SPECTRA 70

RADIO CORPORATION OF AMERICA • ELECTRONIC DATA PROCESSING

SYSTEM

70/15

TRAINING MANUAL

RADIO CORPORATION OF AMERICA

70-15-801

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

FOREWORD

This manual is designed for use in formal training programs which vary in length from about 10 classroom hours (with appropriate outside assignments and work sessions) to 30 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 that should be used in either formal or self-study situations are:

1. 70/15 Assembly Manual
2. 70/15 System Reference Manual

GENERAL DESCRIPTION

INTRODUCTION

The RCA 70/15 is the smallest member of the Spectra 70 Data Processing Systems. It is designed for small-scale data processing applications, and as a low-cost satellite to the other machines in the Spectra 70 System. Equipped with communications capabilities, the RCA 70/15 can be a remote computer for printing, card reading and punching, and magnetic tape transfer to and from locations far removed from the central processor.

70/15 PROCESSOR

The RCA 70/15 Processor is a general-purpose, stored program digital machine with a High-Speed Memory, a Program Control, and an Input/Output connection for the RCA Spectra 70 standard Interface Unit.

HIGH-SPEED MEMORY

The High-Speed Memory (HSM) is a magnetic core device which provides storage and work areas for programs and data. The memory unit can store 4096 bytes; two such units, making a total of 8192 bytes, can be associated with each processor. A byte is the smallest addressable unit in the HSM, and consists of eight information bits and a parity bit. Each byte is binarily addressable; thirteen binary bits are required to express the maximum memory address. The memory cycle time is two microseconds, which is the time it takes to transfer a byte from HSM to a memory register, and to regenerate the byte 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. The instruction is executed after it is staticized.

AUTOMATIC INTERRUPT

The RCA 70/15 can staticize and execute all instructions in one of two programming states; the Processing State is the normal mode of operation for the main program. A condition that causes interrupt transfers the computer to the Interrupt State. Interrupt is mechanized in the 70/15 hardware. It automatically senses the presence of interrupt conditions, and transfers control to the Interrupt State.

INSTRUCTION COMPLEMENT

The RCA 70/15 Order Code consists of twenty-five instructions that are divided into four classes.

1. DATA HANDLING

The data handling instructions permit 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.

2. ARITHMETIC INSTRUCTIONS

The arithmetic instructions call for the adding and subtracting of fields in either decimal or binary format; in addition, logical instructions perform Boolean operations on the bit structure.

3. DECISION AND CONTROL

The decision and control instructions compare decimal or binary fields, and carry out branching operations to new locations in HSM according to a Condition Code Indicator. In addition, one instruction returns the computer to the Processing State after an interrupt.

4. INPUT/OUTPUT

The input/output instructions control the reading and writing of data between the 70/15 processor and all peripheral equipment on-line. There are also instructions that recognize and recover from error conditions.

INPUT/OUTPUT

The RCA 70/15 communicates with all peripheral devices through six I/O trunks. Each peripheral device has 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.

The Card Punch and Printer are fully buffered so as to allow the physical punching or printing operation to overlap the execution of other instructions. Similarly, a Read Auxiliary instruction overlaps the operation of the Card Readers and Magnetic Tapes with other instructions.

INSTRUCTION FORMAT

There are three basic instruction formats in the 70/15; six-byte, four-byte, and two-byte instructions. The first byte of every instruction is the operation code. Depending on the instruction, the remaining bytes refer to field lengths, 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, in the packed format, two numeric digits. Data is represented in HSM in the Extended Binary-Coded-Decimal Interchange Code (EBCDIC).

HIGH-SPEED MEMORY AND MEMORY ADDRESSING

INTRODUCTION

The 70/15 High-Speed Memory (HSM) unit can store 4096 bytes (or 8192 bytes for two units). A byte is the smallest addressable unit of HSM 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

The 70/15 transfers a byte from HSM to a memory register, and regenerates the byte into memory in two microseconds. This interval is called Memory Cycle Time.

HSM ADDRESSING

Each byte in HSM is addressable, and the address of its location is expressed as a binary number. It requires twelve bits to express the highest memory address of the first block (4095), and thirteen bits to define the maximum HSM address (8191).

Consistent with other machines in the Spectra 70 System, sixteen bits (two bytes) have been allocated within the instruction to define an address. The 70/15 requires only thirteen bits to define the maximum address so the three high-order bits (2^{13}-2^{15}) of the address must be zero.

The following are examples of 70/15 addresses expressed in the binary system, and their decimal equivalents.

The first example shows the binary representation of HSM location 25. To convert to decimal the values of the powers of 2 in the columns carrying a 1 are added together.

Binary	Decimal Equivalent
2^0	1
2^3	8
2^4	16
	25

HEXADECIMAL NUMBERING SYSTEM

The binary system, although efficient for the 70/15, 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.

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.

Binary Address

	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	Decimal Equivalent
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	4094

Decimal	0	1	2	3	4	5	6	7	8
Hexadecimal	0	1	2	3	4	5	6	7	8

9	10	11	12	13	14	15
9	A	B	C	D	E	F

16	17	18	.	.	.	.
10	11	12	.	.	.	.

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:

$4x10^2 + 7x10^1 + 2x10^0$

$4x100 + 7x10 + 2x1 = (472)_{10}$

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

$1x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0$

$32 + 0 + 8 + 4 + 0 + 1 = (45)_{10}$

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

   $1x16^3 + 0x16^2 + 2x16^1 + 4x16^0$

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

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

   $3x16^2 + 10x16^1 + 15x16^0$

   $3x256 + 10x16 + 15x1$

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

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

0001, 0000, 0010, 0100

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

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

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

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

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

Exercises

1. A byte consists of_____information bits and a_____bit, and is the_____addressable unit in the 70/15 HSM.

2. A HSM address consists of_____bits of which the_____ _____ _____must be zero.

3. Convert following hexadecimal numbers to binary:

   a. $A4E8_{(16)}$

   b. $E82C_{(16)}$

   c. $3D71_{(16)}$

4. Convert following hexadecimal numbers to decimal:

   a. $B5F9_{(16)}$

   b. $F93D_{(16)}$

5. Convert following binary numbers to hexadecimal:

   a. $1100011000001010_{(2)}$

   b. $0000101001001110_{(2)}$

   c. $0010110001100000_{(2)}$

6. 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					
Char.	Hex.	Char.	Hex.	Char.	Hex.
A	C1	J	D1		
B	C2	K	D2	S	E2
C	C3	L	D3	T	E3
D	C4	M	D4	U	E4
E	C5	N	D5	V	E5
F	C6	O	D6	W	E6
G	C7	P	D7	X	E7
H	C8	Q	D8	Y	E8
I	C9	R	D9	Z	E9

Numeric	
Char.	Hex.
0	F0
1	F1
2	F2
3	F3
4	F4
5	F5
6	F6
7	F7
8	F8
9	F9

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

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

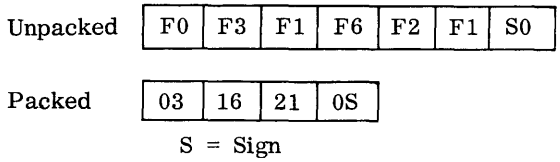
However, the decimal numeric field must be "packed" before it may be used as an operand in a decimal arithmetic operation.

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.

Example

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



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 will be 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 will be considered negative.

After a decimal arithmetic operation the sign of the result will be 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 will generate a zone portion of (1101_2).

EDITED FORMAT

Data that is packed may be unpacked and placed in edited format with one instruction.

Edited format means the inclusion of bytes for characters other than the decimal numeric.

As an example, a packed field, such as the following:

01	01	23	4S

may be converted to a field, such as the following:

-	-	1	,	0	1	2	.	3	4	-

by a single EDIT instruction that both unpacks the field and inserts editing symbols.

MACHINE INSTRUCTION FORMAT

There are three basic instruction formats in the 70/15; a six-byte, a four-byte, and a two-byte instruction.

The first byte of every instruction is the operation code.

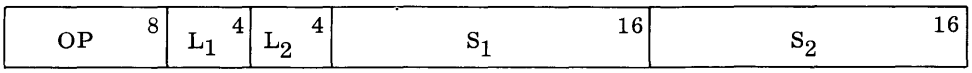
The format of the second byte varies with different types of instructions. In some instructions it is used as a binary length (L) counter of eight bits. In some other instructions, the byte is used to form two length counters (L_1 L_2) of four bits each. And in still some other instructions, it is used to specify a mask (M), or for I/O instructions a Trunk (T) and Unit (U) designation. For some instructions, all or a portion of the second byte is ignored (IGN).

The third and fourth bytes are used for a storage address (S1) in the four-byte and six-byte instructions.

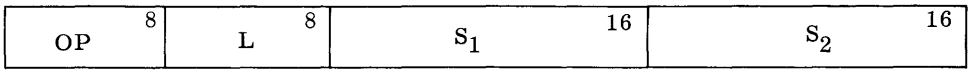
The fifth and sixth bytes are used for a storage address (S2) in the six-byte instructions.

The machine formats and the types of instructions using each format are as shown on the following page.

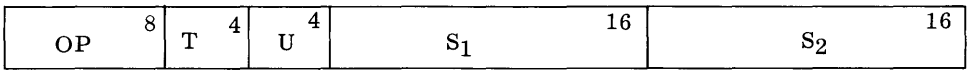
SIX-BYTE INSTRUCTIONS



Binary Arithmetic
Decimal Arithmetic
Decimal Comparison
Packing and Unpacking

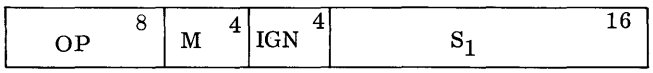


Data Movement
Logical Operations (And, Or, Excl. Or)
Logical Comparison
Data Editing

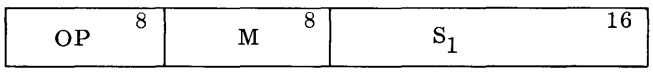


Input/Output

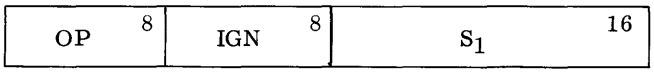
FOUR-BYTE INSTRUCTIONS



Conditional and Unconditional Branch



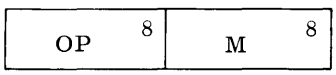
Test Under Mask



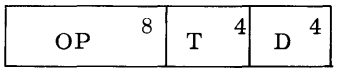
Set P2 Register

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

TWO-BYTE INSTRUCTIONS



Halt



Input/Output (Post Status)

INTERRUPT

INTRODUCTION

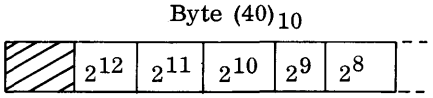
An interrupt facility provides an automatic means for the detection of exceptional conditions, and a method for an immediate program response. The function of sensing for exceptional conditions and the automatic transfer of control to software has been mechanized in the RCA 70/15 hardware. Combining software with hardware interrupt makes it unnecessary to halt the computer when an error develops, and eliminates program sensing of external demands. This type of system allows the user to program a response completely independent of his production processing.

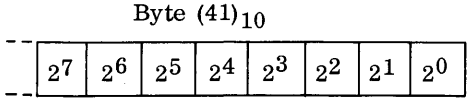
PROGRAMMING STATES

All instructions are executed in one of two states: the Processing State (P1), or 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}).



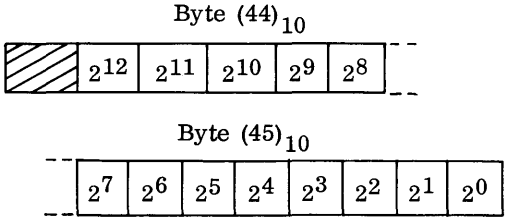


P_1 Counter

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 twenty-five instructions may be executed in the P1 state. The computer remains in this state until an interrupt occurs.

CONTROL STATE

A condition that causes the P1 state to be interrupted causes the computer to execute the instruction whose address is stored in the P2 counter (reserved HSM forty-four $(2C)_{16}$ and forty-five $(2D)_{16}$).



P2 Counter

The interrupt places the computer in the P2 state, and 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 in this state. All twenty-five instructions may be executed in the P2 state, and the computer remains in this state until a STP2 instruction (see page 34) is executed. The STP2 instruction resets the P2 counter to its original value, and returns control to the P1 state. The Interrupt State is not interruptible. Any interrupt attempted will be "pending" until the computer returns to the Processing State, except for the operation code trap which stops all processing.

On returning to the P1 state the computer executes the instruction whose address is stored in the P1 counter. Notice that the first instruction staticized upon returning to the P1 state is that instruction that would have been staticized had interrupt not occurred.

INTERRUPT CONDITIONS

There are two conditions that can interrupt the Processing State.

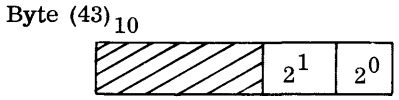
1. I/O Device
2. Operation Code Trap

I/O INTERRUPT

The computer enters the Interrupt State automatically when it receives a byte of data from a communication line, or when someone has pressed the interrupt button on the Interrogating Typewriter.

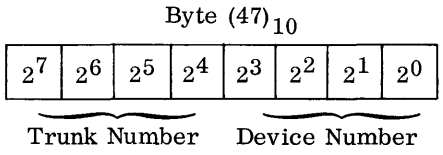
Prior to entering the P2 state, the computer automatically:

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 $(2B)_{16}$



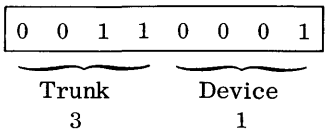
The Condition Code Indicator is then set to 00.

2. 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 43 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 would test the Condition Code (with a Branch on Condition instruction). A setting of $(00)_2$ would indicate that interrupt had been caused by an I/O device. The Trunk and Device Numbers have been stored in the reserved area of HSM, allowing the routine to identify the device that caused the interrupt. For example, if the Interrogating Typewriter is Device one on Trunk three, and the Interrupt button had been depressed, then the contents of HSM location forty-seven would contain:



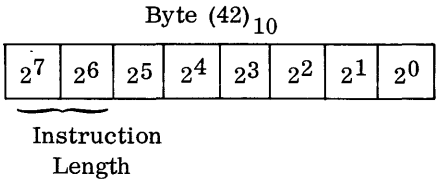
OPERATION CODE TRAP

Staticizing an instruction in which the operation code (the first byte of an instruction) is not one of the twenty-five 70/15 operation codes causes the interrupt called "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

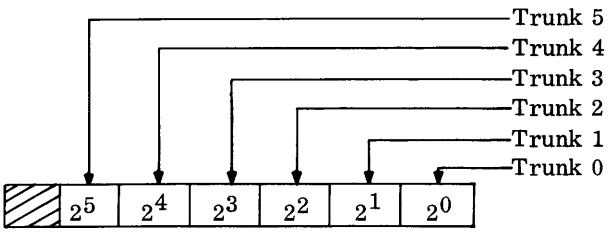
3. Sets the Condition Code to $(01)_2$.

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 be used to simulate an instruction that is not part of the 70/15 order code. For example, the 70/25 operation code $(FC)_{16}$ for Multiply Decimal would cause an interrupt on the 70/15, but the multiply could be simulated by instructions in the P2 state.

INHIBITING I/O DEVICE INTERRUPT

An I/O interrupt from a particular Trunk can be inhibited, but an Operation Code Trap cannot be inhibited.

The reserved HSM location forty-nine $(31)_{16}$ allows the programmer to indicate whether to allow or inhibit the occurrence of an I/O interrupt. The programmer places a mask into location forty-nine of which the rightmost six bits correspond to the six input/output Trunks.



Location 49 $(31)_{16}$

The 2^0 bit corresponds to Trunk Zero, the 2^1 bit corresponds to Trunk one, etc. A bit set to one (1) allows a device on the corresponding Trunk to interrupt, and a zero (0) inhibits interrupt from that Trunk.

A mask of 00010110 allows Trunks one, two, and four to interrupt, and inhibits interrupt from Trunks zero, three and five. If interrupt on an I/O channel is inhibited, that channel will remain "busy" until a Post Status instruction, addressed to the Trunk, is executed (see page 43).

The Flow Chart, p. 14, summarizes the 70/15 interrupt logic.

Exercises

T F 1. The Interrupt State can execute only fifteen of the twenty-five instructions in the 70/15.

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 interruptible.

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.

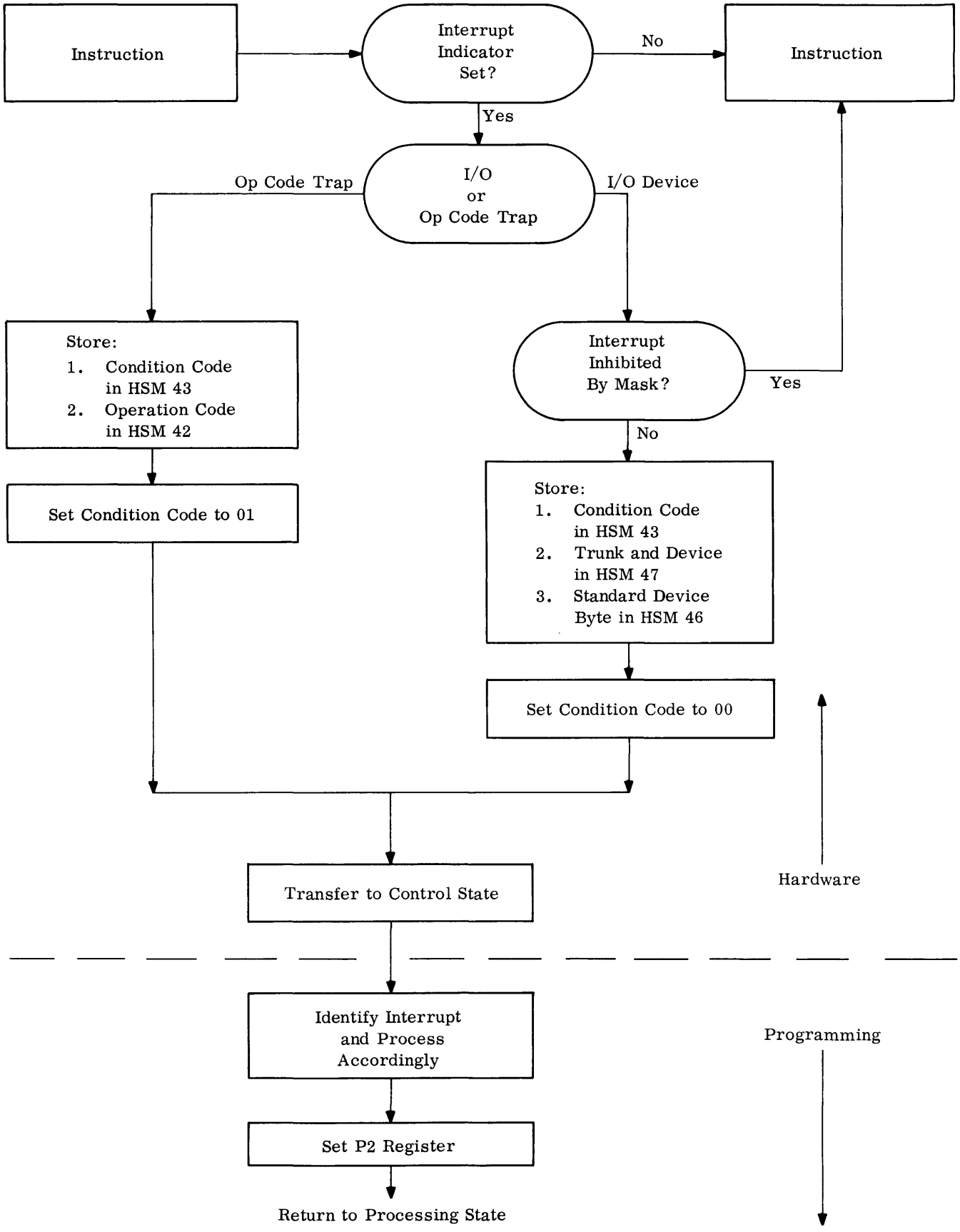
T F 13. Interrupt from any I/O device can be inhibited.

14. Describe the use of HSM location 49.

15. Describe two uses of the Operation Code Trap.

16. List and define all the reserved HSM locations that are used by the interrupt logic.

The Flow Chart below summarizes the 70/15 interrupt logic.



ASSEMBLY LANGUAGE

FORMAT REQUIREMENTS

The RCA 70/15 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 70/15 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. 1-4 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 four 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 a 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 programmer may use the IDENTIFICATION Field (Cols. 73 to 80) for an entry that identifies the program or furnishes a sequence number to each line. The first four characters (columns 73-76) are used by the assembler. When written into the "START" card they are used as the name assigned to the Identification field produced for the object program. The last four characters (columns 77-80) are used as the start of the sequence counter. If they are not numeric, the assembler ignores them and sets its sequence counter to all zeros. Every assembled instruction card has its sequence number derived from the sequence counter set by the START card.

ADDRESSING

SYMBOLIC

Perhaps the most frequently used addressing method is by symbolic names. When a symbol has been used in the NAME field to define a field 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 alphabetic (A-Z) or numeric (0-9) characters with the restrictions of (1) a maximum of four characters and (2) the first character must be alphabetic.

The following are examples of valid and invalid symbols as written in the NAME field:

VALID

A1
STKN
C
IN1

INVALID

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

As an aid to understanding how addresses are assigned to symbolic names, the user should be aware that the Assembler uses a Location Counter to generate an object (or machine language) program from a source (or assembly language) program.

The Location Counter can be considered the same as any other type of internal counter. It may be given an initial value either by the programmer (see ORG code page 17) or by the Assembler.

Then, based on the user's input, a part of the assembly operation is to assign values to each of the user's tags (symbols).

The Location Counter may be advanced at the option of the user or by the Assembler. As an example the Assembler, upon recognizing an instruction code in the OPERATION field, advances the location counter to the next even address.

The example below illustrates the assignment of addresses for symbolic names using the Location Counter (Assume Location Counter set initially to 2000).

Assembly Line	Contents Of Line	Symbol (Tag) Assigned	Address Assigned	For This Line Advance Location Counter To:
First	A 5-byte field	Work	2000	2000
Second	A 2-byte field	ADDR	2005	2005
Third	A 10-byte Constant	WCON	2007	2007
Fourth	A 6-byte Instruction	STRT	2018	2018*
Fifth	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.

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 field defined by the assembly statement.

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

WARE+3

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 respectively of the first byte generated by the statement line.

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

SELF-DEFINING VALUES

In the previous example, a self-defining value of 3 incremented a symbol 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 addressing but should be used with caution as such addresses are not relocatable.

Decimal

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

Examples:

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 four 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 useful for representing binary configurations such as masks.

Example:

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 11000001) is desired.

Example: Three methods generate the same value.

Operand		
C'A',	Character	All will
X'C1'	Hexadecimal	generate
193,	Decimal	$1100\ 0001_2$

ASSEMBLER INSTRUCTIONS

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

In the OPERAND field appears the number of bytes to be reserved followed by the letter C. A symbol appearing in the NAME field is assigned the address of the leftmost byte reserved.

The programmer may also set the Location Counter to any desired value by using the ORG code. This code appearing in the OPERATION field 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 is used, it must have been previously defined in the NAME field.

Example

In this example assume that an input transaction tape contains records with various formats. The maximum size transaction is 80 characters. An input area could be reserved as follows:

Name	Operation	Operand
INAR	DS	80C

The location counter could be reset and areas for transaction formats would be reserved as follows:

Name	Operation	Operand	
	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	
	ORG	INAR	
RACN	DS	10C	
RCOD	DS	2C	Receipt
RDAT	DS	4C	Transaction
RAMT	DS	7C	
	ORG	INAR	
PACN	DS	10C	
PCOD	DS	2C	Payment
PDAT	DS	4C	Transaction
PAMT	DS	7C	
PTYP	DS	2C	
	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 in the program.

CONSTANTS

The DC (Define Constant) code both allocates memory for and stores the value of a constant. The value of the constant is written in the OPERAND field. The value is expressed in one of three forms but they may not be intermixed on any one statement line. The length of each constant is implied by the value appearing 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.

Example

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.

Example

Operation	Operand
DC	X'E020206B20204B202060'

An explanation of the above example of an editing mask is given in DATA EDITING (page 28).

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).

PROGRAM LINKING CODES

There are two codes, ENTRY and EXTRN, that provide communication between two programs each of which were 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

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
RTN	- -	
	- -	
	- -	
	END	

COS is an ENTRY point in the SINE-COSINE Routine

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

ASSEMBLER CONTROL CODES

The first and last statements presented to the Assembler 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

The START code notifies the Assembler to begin assembly of a program. In addition it 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 is written in the OPERAND field which will be used to set the location counter. A symbol appearing in the NAME field will be considered an ENTRY point into the program (see ENTRY code).

END CODE

The END code informs the Assembler that all source input statements have been processed. The OPERAND field specifies the address of the beginning instruction to be executed after the object program has been loaded. A symbol, self-defining value, or asterisked address may appear in the OPERAND field.

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

If no ORG entry, then a segment will be loaded following the previous segment

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 will be loaded (when called) at HSM location 1100_{10}. The third segment would overlay the first two segments at the instruction NAMEd SEG2.

INSTRUCTIONS

INTRODUCTION

The RCA 70/15 Order Code consists of twenty-five instructions that are divided into four classes.

1. DATA HANDLING

   The data handling instructions permit 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.

2. ARITHMETIC INSTRUCTIONS

   The arithmetic instructions call for the adding and subtracting of fields in either decimal or binary format; in addition, logical instructions perform Boolean operations on the bit structure.

3. DECISION AND CONTROL

   The decision and control instructions compare decimal or binary fields, and carry out branching operations to new locations in HSM according to a Condition Code Indicator. In addition, one instruction returns the computer to the Processing State after an interrupt.

4. INPUT/OUTPUT

   The input/output instructions control the reading and writing of data between the 70/15 processor and all peripheral equipment on-line. There are also instructions that 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 INSTRUCTIONS

The Move Character instruction transfers one byte at a time from the sending to the receiving field without any change. The number of bytes transferred is controlled by the L Register. 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	S_1	S_2
$D2_{16}$	7	2200_{10}	2000_{10}

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 contained the value as shown below.

	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 with the effect on the registers after each character is transferred as shown below:

22	00	01	02	03	04	05	06	07	No. of Chars. Transf.	L	A(S_1)	B(S_2)
	x	x	x	x	x	x	x	x	0	07_{16}	2200	2000
	3	x	x	x	x	x	x	x	1	06_{16}	2201	2001
	3	7	x	x	x	x	x	x	2	05_{16}	2202	2002
	3	7	0	x	x	x	x	x	3	04_{16}	2203	2003
	3	7	0	1	x	x	x	x	4	03_{16}	2204	2004
	3	7	0	1	4	x	x	x	5	02_{16}	2205	2005
	3	7	0	1	4	9	x	x	6	01_{16}	2206	2006
	3	7	0	1	4	9	6	x	7	00_{16}	2207	2007
	3	7	0	1	4	9	6	5	8	*FF_{16}	2208	2008

Register Contents (in Decimal)

*Execution Terminating Condition

The Move Character instruction may also be used to move instructions or addresses in the coding portion of a program. An example of this use of the Move Character instruction is furnished in the section on Transfer of Control Instructions.

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 a 120 character area is to be filled with blanks and it is known that a blank $(EO)_{16}$ appears in the first position.

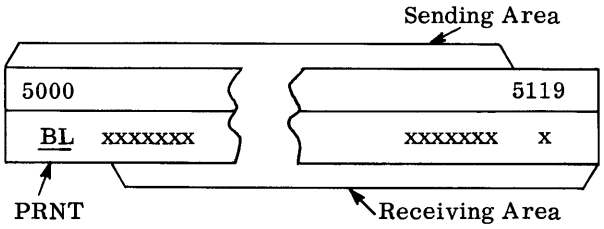
The area is allocated as follows:

Name	Operation	Operand
PRNT	ORG	5000
	DC	X'EO'
	DS	119C

The area PRNT would be allocated HSM location 5000. The area is cleared initially 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.



Because the transfer takes place one character (byte) at a time, upon completion of the execution of the instruction the area 5000 to 5119 will be filled with the Blank $(EO)_{16}$ character.

Exercises: Move Character (MVC) Instruction

For the purpose of this exercise, assume a programmer has allocated memory 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 the allocated areas appear as below, and show the results of each of the instructions on Line 2 below. (Do each question in sequence as each previous question may affect the result of the following question.)

Line 1

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
20	BL	0	1	7	5	J	O	H	N	BL	S	M	I	T	H	0	0	1	4	3	2	2	6	4	3	1
	WORK					NAME										BAL					DATA						

Line 2

WA1

	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
21																										

	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+5(15),WA1+4

PART III

Each record on an input tape contains the following units of information.

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 an output record in the following format.

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

In addition to the instructions 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 at the beginning of your routine.

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 the structure of the individual bytes.

However, 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 the Pack instruction assume that an area must be allocated for input transactions (Unpacked) which are in the following format:

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) 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	S_1	S_2
	$F2_{16}$	4	7	3050_{10}	3010_{10}

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 of 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 is used to form a byte in the unpacked field with a zone portion of F_{16} $(1111)_2$ 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. It is in packed format and assume it contains the following value:

BAL

Packed (Sending) Field	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	S_1	S_2
$F3_{16}$	6	3	5037_{10}	4050_{10}

PBAL

Unpacked (Receiving) Field	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 will be filled with the numeric character zero $F0_{16}$.

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

Exercises

Assume that the following allocations have been made in a program:

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 actual locations contain the values as shown below: (Each location has the contents represented in Hexadecimal digits.)

BAL ↘ ↙ WDAT

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

DATA ↘ ↙ WBAL

21	50	51	52	53	54	55	56	57	58	59	60	61	62
	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 'WDAT'

ANSWERS

	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	S_1	S_2

	Operation	Operand
4.	UNPK	DATA+4(5),DATA(3)

OP	L_1	L_2	S_1	S_2

	Operation	Operand
5.	PACK	DATA+3(1),BAL(5)

OP	L_1	L_2	S_1	S_2

6. 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

Using DS and ORG controlling codes, allocate memory for the input format (as above) to begin at location 3000_{10}. The output record format is the same as above, except that asterisked items are in packed format. Allocate an area for this format beginning at 3100_{10}.

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

The RCA 70/15 has two decimal arithmetic instructions, Add Decimal (AP) and Subtract Decimal (SP).

Both require data fields to be in packed format, and the rightmost byte addressed 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 based on the algebraic result of the addition or subtraction of the two signed operands.

The sign (rightmost four bits of the result operand) will be 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, 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:

	Operation	Operand
Assembly Instruction	AP	BAL(3),AMT(3)

	OP	L_1	L_2	S_1	S_2
Generated Instruction	FA_{16}	2	2	5007_{10}	5150_{10}

then 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)

Whenever 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. Also, 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 will 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)

Generated Instruction

OP	L_1	L_2	S_1	S_2
FA_{16}	2	4	3000_{10}	3020_{10}

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 zero fills and preserves the sign position:

Assembly Instruction

Operation	Operand
SP	WBAL(4),WBAL(4)

Generated Instruction

OP	L_1	L_2	S_1	S_1
FB_{16}	3	3	6143_{10}	6143_{10}

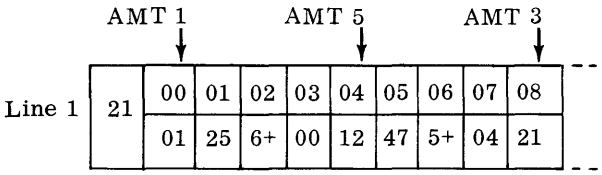
Result Field

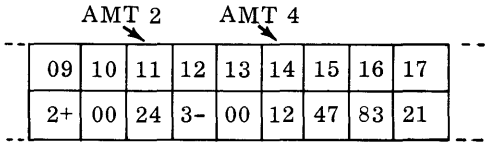
WBAL

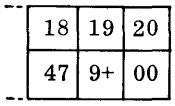
61	43	44	45	46
	00	00	00	0+

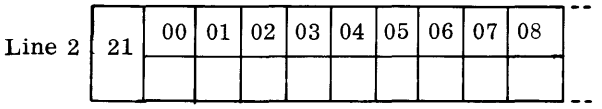
Exercises

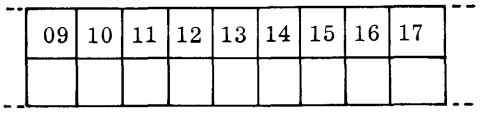
Assume HSM has been allocated as indicated on Line 1 and that each location contains the values as shown.

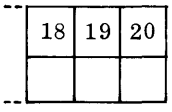












For each of the assembly instructions listed below, show the result of the instruction on Line 2 above and the Condition Code as set following execution of the instruction. (Consider each question independently based on the contents of Line 1.)

	Operation	Operand
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)

Condition Code

	0	1	2	3
1.				
2.				
3.				
4.				
5.				

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.

Another option is that data may be edited as it is moved.

Editing is very much like unpacking data with the advantage, however, of having two added functions being performed as the data is unpacked. The editing instruction allows the programmer to (1) suppress leading zeros to a predetermined location in the edited field and (2) to 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 used as 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 will be inserted unless it is a leading insignificant zero and a Significance Start character has not been previously encountered.

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 to begin for 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+

HSM before Execution

MASK

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

Assembly Instruction

Operation	Operand
ED	MASK(9),AMT

Generated Instruction

OP	L	S_1	S_2
DE_{16}	8	2900_{10}	2000_{10}

HSM after Execution

	00	01	02	03	04	05	06	07	08
29	-	-	-	-	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 use of 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. It 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 will reset the trigger to zero.

The condition code is set by the Edit Instruction. It is set to zero if the packed field has a zero value. It is set 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 also a comma if the value were 1,000.00 or higher in value 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	S_1	S_2
	DE_{16}	10	2900_{10}	2000_{10}

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	20	00 01 02 03
		00 00 00 0+

MASK

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

Assembly Instruction	Operation	Operand
	ED	MASK (11), AMT

MASK

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

0 → 1 → 0

Condition Code = 0

Examples 3 and 4

(Same Mask - Result after positive and negative field.)

Example 3

HSM before and after Execution	20	00 01 02
		01 23 4+

Example 4

	20	00 01 02
		04 27 5⊖

MASK

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

Assembly Instruction	Operation	Operand
	ED	MASK (10), AMT

Example 3

HSM after Execution	21	00 01 02 03 04 05 06 07 08 09
		- - 1 2 . 3 4 - - -

Trigger → 0 → 1 → 0 →

Condition Code = 2

Example 4

HSM after Execution	21	00 01 02 03 04 05 06 07 08 09
		- - 4 2 . 7 5 ⊖ C R

0 → 1 →

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

20	00 01 02 03 04 05
	01 23 7+ 00 29 5-

MASK

HSM before Execution

21	00
	- d d S . d d C R f f f -

21
d d S . d d C R f

Assembly Instruction

Operation	Operand
ED	MASK(22),AMTS

MASK

HSM after Execution

21	00
	- - 1 2 . 3 7 - - - - - -

0 → 1 → 0 →

21
- - 2 . 9 5 C R -

Significance Trigger → → 1 → 0

Condition Code = 1 (Based on last field processed)

The field separator character resets the significance trigger to zero, so that unwanted characters are properly suppressed in the next field.

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

Exercises

	VAL			ACC			BOH				
21	00	01	02	03	04	05	06	07	08	09	10
	01	24	7+	00	00	0+	00	00	47	21	5⊖

DEST

11	12	13	14	15
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
- ⊖ = Minus
- d = Digit Select
- * = Asterisk
- s = Significance Start
- f = Field Separator
- ,. = Insertion Characters

COLUMN I

1.

Operation	Operand
ED	MASK(8),VAL

COLUMN II

MASK

	00	01	02	03	04	05	06	07
22	-	d	d	S	.	d	d	⊖

2.

Operation	Operand
ED	MASK(8),ACC

MASK

	00	01	02	03	04	05	06	07
22	-	S	d	d	.	d	d	⊖

3.

Operation	Operand
ED	TDTL (14), BOH

TOTL

	50	51	52	53	54	55	56	57
22	*	d	,	d	d	S	,	d

58	59	60	61	62	63
d	d	.	d	d	⊖

4.

Operation	Operand
ED	OVL (15), DEST

OVL

	70	71	72	73	74	75	76	77
22	-	d	.	d	d	⊖	f	-

78	79	80	81	82	83	84
d	d	S	.	d	d	⊖

5. The 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

6. Prepare assembler statements for allocating storage memory and constants. Routine coding will 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 two 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

HSM before and after Execution

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	S_1	S_2
$D5_{16}$	7	2700_{10}	2800_{10}

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), ADR 2

Generated Instruction

$D5_{16}$	1	2000_{10}	2002_{10}

Condition Code = 1 (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 will not enter into the comparison. If the second operand is shorter in length it will be 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 will, in this case, also be based on the relative algebraic values of the operands.

Example

HSM Before and After Execution

AMT

	00	01	02	03
50	01	39	64	2-

VAL

	20	21	22
51	02	34	5+

Instruction

Operation	Operand
CP	AMT(4), VAL(3)

Generated Instruction

OP	L_1	L_2	S_1	S_2
$F9_{16}$	3	2	5000_{10}	5120_{10}

Condition Code = 1 (First Operand Low)

Example

HSM Before and After Execution

CHK

	02	03	04
40	12	39	4+

BAL

	20	21	22	23
41	09	12	39	4+

Instruction

Operation	Operand
CP	CHK(3), BAL(4)

Generated Instruction

OP	L_1	L_2	S_1	S_2
FG_{16}	2	3	4002_{10}	4120_{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 will generate 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 is used following 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 will take 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

SET P2 REGISTER (STP2) INSTRUCTION

This instruction transfers the computer 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

00	43	
	03_{16}	

00	40	41
	2300_{10}	

P2 Counter

00	44	45
	4000_{10}	

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

The following instruction will transfer the computer to the P1 state, and store 3800_{10} in the P2 counter.

Assembly Instruction

Operation	Operand
STP 2	ENTR

Generated Instruction

OP	M	S_1
82_{16}	00	3800_{10}

HSM After Execution

P1 Counter

00	40	41
	2300_{10}	

P2 Counter

00	44	45
	3800_{10}	

Transfer of Control to 2300_{10}

Condition Code Reset to 3

Exercise

	BAL			VAL			AMT		
24	50	51	52	53	54	55	56	57	58
	00	12	4+	12	74	5⊖	98	21	2+

	NUM				COST			UNIT		
	59	60	61	62	63	64	65	66	67	68
	98	42	17	0+	00	00	1+	00	12	4⊖

Based on the packed format and symbolic names as shown above, show the Condition Code that will be set following execution of the instruction.

Compare Decimal — Cond. Code

	Operation	Operand	
1.	CP	BAL (3), UNIT + 1 (2)	_____
2.	CP	VAL (3), COST (3)	_____
3.	CP	AMT (3), NUM (4)	_____

	DEF				GHI		MNO			
25	00	01	02	03	04	05	06	07	08	09
	F0	F1	F3	F7	AA	AA	AB	CD	EF	01

	PQR				JKL		ABC		
	10	11	12	13	14	15	16	17	18
	24	57	AB	CD	EF	AA	AO	FO	F1

	19	20
	F4	F7

Based on the above, show the Condition Code that will be set following execution of the instruction. (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		THRE			FOUR	
24	50	51	52	53	54	55	56	57	58	59
	00	12	4+	12	74	5⊖	98	21	2+	98

	FIVE			SIX					
	60	61	62	63	64	65	66	67	68
	42	17	0+	00	00	1+	00	12	4⊖

Based on the above, indicate which instruction will be executed next in the space provided:

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 _____

BINARY ARITHMETIC

The two binary arithmetic instructions, Add Binary and Subtract Binary, operate on an integral number of bytes as controlled by the length of the first operand.

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

Both instructions operate from right to left in performing the binary arithmetic operation. 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 is used to increment the second address of the instruction which moves a record to the processing area.

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

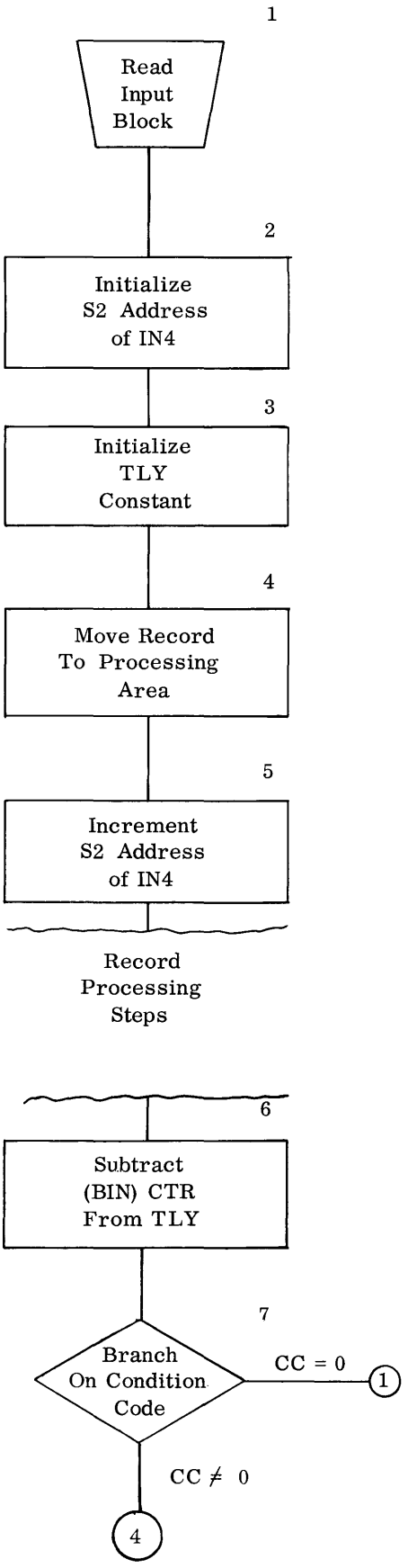
The Input Block, Record Processing, and constant areas can be allocated as follows:

Allocation of Input and Record Processing Area

Name	Operation	Operand	HSM Allocation	Stored Value of Constants (Hex. Format)
	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		
RDIN	DC	A(INP)	3480-3481	OB B8
INCR	DC	A(80)	3482-3483	00 50
TLY	DC	X'0505'	3484-3485	05 05
CTR	DC	X'01'	3486	



Name	Operation	Operand
IN2	MVC	IN4 + 4 (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 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

The preceding functional chart shows the matching coding steps. Only the steps pertinent to the use of the Binary Arithmetic instructions are shown.

Example

ADD BINARY OF PREVIOUS INSTRUCTION BINARY VALUE

INCR

HSM Before and After Execution

34	82	83
	00	50

0000 0000 0101 0000

IN4+4

HSM Before Execution

40	16	17
	OB	B8

0000 1011 1011 1000

Assembly Instruction: AB IN4+4 (2), INCR (2)

Generated Instruction

OP	L_1	L_2	S_1	S_2
F6	1	1	4016_{10}	3482_{10}

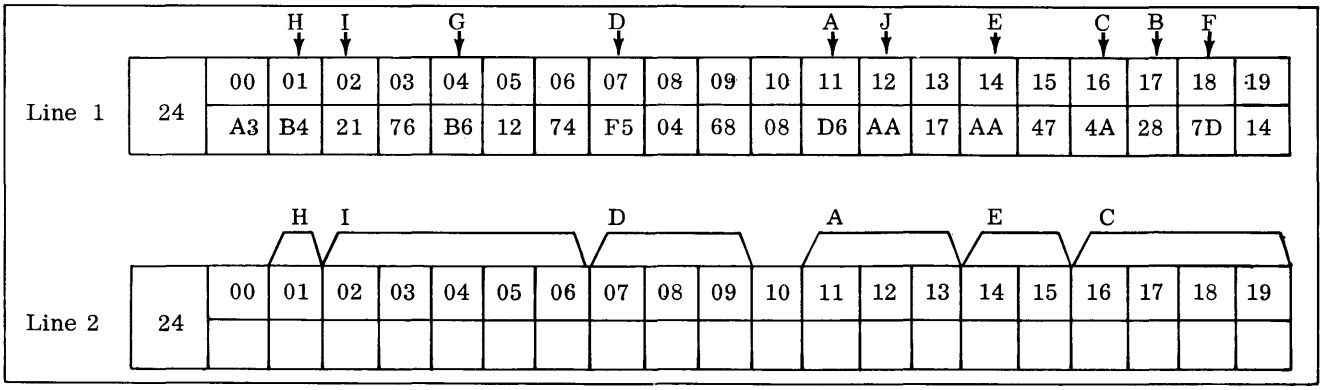
HSM After Execution

40	16	17
	OC	08

0000 1100 0000 1000

Condition Code = 2

Exercise



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

	Column I		Column II
	Add Binary		Condition Code
	Operation	Operand	
1.	AB	A(3), B(3)	_____
2.	AB	C(4), D(4)	_____
3.	AB	E(2), F(2)	_____
	Subtract Binary		
	Operation	Operand	
4.	SB	D(3), G(3)	_____
5.	SB	H(1), I(1)	_____
6.	SB	I(5), J(2)	_____

LOGICAL INSTRUCTIONS

The Logical Instructions perform operations on the individual bits of a byte. The operation works from 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 in 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 / 00 9A	0000 0000 1001 1010

INC

HSM Before and After Execution	31 | 00 01 / FF FA	1111 1111 1111 1010

Assembly Instruction	Operation	Operand
	NC	AD1 (2), INC

	OP	L	S_1	S_2
Generated Instruction	$D4_{16}$	1	3000_{10}	3100_{10}

AD1

HSM After Execution	30 | 00 01 / 00 9A	0000 0000 1001 1010

Condition Code = 1

OR INSTRUCTION

This instruction inserts 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 will product 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 / 00 00 01	0000 0000 0000 0000 / 0000 0001

BAL

HSM Before Execution	40 | 80 81 82 / 07 89 8C	0000 0111 1000 1001 / 1000 1100

Assembly Instruction	Operation	Operand
	OC	BAL (3), CH

	OP	L	S_1	S_2
Generated Instruction	$D6_{16}$	2	4080_{10}	3700_{10}

BAL

HSM After Execution	40 | 80 81 82 / 07 89 8D	0000 0111 1000 1001 / 1000 1101

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. This is accomplished by a modifying mask with one bit in the designated bit positions where the function is desired.

The rule of Exclusive Or may be considered the same as binary addition without a carry being generated, or as follows:

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

Example

INH

		Bit Configurations
HSM Before and After Execution	31 | 10 / 3F	0011 1111
HSM Before Execution	00 | 49 / 3F	0011 1111

Assembly Instruction	Operation	Operand
	XC	X'0031'(1), INH

	OP	L	S_1	S_2
Generated Instruction	$D7_{16}$	0	0049	3110

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 this condition is desired based on the data, a Logical instruction may be used which inserts 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 which inserts all zero bits in the mask may be used. This makes the BC a 'no-Op' instruction.

Logical instructions are used to alter the value of a field. A logical instruction may be used to change the sign of a packed field from a plus sign $(1100)_2$ to a minus sign $(1101)_2$. This 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 will test 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 will be one if the selected bits are a mixture of zeros and ones. Condition Code three will be set if the selected bits are all ones. Condition Code two is not set by this instruction.

Example #1

HSM Before and After Execution

	LOC
60	10
	$0101\ 1100_2$

Assembly Instruction

Operation	Operand
TM	LOC, X'OF'

Generated Instruction

OP	M	S_1
91_{16}	$0000\ 1111_2$	6010_{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	S_1
91_{16}	$0000\ 0010_2$	0049_{10}

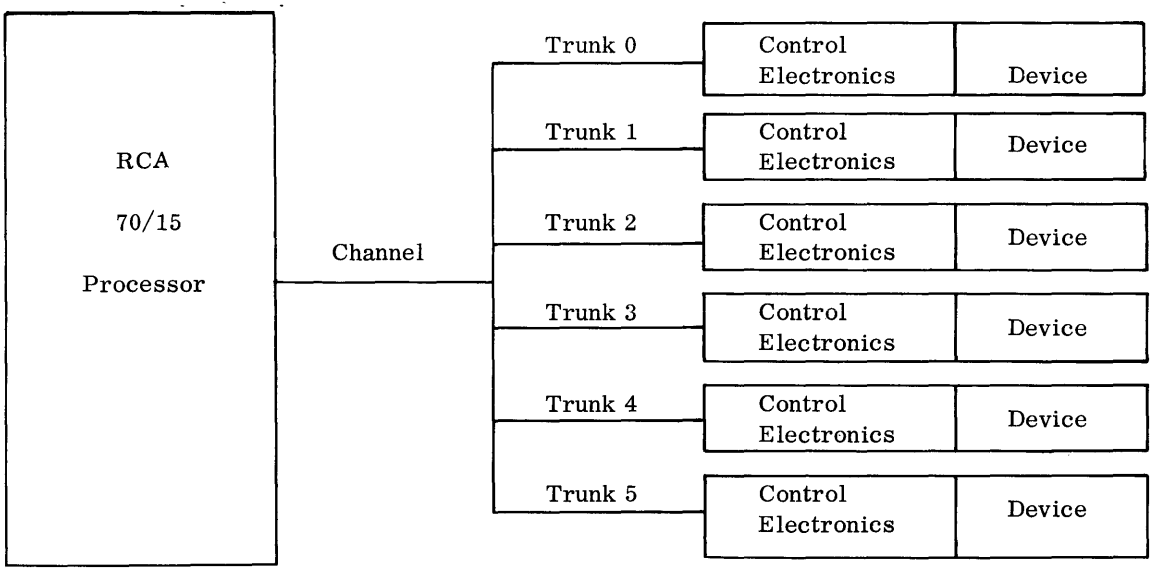
Condition Code = 0

INPUT/OUTPUT

INTRODUCTION

The Programmer may elect to control Input and Output through the use of assembly I/O commands, or through the use of I/O subroutines which are part of the 70/15 software package. This section describes and illustrates the use of the eight assembly input/output instructions, and the methods of error recognition and recovery.

The RCA 70/15 communicates with all peripheral devices through the RCA Spectra 70 standard interface unit. Each peripheral device contains its own Control Electronics which communicates between the device and the standard interface unit. The Control Electronics also has the ability to transmit to the processor the status of the device, and any error conditions generated by an I/O command.



The assumed configuration indicated below will be referenced by all the examples in this section.

Trunk	Unit	Device
1	1	Card Reader
2	1	Card Punch
3	1	Printer
4	1,2,3,4	Tape Stations

READING DATA

There are three Read Commands to bring data into High-Speed Memory (HSM). The Read Device Forward and Read Auxiliary commands can select all peripheral equipment on line to the 70/15. The Read Device Reverse command can be issued only to tape stations.

The Read instructions:

Operation	Operand
RDF	T(D), S_1, S_2
RDR	T(D), S_1, S_2
RDA	T(D), S_1, S_2

select a Trunk and Device, and indicate the HSM area 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).

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 (HSM 1078), etc., and the last byte read will be placed in INPT (HSM 1000). Notice that the S_1 and S_2 addresses had to be reversed.

The Read Device Forward and Read Device Reverse instructions must complete execution before the next instruction in sequence can be staticized. The Read Auxiliary (RDA) instruction uses auxiliary registers to provide overlapping operations. That is, the computer can staticize and execute another instruction at the same time that the Read Auxiliary instruction is being executed. The format of the Read Auxiliary instruction is the same as the RDF instruction except that the S_2 operand is ignored. The RDA instruction will be terminated by either reaching a gap on magnetic tape or by reading one card. The RDA instruction:

Operation	Operand
RDA	4(2), INPT, INPT+99
MVC	AREA(12), TEMP

reads 100 bytes from magnetic tape 2 into HSM 1000-1099. The next instruction in sequence (MVC) will be staticized and executed while the data is being physically read from magnetic tape 2.

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.

The RDA instruction is terminated only by the first condition (above).

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 1200-1319 (OUTP area) to the Printer. The same command with the Trunk number changed:

Operation	Operand
WR	4(1), OUTP, OUTP+79

writes a block of eighty bytes to magnetic tape 1 on Trunk 4, or:

Operation	Operand
WR	2(1), OUTP, OUTP+79

punches a card.

The Erase (WRE) instruction can be considered an output command except that it can only write blanks to magnetic tape. The instruction:

Operation	Operand
WRE	4(2), OUTP, OUTP+50

erases an area of tape that is equal to 51 bytes in length.

CONTROLLING PERIPHERAL DEVICES

The I/O instructions covered so far have as their function the moving of data between HSM and the selected devices. The Write Control (WRC) instruction has the function of communicating control information (rewind tape, paper advance, pocket select, etc.) to the selected device. The WRC instruction transmits a byte from HSM to the Control Electronics of the selected device. The configuration (bits) of the control byte is defined for each of the peripheral devices as indicated below.

CARD READER CONTROL BYTE

2^0 Select Output Stacker #1

2^1 Select Output Stacker #2

2^2 Translate mode

2^3 Binary

2^4-2^7 NOT USED

MAGNETIC TAPE CONTROL BYTE

2^0 Not Used

2^1 Not Used

2^2 Re-Read

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 = COUNT — Advance the paper up to fifteen single spaces, or selection of Paper Loop Channel (1-11).

2^4-2^5 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.
- 1 = Paper advance by Paper Tape Loop as selected by the 2^0-2^2 count.

Examples

Assuming the following bytes in memory:

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

The instructions:

	Operation	Operand
1.	WRC	1(1), CTL1, CTL1,
2.	WRC	4(3), CTL2, CTL2
3.	WRC	3(1), CTL3, CTL3

will:

1. Select Output Stacker #1 on the Card Reader.
2. Rewind one gap on Magnetic Tape 3.
3. Paper advance the printer two lines immediately.

ERROR RECOGNITION

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

If the selected device is busy, the instruction is restaticized until the device is available. All I/O instructions set the condition code to one of three conditions. If the device is inoperable, the instruction is terminated and sets the Condition Code to one (1). If the instruction was executed (with or without errors) the Condition Code is set to zero (0) when the instruction terminates. If an interrupt is pending on a device, and an I/O instruction selects that device, then the instruction terminates and the Condition Code is set to 2.

For example, if the interrupt button was depressed on the Interrogating Typewriter, but interrupt had been inhibited on that Trunk, then an I/O command to the Interrogating Typewriter would not be executed.

The flow chart (shown on the following page) indicates both hardware and programming logic for basic I/O.

STATUS INFORMATION

An indication that an error occurred is made available by directing the Control Electronics of a particular device to transmit the Standard Device Byte into the reserved area of memory for that Trunk. The Standard Device Byte is standard for all peripheral equipment in the RCA Spectra 70 System.

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 the condition is cleared, i.e., no power, jammed, interlock open, etc.

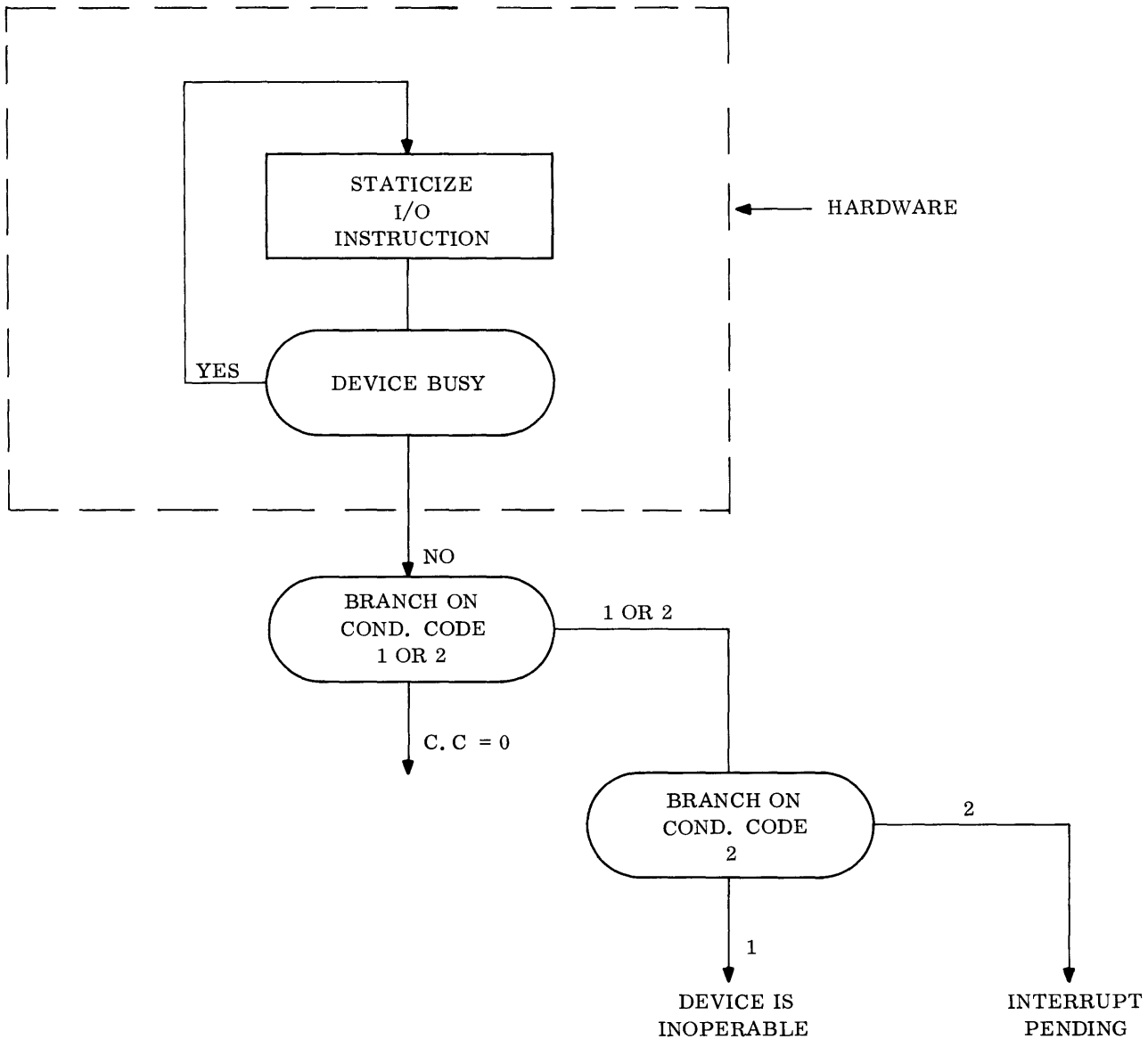
2^2 SECONDARY INDICATOR SET - Indicates that a bit in the "SENSE BYTE" is set. A "I/O SENSE" operation must be executed to determine the particular condition.

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

2^4 CONTROL BUSY - Same as DEVICE BUSY on 70/15.

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

2^6 TERMINATION INTERRUPT - Not used on 70/15.



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

Six of the eight bits convey information that is not applicable to the 70/15. The Manual Interrupt Pending (2^7) bit indicates the same information as the Condition Code being set to 2 after staticizing an I/O instruction. This bit may be tested to determine if the interrupt button on the Interrogating Typewriter had been depressed (and the interrupt had been inhibited) prior to issuing a command to the Typewriter.

The "Secondary Indicator Set" bit (2^2) is the primary indicator of the Standard Device Byte. This bit set to one (1) means that an error has occurred and that control information is available. The Programmer must issue an I/O Sense instruction (see page 45) to determine the exact condition that caused this bit to be set.

The Post Status (PS) instruction directs the selected device to transmit the Standard Device Byte to the reserved area of memory for that Trunk.

The instruction:

Operation	Operand
PS	2 (1)

causes the Standard Device Byte for the Card Punch to be placed in the reserved HSM location 10.

I/O SENSE INFORMATION

The Input/Output Sense instruction:

Operation	Operand
IOS	T(D), S_1, S_2

causes the selected device to transmit one byte to the HSM location referenced by S_1, S_2. The Sense Byte is different for each type of device. Each bit of the Sense Byte indicates a particular error or control condition. The programmer can test the bits to determine the condition indicated.

The chart (shown below) summarizes the meaning of each bit of the Sense Byte for the Card Reader. Card Punch, Magnetic Tapes, Interrogating Typewriter, and Printer.

Exercise

T F 1. All I/O operations are serial in the 70/15.

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

T F 3. On the 70/15 one to six I/O channels may be 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".

Typical 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	Service Request Not Honored	Not Used
2^7	Read Error	Punch Error	Read or Read After Write Error	Not Used	Not Used

Sample Coding

The series of instructions in the sample coding does not represent a complete program. It is included to illustrate the use of the Read, Post Status, I/O Sense, and the necessary masks required to do some input operations in 70/15 assembly language.

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

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

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

9. What are the formats of the 70/15 I/O instructions?

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

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

12. Write the necessary instructions to:

    a. Read a Card

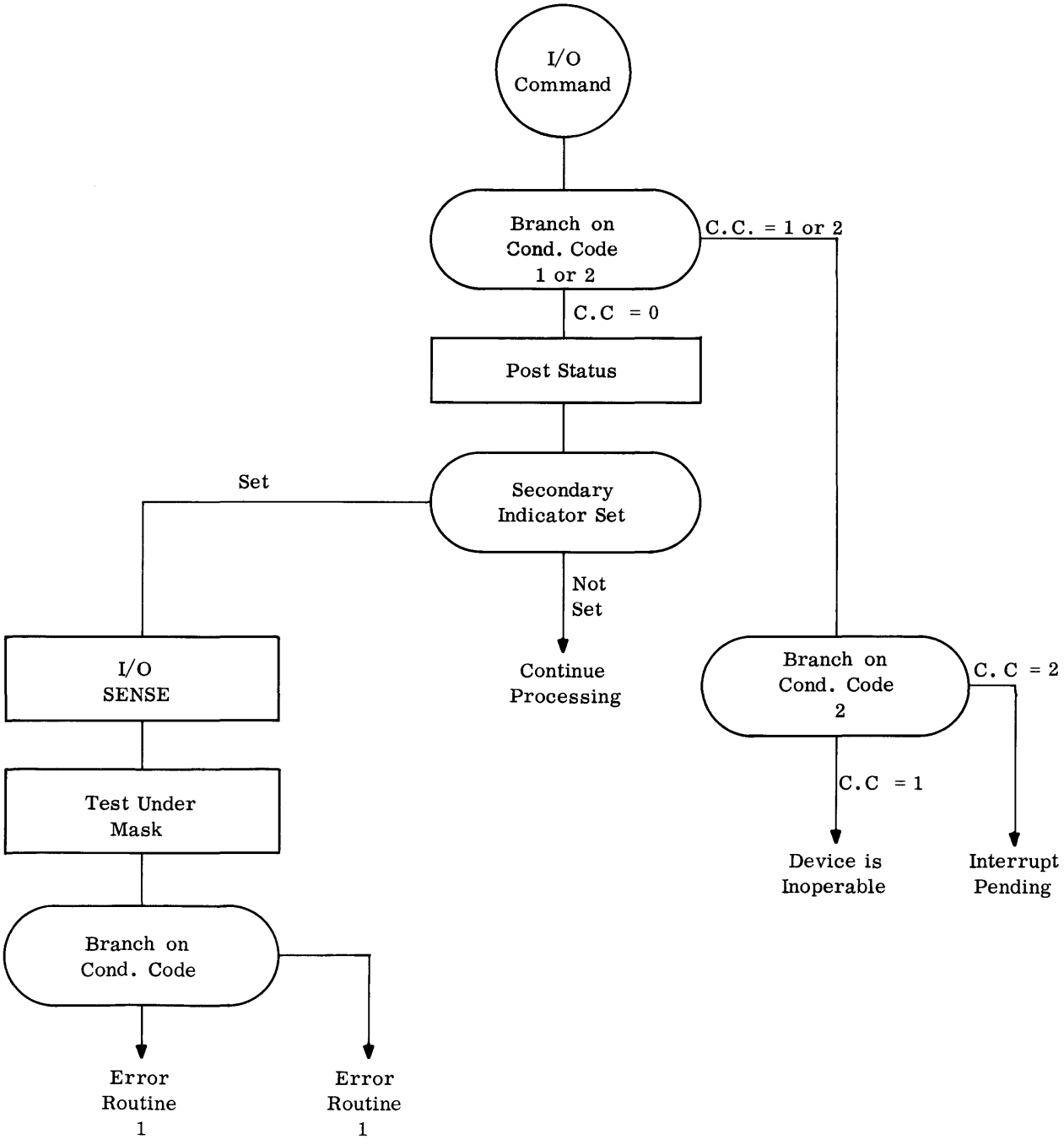
    b. Punch a Card

    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.

The flow chart (below) indicates the programming logic for Input/Output control:



SAMPLE CODING

NAME	OPERATION	OPERAND	COMMENTS
READ	RDF	4(1),INPT,INPT+99	READ DATA
	BC	X'6',NCDO	BRANCH IF INOPERABLE
	PS	4(1)	POST STATUS
	TM	18,MSKI	TEST IF SEC. INDICATOR SET
	BC	X'1',ERR	BRANCH IF 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 SENSE BYTE
	TM	SENS,MSK2	TRANSMISSION OR READ ERROR
	BC	X'1',ERR1	BRANCH IF ERROR
	B	READ	BRANCH TO READ
ERR1	H		ERROR
SENS	DS	1C	
INPT	DS	100C	
MSK1	DC	X'04'	
MSK2	DC	X'90'	