.

Two- If I/O Interrupt is inhibited, it is necessary to execute a Post Status (PS) instruction. This command stores the Standard Byte in one of eight reserved locations, depending on which channel is addressed.

If the instruction:

OPERATION	OPERAND
PS	2(1)

is executed, the Standard Device Byte for Channel 2, device 1 is stored in reserved HSM Location 10.

The Post Status Command does not transfer the Standard Byte into memory, until an I/O instruction has reached some type of termination. If we immediately follow a read or write with a Post Status, the mainframe is "held-off", and simultaneous overlap is lost.

SENSING EXCEPTIONAL CONDITIONS

Once the Standard Device Byte has been stored in HSM, the Secondary Indicator (2^2) can be checked. If the bit is set, it is necessary to transfer a second byte, the Peripheral Unit Sense Byte, from the control Electronics to memory. The format of this byte differs from one peripheral unit to the next, and indicates specifically what error or exceptional condition set the Secondary Indicator. The chart (below) summarizes the meaning of each bit of the Sense Byte for the Card Reader, Card Punch, Magnetic Tapes, Interrogating Typewriter, and Printer.

The Input/Output Sense instruction transfers the "sense" byte into a HSM location determined by the operands of the instruction.

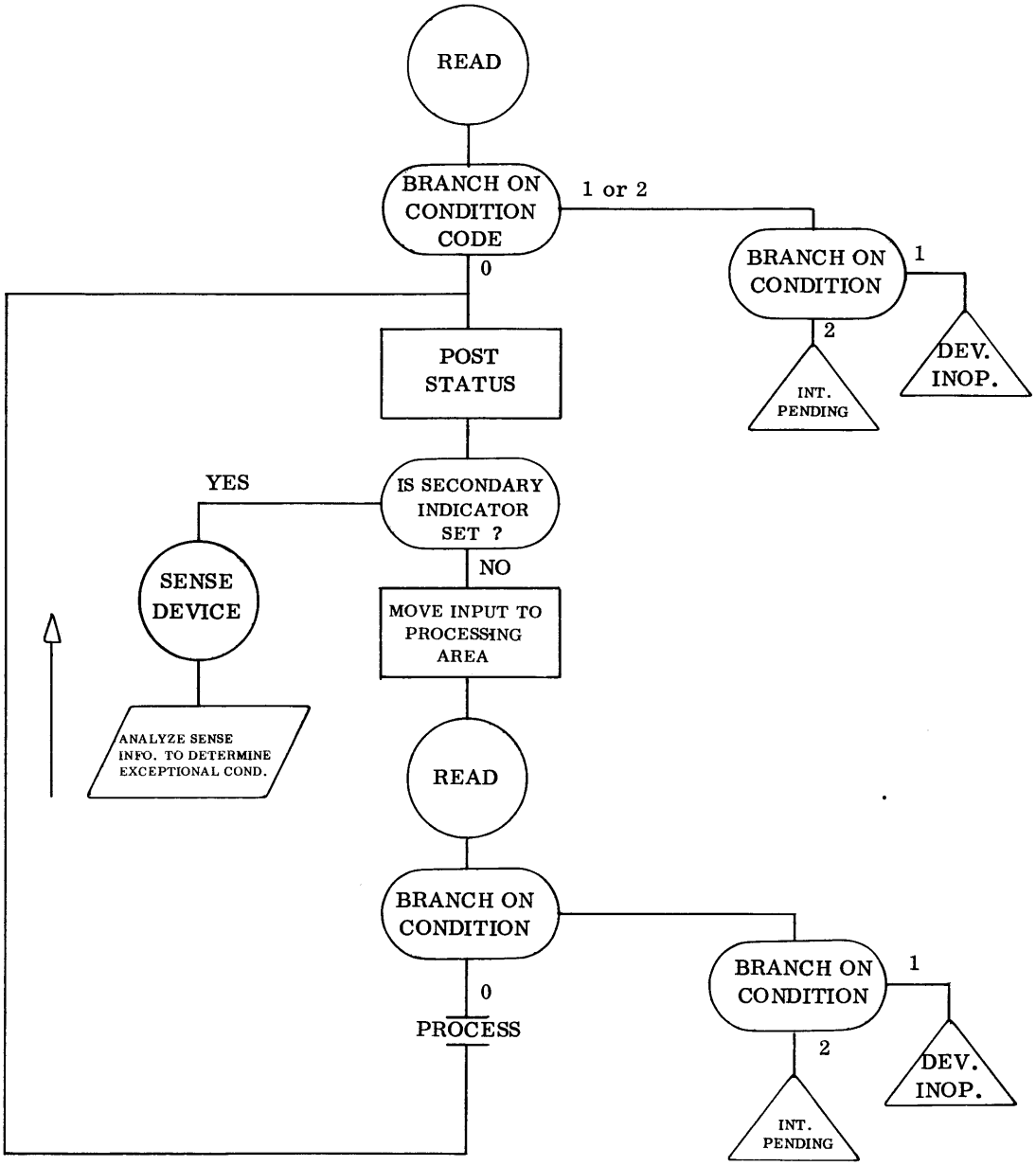
OPERATION	OPERAND
IOS	T(D), $D_1(B_1)$, $D_2(B_2)$

There the byte is analyzed by the Test Under Mask instruction.

Example of I/O Coding

The following sample I/O routine illustrates how the Branch on Condition, Post Status, and I/O Sense instructions facilitate input-output logic. It is not intended to be a complete program; nor is simultaneous logic included. Assume that I/O interrupt is inhibited.

PERIPHERAL UNIT SENSE BYTES					
BIT POSITION	CARD READER	CARD PUNCH	2887-2885 MAG. TAPES	TYPEWRITER	PRINTER
2^0	Tape Mark Code	Not Used	Not Used	Not Used	Parity Error
2^1	Not Used	Not Used	ET or BT	Not Used	Low Paper
2^2	Manual Service in Progress	Manual Service in Progress	Tape Mark	Human Error	Manual Service
2^3	Not Used	Intervention Required	Short Message	Time Out	Non-Printable Code
2^4	Invalid Punch Code	Transmission Parity Error	Transmission Error	Write Error	Transmission Parity Error
2^5	Pocket Selection Too Late	Punch Memory Parity Error	Data Block >Than Count	Not Used	Not Used
2^6	Service Request Not Honored	Not Used	Service Request Not Honored	Not Used	Not Used
2^7	Read Error	Punch Error	Read or Read After Write Error	Not Used	Not Used


READ
BRANCH ON CONDITION CODE
1 or 2
0
BRANCH ON CONDITION
1
2
DEV. INOP.
INT. PENDING
POST STATUS
YES
IS SECONDARY INDICATOR SET ?
NO
SENSE DEVICE
MOVE INPUT TO PROCESSING AREA
ANALYZE SENSE INFO. TO DETERMINE EXCEPTIONAL COND.
READ
BRANCH ON CONDITION
0
PROCESS
BRANCH ON CONDITION
1
2
DEV. INOP.
INT. PENDING

```
NAME      OPERATION  OPERAND                  COMMENTS
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
READ      RDF      4(1),INPT,INPT+99        READ DATA
          BC       X'6',NCDO                BRANCH IF INOPERABLE
          PS       4(1)                     POST STATUS
          TM       18,MSK1                  TEST IF SEC.INDICATOR SET
          BC       X'1',ERR                 BRANCH IF IT IS SET
          B        READ                     BRANCH TO READ
NCDO      BC       X'2',CD2                 TEST FOR INOPERABLE
          H                                 CONDITION ONE
CD2       H                                 CONDITION TWO
ERR       IOS      4(1),SENS,SENS           READ IN SENSE BYTE
          TM       SENS,MSK1                TEST IF FOR TRANS OR READ ERROR
          BC       X'1',                    BRANCH IF ERROR
          B        READ                     BRANCH TO READ
SENS      DS       1C
INPT      DS       100C
MSK1      DS       X'04'
MSK2      DC       X'90'
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
```

Exercise:

T F 1. All I/O OPERATIONS overlap other operations in the 70/25.

T F 2. All I/O instructions are two address (6 byte) format.

T F 3. On the 70/25 one to eight I/O channels are available.

T F 4. If an I/O device is busy when an I/O instruction using that device is attempted, a hold off will occur and the instruction will be restaticized until the device becomes available.

T F 5. After an I/O instruction is performed, a Condition Code setting of 3 indicates Interrupt pending.

T F 6. The direction of operation for all I/O instructions is from left to right.

T F 7. The instruction that tests for an error condition after an I/O instruction is the POST STATUS instruction.

T F 8. Unless inhibited an interrupt takes place after each I/O command.

T F 9. The purpose of I/O termination interrupt is to transfer the Standard Device Byte into HSM.

T F 10. A Post Status Command will terminate, and control will be passed to the next instruction if the channel addressed is busy.

T F 11. If a device is inoperable, and a read or write is directed to it, the Secondary Indicator of the Standard Device Byte is set and interrupt takes place.

12. What are some of the uses of the WRITE CONTROL instruction?

13. What are the formats of the 70/25 I/O instructions?

14. What does the S_1 field contain in an I/O instruction?

15. Write a Read Forward and a Read Reverse instruction which will read data from trunk #2, unit #2, into the area 0100-0110.

16. Write the necessary instructions to:

    a. Read a Card

    b. Punch a Card

    c. Write a message to the console typewriter.

    d. Read a message from magnetic tape.

    Include the sensing, the condition code, POST STATUS, I/O SENSE, for each operation.