HEWLETT hp PACKARD

ASSEMBLER / BCS

TRAINING MANUAL

ASSEMBLER / BCS

TRAINING MANUAL

HEWLETT hp PACKARD

11000 Wolfe Road
Cupertino, California
95014

HP 02116-9073

April 1970

First Edition: December 1967
Revised: April 1970

Printed in the U.S.A.

Second Edition

PREFACE

This training manual is an introduction to programming for the HP 2116 computer, but the information also applies to the 2115 and 2114 computers. The book focuses only on information pertinent to programming concepts; specific operating procedures may be found in the manuals listed below.

The training manual describes number systems, general computer hardware characteristics as well as the specific characteristics of the HP computers. The Assembler and Basic Control System are explained; flowcharting and program coding are included in sections on problem analysis and instruction formats. Explanation for coding machine instructions, assembler pseudo operations and input/output requests is given.

Other computer publications provided by Hewlett-Packard include:

ALGOL Programmer's Reference Manual
Assembler Programmer's Reference Manual
Basic Control System Programmer's Reference Manual
FORTRAN Programmer's Reference Manual
Specifications and Basic Operation Manual
Standard Software Systems Operating Manual
Symbolic Editor Programmer's Reference Manual

NEW AND CHANGED INFORMATION

All known errors in this manual have been corrected. Changes in the text are marked by a vertical line in the margin.

CONTENTS

INTRODUCTION

The term computer usually calls to mind a huge box with switches, dials, and blinking lights: an engineering marvel, the electronic "brain". The computer, however, is virtually useless without two other vital elements: some method of translation between the computer and its users, and a person capable of stating the logical processes the computer must perform to solve his problem.

These three elements of computing correspond to the terms hardware, software, and programmer.

HARDWARE

Computer hardware consists of four general elements:

1. Control -- directs transfer of data and controls operations performed.

2. Arithmetic element --element in which computations are performed.

3. Memory -- place where information to be processed is stored.

4. Input/Output -- allows information to be transferred between the computer memory and external devices such as paper tape readers and punches, printers, and so forth.

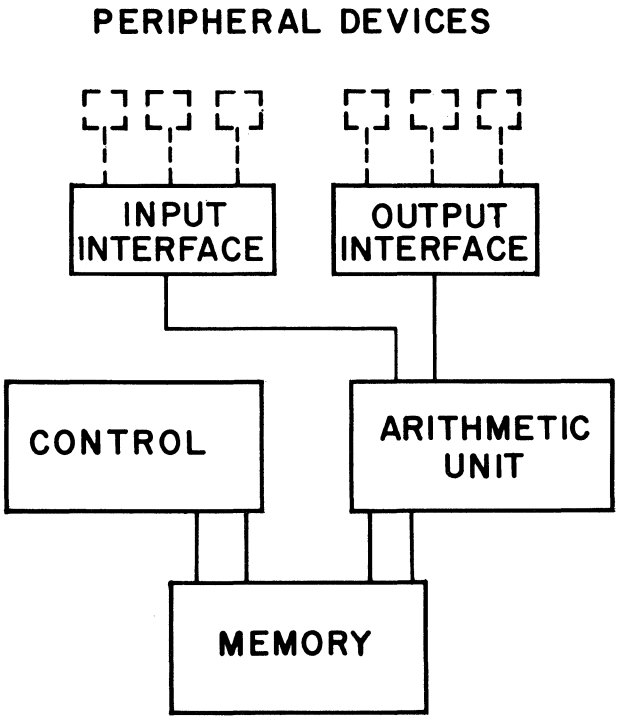
A greatly simplified illustration of the flow of information between these elements is given on the following page.

SOFTWARE

Computer hardware recognizes information as patterns of "off/on" current pulses, or bits. Since it would be difficult for a person to easily express himself in the "off/on" language which is understood by the computer, a method of translating was devised.

The term "software" originated to differentiate between hardware, a physical device, and translators and control systems, which are a sequence of computer instructions. Software translators and control systems are usually stored

on a medium such as paper tape or magnetic tape, from which they are input to the computer and executed.



Translators accept and translate readily understandable instructions into machine language. Translators are composed of two general categories:

Assemblers allow expression of instructions as abbreviated mnemonic codes. In general, one code is translated into one machine instruction. However, most assemblers contain pseudo instructions and subroutine calls, which generate a number of machine instructions to perform a specific task.

Compilers allow expression in a language more nearly resembling words and/or formulas. One instruction statement may generate many machine instructions. Compilers are usually designed with a certain type of problem in mind. Thus, assemblers are machine-oriented languages; compilers are problem-oriented languages. For example, FORTRAN, or FORmula TRANslation, allows easy expression of complex

mathematical formulas for scientific use. ALGOL, or ALGOrithmic Language, provides a concise language for expressing a large class of numerical processes.

Another type of software is the control system (also called monitor system or operating system). A control system provides functions useful to translated programs produced by assembler and compiler systems, for example, program loading and error detection aids.

PROGRAMMING

Programming, then, consists of:

(1) Analyzing a problem and determining the process necessary to obtain a solution.

(2) Coding the solution process in the software language which is most applicable.

NUMBER SYSTEMS 1

The "off/on" computer language can be related to a "zero/one" number system called the binary number system.

1.1 DEFINITION OF NUMBER SYSTEMS

Number systems are characterized by:

(1) radix, or base; the number of unique symbols used in the system. In the decimal number system, the base is ten, corresponding to the ten unique symbols 0 through 9. In the binary number system, the base is two, corresponding to the two unique symbols 0 and 1.

(2) modulus; the number of unique quantities or magnitudes a system can distinquish. The modulus of the binary and decimal number systems is infinite; any quantity can be expressed with either of these systems. However, a machine with a physical limit to the number of digits it can hold has a modulus. For example, a decimal adding machine with ten digits, or counting wheels, would have a modulus of 10^{10}, or 10,000,000,000. (0 to 9,999,999,999) A binary computer which can hold a unit of 12 binary digits (or bits) has a modulus of 2^{12}, or 4096 (in decimal). (The formula for the modulus of a number system is b^n, where b = base, n = number of digit positions available).

1.1.1 DECIMAL NUMBER SYSTEM

The value which a digit assumes in a number system is dependent upon its position. For example, in the decimal number system:

					decimal point	
...thousands	hundreds	tens	units	↑	tenths	hundredths...
(10^3)	(10^2)	(10^1)	(10^0)		(10^{-1})	(10^{-2})

Positions to the left of the decimal point increase in value in ascending powers of ten, beginning with zero. Positions to the right of the decimal point decrease in value in ascending negative powers of ten, beginning with -1.

Thus, the number 32,768.9 represents:

$$3 \times 10^4 + 2 \times 10^3 + 7 \times 10^2 + 6 \times 10^1 + 8 \times 10^0 + 9 \times 10^{-1}$$

1.1.2 BINARY NUMBER SYSTEM

Similarly, in the binary number system, positions relate to ascending positive and negative powers of two:

						binary point			
Value (in decimal)	32	16	8	4	2	1	1/2	1/4	
. . .	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	. . .

Thus, the binary number 1011.01 represents:

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}. \text{ or}$$

$$8 + 2 + 1 + 1/4 = 11\ 1/4 \text{ (in decimal)}$$

Observe that this number, in binary, represents quite a different magnitude than the same number in decimal. To distinguish the number system in which a quantity is being expressed, the base is appended as a subscript at the end of a number.

10010_{10} (decimal number system)

10010_2 (binary number system)

1.1.3 OCTAL NUMBER SYSTEM

Another number system useful in computer terminology is the octal number system, with a base of 8. The octal number system is useful in that the unique symbols, 0 through 7, correspond to all the quantities expressable in a 3-digit group in the binary number system.

octal	binary
0	000 $(0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 0)$
1	001 $(0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1)$
2	010 $(0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2)$
3	011 $(0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 3)$
4	100 $(1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 4)$
5	101 $(1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5)$
6	110 $(1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 6)$
7	111 $(1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 7)$

Thus, the octal number system can be used as a convenient "shorthand" method of expressing binary numbers.

1.2 NUMBER SYSTEM CONVERSION

Octal to Binary and Binary to Octal

The simplest conversion is the binary to octal or octal to binary conversion, because of the correspondence of one octal digit to a triplet of binary numbers. The triplets must be measured from the binary point.

Examples:

$$\underbrace{101}_{5}\ \underbrace{110}_{6}.\ \underbrace{001}_{1}{}_{2}$$

$$\underbrace{1}_{1}\ \underbrace{000}_{0}\ \underbrace{011}_{3}.\ \underbrace{01}_{2}{}_{2}$$

(zero's are implied at each end to fill the triplet.)

$$\underbrace{7}_{111}\ \underbrace{6}_{110}\ \underbrace{5}_{101}\ \underbrace{3}_{011}\ \underbrace{4}_{100}.\ \underbrace{2}_{010}\ \underbrace{0}_{000}\ \underbrace{1}_{001}{}_{2}$$

Binary/Octal to Decimal

In previous sections of this chapter, we have used a process for converting binary numbers to decimal. The process may be stated more generally as follows:

The number $a_n\ a_{n-1}. . .a_2a_1a_0.\ a_{-1}a_{-2}. . .a_{-m}$ where the a's represent the digits and the subscripts represent the position of the digit from the binary or octal point, may be converted by the following formula:

$$a_n \times b^n + a_n \times b^{n-1} + \ldots + a_2 \times b^2 + a_1 \times b^1$$
$$+ a_0 \times b^0 + a_{-1} \times b^{-1} + \ldots a_{-m} \times b^{-m}$$

where b = base from which conversion is being made.

Examples:

$765.42_8 =$

$$7 \times 8^2 + 6 \times 8^1 + 5 \times 8^0 + 4 \times 8^{-1} \times 2 \times 8^{-2} =$$
$$448 + 48 + 5 + .5 + .03125 = 501.53125_{10}$$

1101.01_2 =

$$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} =$$
$$8 + 4 + 0 + 1 + 0 + .25 = 13.25_{10}$$

$010\ 111\ 001\ 100\ 011_2 = 27143_8 =$

$$2 \times 8^4 + 7 \times 8^3 + 1 \times 8^2 + 4 \times 8^1 + 3 \times 8^0 =$$
$$8192 + 3584 + 64 + 32 + 3 = 11,875_{10}$$

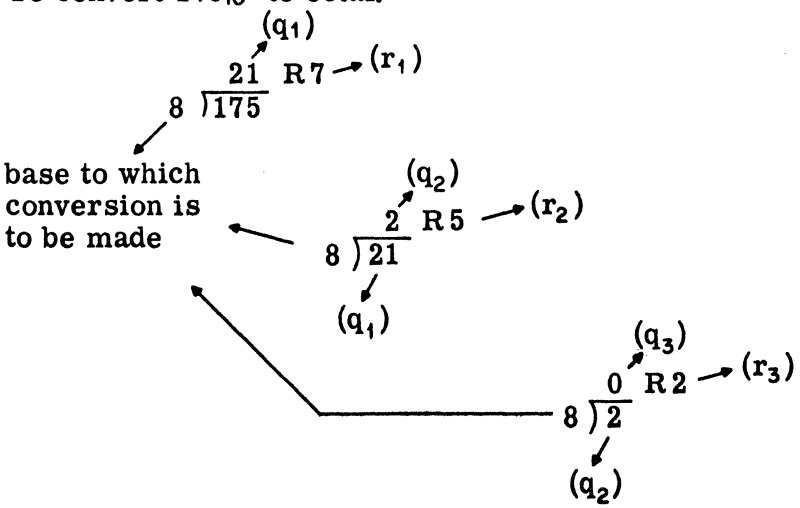
(In this example, rather than work with the longer binary number, we first convert to octal, then to decimal.)

Decimal to Octal/Binary

To convert decimal numbers to binary and octal involves a slightly more complicated process: divide the decimal number by the base to which conversion is to be made, attaining a quotient (q_1) plus a remainder (r_1). Divide q_1 by the base, again attaining a quotient (q_2), plus a remainder (r_2). Repeat this process until a zero quotient plus a remainder (r_n) is obtained. The converted number is $r_n r_{n-1} \cdots r_3 r_2 r_1$.

Examples:

To convert 175_{10} to octal:



$175_{10} = 2\ 5\ 7_8$
$(r_3\ r_2\ r_1)$

To convert 175_{10} to binary:

$$2\overline{)175}\quad 87\ \text{R}1 \rightarrow (r_1)$$

$$2\overline{)87}\quad 43\ \text{R}1 \rightarrow (r_2)$$

$$2\overline{)43}\quad 21\ \text{R}1 \rightarrow (r_3)$$

$$2\overline{)21}\quad 10\ \text{R}1 \rightarrow (r_4)$$

$$2\overline{)10}\quad 5\ \text{R}0 \rightarrow (r_5)$$

$$2\overline{)5}\quad 2\ \text{R}1 \rightarrow (r_6)$$

$$2\overline{)2}\quad 1\ \text{R}0 \rightarrow (r_7)$$

$$2\overline{)1}\quad 0\ \text{R}1 \rightarrow (r_8)$$

$175_{10} = 10101111_2$ (Convert this to octal and compare with the answer to the previous example.)

To convert $28,768_{10}$ to binary:

$$8\overline{)28,768} \quad 3\ 596 \text{ R0}$$

(rather than make numerous divisions by 2, we first convert to octal, then to binary)

$$8\overline{)3596} \quad 449 \text{ R4}$$

$$8\overline{)449} \quad 56 \text{ R1}$$

$$8\overline{)56} \quad 7 \text{ R0}$$

$$8\overline{)7} \quad 0 \text{ R7}$$

$$28,768_{10} = 70140_8 = 111\ 000\ 001\ 100\ 000_2$$

To convert fractional decimal numbers to octal or binary.

Multiply the fraction times the base to which conversion is to be made, attaining an integer (i_1) and a fraction (f_1). Multiply f_1 times the base, again attaining an integer (i_2) and a fraction (f_2). Repeat this process until an integer (i_n) and a zero fraction is obtained, or until the desired degree of accuracy is obtained. The converted number is $.i_1 i_2 i_3 \ldots i_n$.

Examples:

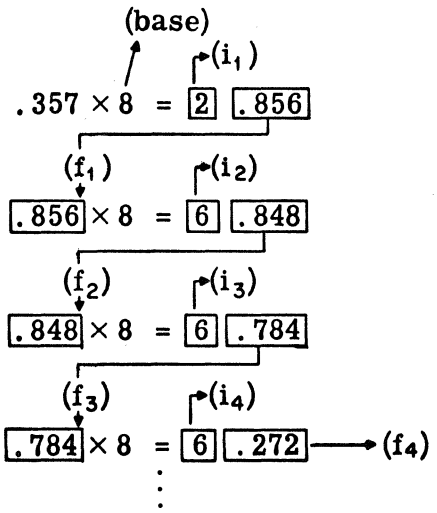
To convert $.75_{10}$ to octal:

$$.75 \times 8 = \boxed{6}.\boxed{00} \rightarrow (f_1)$$

(base) (i_1)

$$.75_{10} = .6_8$$

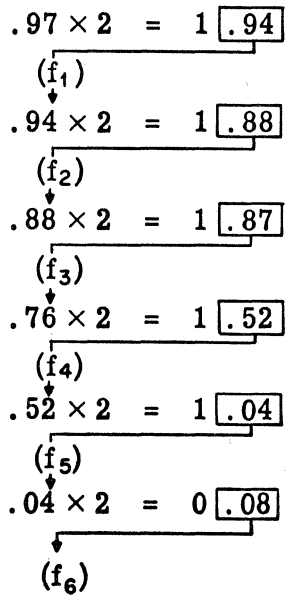
To convert $.357_{10}$ to octal:



and so forth

$.357_{10} = .2666_8$ (accurate to four places)

To convert $.97_{10}$ to binary:



and so forth

$.97_{10} = .1111100_2$ (accurate to 7 places)

To convert $.97_{10}$ to octal, then to binary:

$.97 \times 8 = 7\ .76$

$.76 \times 8 = 6\ .08$

$.08 \times 8 = 0\ .64$

$.64 \times 8 = 5\ .12$

$.12 \times 8 = 0\ .96$

$.96 \times 8 = 7.68$

:

.

$.97_{10} = .760507_8 = .111\ 110\ 000\ 101\ 000\ 111_2$

(accurate to 18 places)

To convert a mixed decimal number to octal or binary, the previous two processes are combined:

Example:

To convert 321.42_{10} to octal:

$8\overline{)321}$ 40 R1 $\qquad .42 \times 8 = 3.36$

$8\overline{)40}$ 5 R0 $\qquad .36 \times 8 = 2.88$

$8\overline{)5}$ 0 R5 $\qquad .88 \times 8 = 7.04$

$321.42_{10} = 501.327_8$ (accurate to 3 places)

1.3 ARITHMETIC OPERATIONS

Arithmetic operations in the octal and binary number systems follow the same rules as for the decimal number system.

1.3.1 ADDITION

In the decimal number system, a carry is generated each time the addition in a column reaches the base (10) or an integer multiple of the base. The difference between the sum and the base multiple is then placed in the column being added as part of the answer. The same is true in octal and binary.

For example:

decimal

```
Carry→2
      7
     +6
     +5
     +4
     22
```

The sum in the rightmost column is 22. The base has been reached twice, indicating a carry of 2. The difference between the sum and the base multiple is 2(22-20=2), which is placed in the rightmost column as part of the answer. Nothing but the carry is added in the second column.

octal

```
Carry→2
      7
     +6
     +5
     +4
     26
```

The sum of the rightmost column is 22 in decimal. The base has been reached twice, indicating a carry of 2. The difference between the sum and the base multiple is 6(22-16=6) which is placed in the rightmost column as part of the answer. Nothing but the carry is added in the second column.

binary

```
Carries→1
       111
        11
       +11
       +11
      1001
```

The sum of the rightmost column is 3 in decimal. The base has been reached once, indicating a carry of 1. The difference between the sum and the base multiple is

1(3-2=1), which is placed in the rightmost column as part of the answer. The carry is added in the second column for a partial sum in decimal of 4. The base has been reached twice, indicating a carry of two 1's. The difference between the sum of the second column and the base multiple is 0(4-4=0), which is placed in the second column as part of the answer. The carries are added in the third column for a decimal answer of 2. The base has been reached once, indicating a carry of 1. The difference between the sum of the third column and the base multiple is 0(2-2=0), which is placed in the third column as part of the answer. Only the carry is added in the fourth column.

1.3.2 SUBTRACTION

Borrows from the preceding column have the value of the system base. In the decimal system, the borrow is 10; in octal, 8, and in binary, 2.

For example:

decimal	octal	binary
101010 ← borrows	888 ← borrows	222 ← borrows
9123	7123	1010
-798	-567	-101
8325	6334	101

1.3.3 MULTIPLICATION

As in addition, a carry is generated each time a product reaches a multiple of the base. The partial products are added in the same system as the one in which multiplication is taking place.

For example:

decimal	octal	binary
394	274	111
×5	×5	×11
550	234	111
142 ← multiplication carry	142 ← multiplication carry	111 ← addition carry
1970	1654	1001
		11
		10101

decimal	octal	binary
563	563	1111
×75	×75	×11
505	167	1111
231 → carries	331 → carries	1111
521 →	325 →	10001
342	452	111 → addition carries
42225	54147	01101
		1
		101101

1.3.4 DIVISION

Binary or octal division is the same as decimal division except that intermediate multiplications and subtractions must be performed in the appropriate system. Borrows for subtraction and carries for multiplication are not shown below.

decimal	octal	binary
563	563	1111
75)42225	75)54147	111)1101001
375	461	111
472	604	1100
450	556	111
225	267	1010
225	267	111
		111
		111

1.4 COMPUTER ARITHMETIC

Addition is the basic arithmetic operation for the computer The seventy basic instructions in the HP 2116 include an "add" instruction, but not subtract, multiply, or divide. The Assembler for the HP 2116 contains these instructions; they are constructed from other basic computer instructions. The following paragraphs deal with computer representation of negative numbers and computer subtraction.

A negative number is represented in the computer as the complement of the positive value. There are various kinds

of complements; the HP 2116A uses the base complement, or two's complement. For the decimal number system, the base complement is the ten's complement; for octal, the eight's complement.

These complements are formed by subtracting the numbers from an integer power of the base.

decimal	octal	binary
1000	1000	100000
678	543	11011
322 – 10's complement of 678_{10}	235 – 8' complement of 543_8	101 – 2's complement of 11011_2

With the base complement, it is possible to subtract by complementing the subtrahend and adding, disregarding the final carry. The final carry is that which extends beyond the leftmost digit of the minuend or subtrahend, whichever is longer.

The base complement of the subtrahend is found by subtracting the subtrahend from the integer power of the base which is one digit position longer than the minuend or subtrahend, whichever is longer.

For example:

decimal

968 minuend	968 minuend
-367 subtrahend	+633 10's complement of subtrahend
601	1601
	throw away final carry
33269 minuend	33269 minuend
-249 subtrahend	+99751 10's complement of subtrahend
33020	133020
	throw away final carry
296 minuend	296 minuend
-3295 subtrahend	+6705 10's complement of subtrahend
-2999	7001 negative answer in 10's complement form. No final carry.

In the last example, a larger number is subtracted from a smaller number; the answer is negative. The answer 7001 is in 10's complement form; by taking the 10's complement of 7001 (10,000 - 7001 = 2999), it is seen that the answer is correct.

octal

```
 6576 minuend          6576 minuend
-3257 subtrahend      +4521 8's complement of subtrahend
 3317                 1)3317
                       throw away final carry

777777 minuend        777777 minuend
  -555 subtrahend     +777223 8's complement of subtrahend
777222                1)777222
                       throw away final carry
```

binary

```
101101 minuend          101101 minuend
 -1011 subtrahend      +110101 2' complement of subtrahend
1010010                 1)100010
                         throw away final carry

11001 minuend           11001 minuend
 -111 subtrahend       +11001 2's complement of subtrahend
10010                   1)10010
                         throw away final carry

  101 minuend             101 minuend
-11001 subtrahend       +00111 2's complement of subtrahend
                                negative answer in 2's complement form
-10100                    1100
```

In the last example, a larger number is subtracted from a smaller number; the answer is negative. The answer 1100 is in two's complement form; by taking the 2's complement of 1100(100000-01100 = 10100), it is seen that the answer is correct.

Another type of complement useful in working with computers is the one's complement of a binary number. The one's complement is formed simply by changing 1's to 0's and 0's to 1's. For example, the 1's complement of the binary number 100111011_2 is 011000100_2. The one's complement is useful in that the two's complement of a binary number can be formed by taking the one's complement of the number and adding 1.

For example:

11011	original number
00100	one's complement
+ 1	
00101	two's complement

REVIEW

1. The computer recognizes information as patterns of _____.

2. What are the two general types of software translators?

3. A number system is characterized by its _____ and its _____.

4. What is the base of the (a) binary number system?
   (b) octal number system?
   (c) decimal number system?

5. Define "complement" as applied to the computer.

Exercises

6. Convert:

(a) 110111_2	to decimal	(e) 512_{10}	to binary
(b) 312_{10}	to binary	(f) 111011100_2	to decimal
(c) 7658_{10}	to octal	(g) 398.75_{10}	to octal
(d) 32777_8	to decimal	(h) 277.0053_8	to decimal

7. Find the solution for:

(a) $10110111_2 + 111011_2$

(b) $32767_8 + 256_8$

(c) $3122_8 - 777_8$

(d) $101110_2 - 11_2$

(e) $10111_2 \times 101_2$

(f) $26_8 \overline{)12472_8}$

8. Convert the following to their base complement:

(a) 1111110_2	(c) 97654_{10}	(e) 101010101_2
(b) 377_8	(d) 529_{10}	(f) 101011_8

THE XYZ COMPUTER 2

A general discussion of computer hardware was given in the Introduction. To illustrate a few of these hardware requirements more specifically, we shall examine the central processor of a hypothetical device called the XYZ computer.

First, let us examine some of the properties we would like the central processor to have:

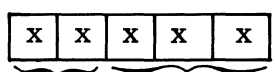
1. The ability to recognize and execute a set of instructions.

2. The ability of these instructions to refer to data stored in memory.

2.1 INSTRUCTION FORMAT

The computer is constructed to process information in units of a specified number of bits. These units are called words. To simplify this discussion, we shall define the XYZ computer as constructed to process 5-bit words. By defining the size of the word, we implicitly set a limit to the computer's memory size (number of words in memory) and also to the instruction repertoire (number of instructions or commands which the computer can recognize).

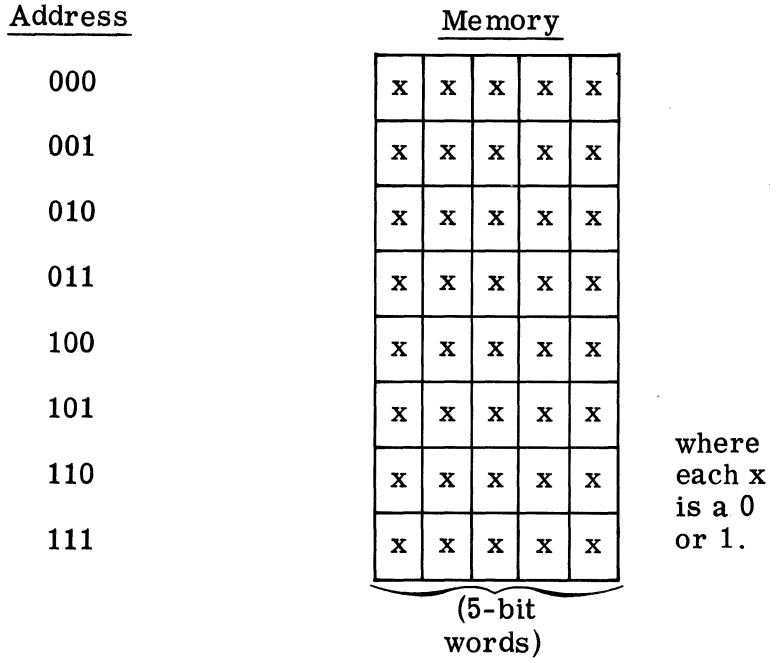
Each instruction the computer will recognize must be contained in five bits. Therefore, if each instruction is recognized as a certain pattern of bits, thirty-two (2^5) different instructions could be defined. However, most of the capabilities we would like also involve operands, or data to be processed. For example, the instruction "add" is meaningless without quantities to sum. So, part of the instruction word must be used to give the operation and part must be used to refer to an operand. Suppose the XYZ instruction word is defined as follows:



where each x is a 0 or 1.

operation ← operand reference (address)

The number of commands is now limited to four (2^2) and the number of operand references to eight (2^3). Since the operands are stored in memory, this means that we may refer to eight different words or memory locations. Thus the memory size has been effectively limited to eight locations. References to memory locations are made through binary addresses, permanently fixed by construction to each location as represented by the diagram below:



where each x is a 0 or 1.

2.2 ACCUMULATOR

In the previous section, the means of referring to one operand in memory was discussed. For operations such as addition, however, we need to refer to another operand. There is no more room in the instruction word to refer to another operand. Therefore, the XYZ computer contains a register called the accumulator, or A-register, which can be used to hold an operand. The "add" instruction obtains one operand from memory and the other from the A-register, adds them and stores the result back in the A-register. The A-register is 6 bits in lengths; the high-order bit is used for overflow, when an add or other operation creates a number longer than 5 bits. For example, if $1011_2(23_{10})$ is added to 10000_2 (16_{10}), the answer 100111_2 (39_{10}) cannot be contained in 5 bits. The 1 is carred into the 6th bit to indicate that overflow has occurred.

2.3 INSTRUCTIONS

The XYZ instruction repertoire is defined as follows:

Operation Code

binary	octal	Mnemonic	Result
00	0	LDA	load the A-register with the contents of the memory location specified by the last 3 bits of the instruction. The contents of the memory location are unmodified.
01	1	ADA	add the contents of the memory location specified by the last 3 bits of the instruction to the contents of the A-register; store the result in the A-register. The contents of the memory location are unmodified.
10	2	HLT	halt; stop processing.
11	3	STA	store the contents of A in the memory location specified by the last three bits of the instruction. The contents of A are unmodified.

2.4 OTHER REGISTERS

Other registers in the central processor receive and process instructions:

T-REGISTER

The T-register, or transfer register is a 5-bit register which holds instructions and data as they are being transferred between memory and the other registers.

I-REGISTER

The I-register, or instruction register, is a 2-bit register which receives, recognizes, and initiates execution of the instruction code after an instruction has been transferred from memory to the T-register.

M-REGISTER

The M-register, or memory address register, is a 3-bit register which receives the address portion of an instruction which has been transferred from memory to the T-register.

P-REGISTER

The P-register, or program counter, is a 3-bit register which holds the memory address of the instruction which is currently being executed.

Instructions and data flow between registers in the following manner:

1. Fetch instruction--transfer instruction from memory to T-register; then to I and M registers. If operand is required, go to step 2; if not, to step 3.

2. Fetch operand--transfer operand to T-register.

3. Execute--perform the desired function.

4. Increment the P-register by 1, replace the contents of M with the contents of P, go to step 1.

2.5 SAMPLE PROGRAM

A simple program to add the contents of two memory locations and store the result in a third location is given below.

Assume that the program and the values have been stored in memory at some previous time.

Mnemonic	Address	Contents of memory before execution.					
LDA 7	000	0	0	1	1	1	
ADA 6	001	0	1	1	1	0	
STA 5	010	1	1	1	0	1	
HLT	011	1	0	0	0	0	
	100	x	x	x	x	x	(x's may be 1's or 0's)
	101	x	x	x	x	x	
	110	0	0	1	0	1	← This is added to location 7
	111	0	1	1	1	1	

Contents of memory after execution.

000	0	0	1	1	1	
001	0	1	1	1	0	
010	1	1	1	0	1	
011	1	0	0	0	0	
100	x	x	x	x	x	(x's may be 1's or 0's)
101	1	0	1	0	0	← now contains result
110	0	0	1	0	1	← unchanged
111	0	1	1	1	1	← unchanged

REVIEW

1. A computer is constructed to process information in units of bits called _____.

2. References to locations in memory are made through binary _____.

3. Certain patterns of bits are recognized by the computer as _____.

4. Operands may be located in _____ or in the _____, or both, depending upon the instruction.

5. Define "overflow".

6. A series of instructions resulting in the solution to a particular problem may be termed a _____.

7. A computer having a 12-bit word divided into a 5-bit operation code field and a 7-bit operand address field would be able to refer to _____ memory locations.

THE HP 2116 COMPUTER 3

The XYZ computer illustrated important computer concepts: the idea of word length, patterns of bits being recognized as instructions, a program consisting of a series of instructions stored in consecutive memory locations, execution of instructions one at a time from these memory locations, and so forth.

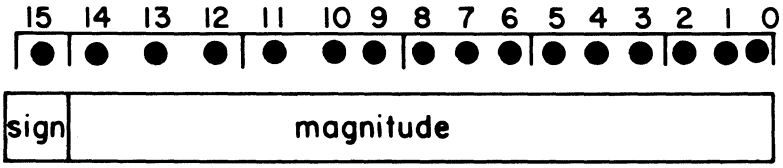
The underlying theory of memory addressing, instruction word format, and registers are the same in the XYZ and the 2116. The difference is primarily in size of the machine word and a somewhat larger set of registers. However, these differences expand the capabilities considerably.

3.1 INSTRUCTION AND FORMAT DATA

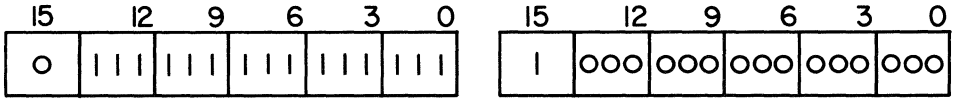
The 2116 processes data in 16-bit words. An operand data word is handled as shown below. The instruction word format varies according to the type of instruction.

3.1.1 DATA FORMAT

Data used as an operand is formatted as follows:

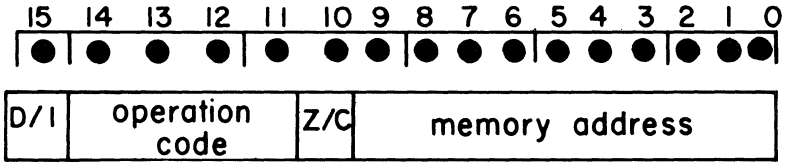


The sign bit indicates positive (bit 15=0), or negative (bit 15=1). Negative data is stored in two's complement form. Thus, a value may range from $+32767_{10}$ to -32768_{10}.



3.1.2 MEMORY REFERENCE INSTRUCTIONS

Instructions which refer to locations in memory are formatted as follows:

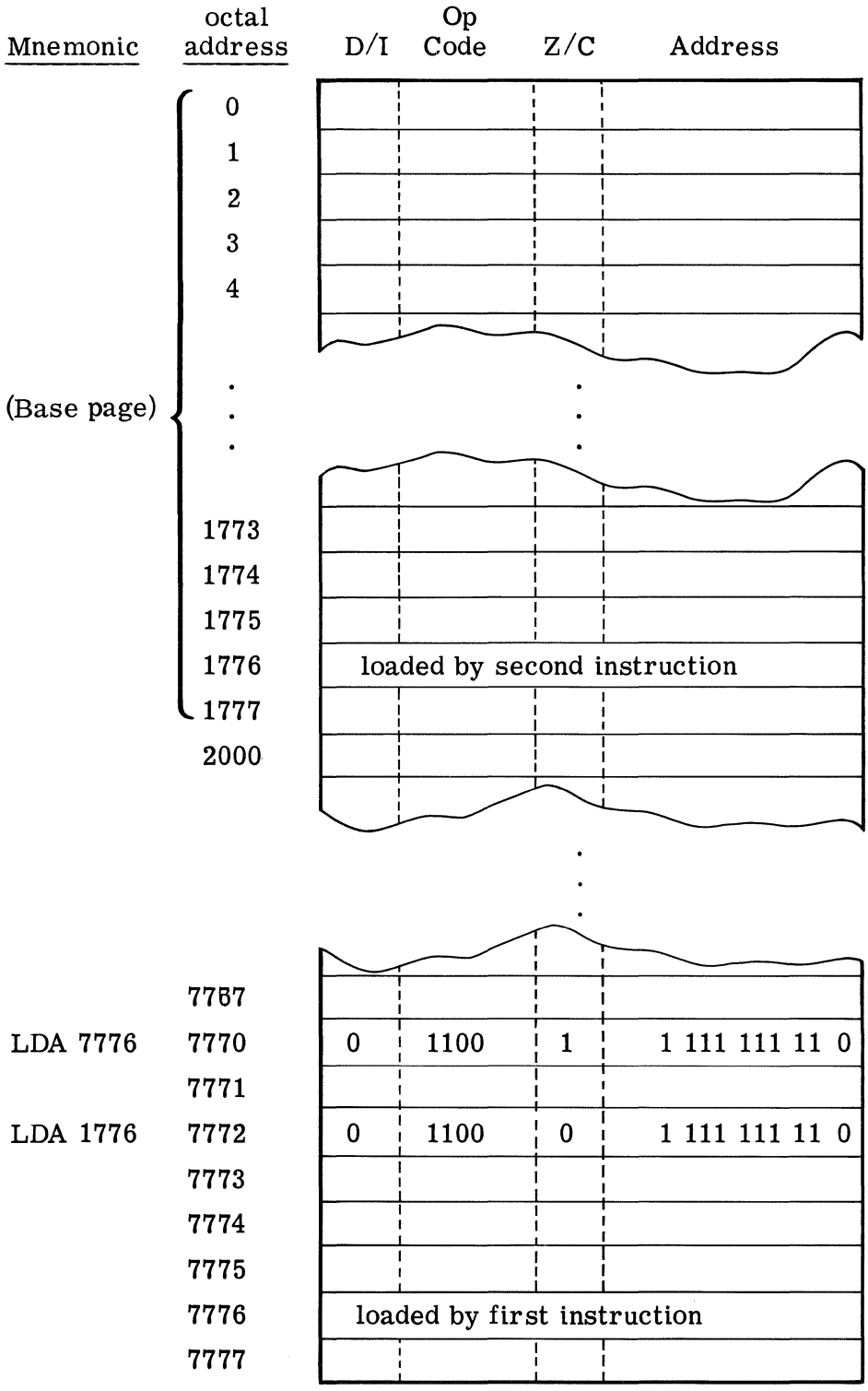


Operation code and memory address have the same function here as in the XYZ computer. Note that the memory address field is 10 bits long; this implies that we can refer to $2^{10} = 1024_{10}$, memory locations. However, with the Z/C bit it is possible to expand the number of addressable locations to 2048_{10}. With the D/I bit, the number of addressable locations is expanded to a maximum of $32,768_{10}$ (32K). The minimum amount of memory space provided with the 2116A is 4096 (4K) words.

A theoretical division of the basic 4K memory is made in $1,024_{10}$ - word blocks called pages. The zero page, or base page, occupies locations $0\text{-}1777_8$. Pages 1, 2, and 3 occupy locations $2000\text{-}3777_8$, $4000\text{-}5777_8$, and $6000\text{-}7777_8$, respectively.

Base (Zero)/Current Page

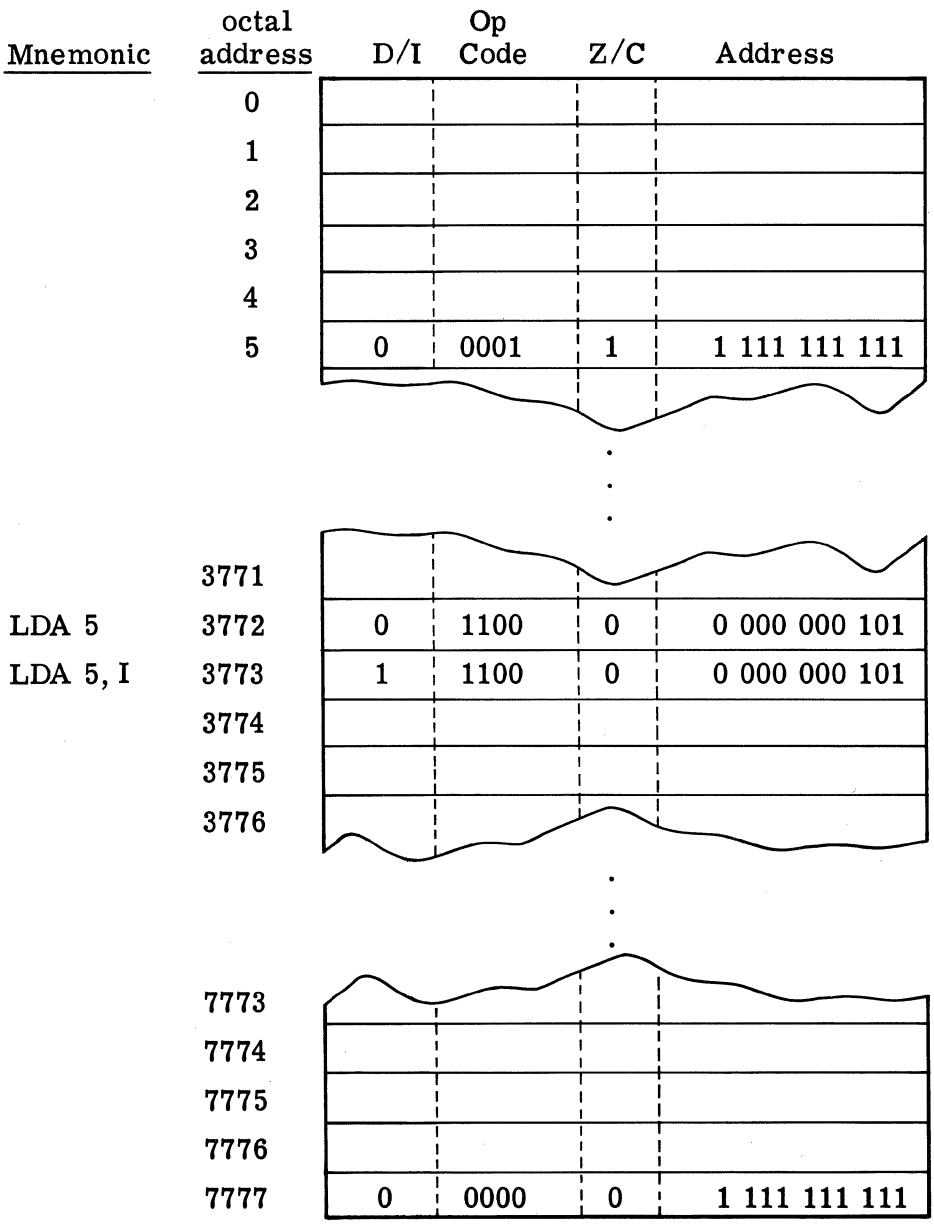
The Z/C bit determines whether the instruction address refers to a location in the base page (Z/C = 0) or the current page (Z/C = 1). The current page is the page in which the instruction is located. For example, the diagram below represents two instructions from a program located in page 3. The first instruction refers to an address in the current page. The second instruction refers to an address in the base page. Thus, the Z/C bit doubles the number of directly addressable memory locations.


Mnemonic
octal address
D/I
Op Code
Z/C
Address
0
1
2
3
4
(Base page)
1773
1774
1775
1776
1777
2000
loaded by second instruction
7767
LDA 7776
7770
0 1100 1 1 111 111 11 0
7771
LDA 1776
7772
0 1100 0 1 111 111 11 0
7773
7774
7775
7776
7777
loaded by first instruction

Direct/Indirect Addressing

The D/I indicates direct or indirect addressing. With direct addressing, the contents of the instruction address is the operand used. Direct addressing is indicated by D/I = 0. With indirect addressing, the contents of the instruction address is used as a 15-bit operand address. Indirect addressing is indicated by D/I = 1.

In the example shown below, the notation (x) is used to denote the contents of x. For example, (A) means the contents of the A-register; (77_8) means the contents of location 77_8.



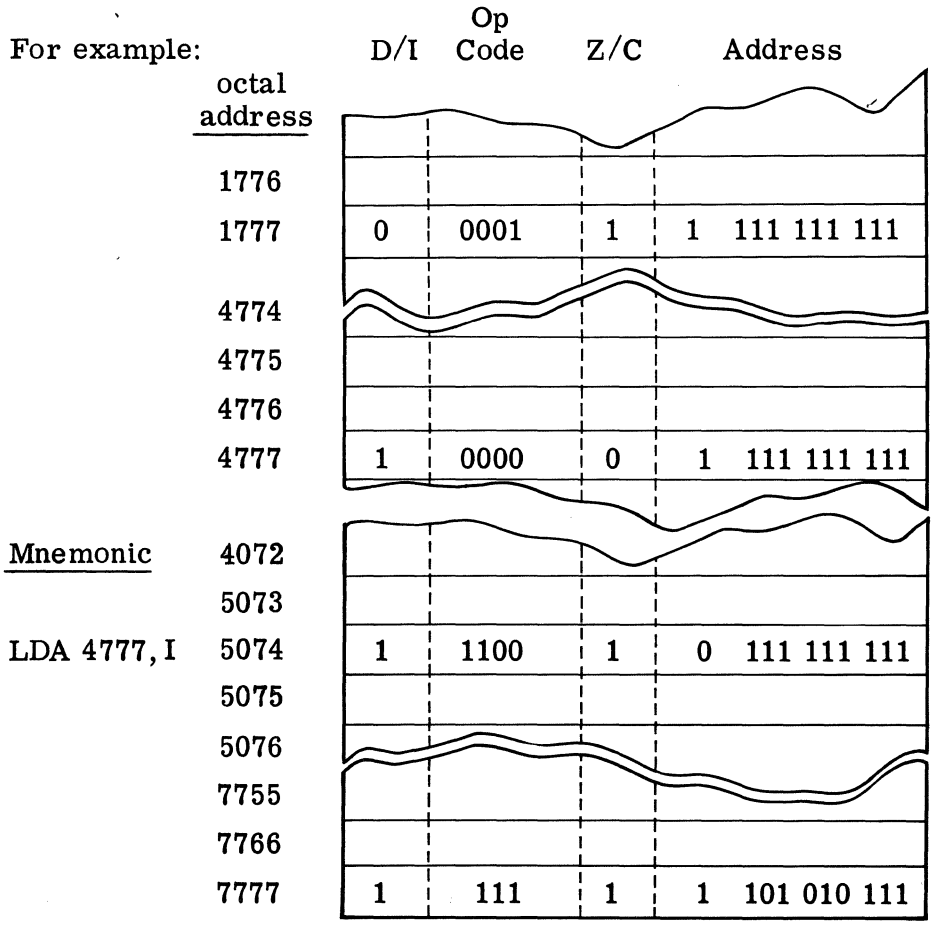
The first LDA instruction refers to the address 5. Since direct addressing is indicated by D/I = 0, (5), or 007777_8 is loaded into the A-register.

The second LDA instruction also refers to the address 5. However, since indirect addressing is specified by D/I = 1, (5) is used as the operand address, and (7777), or 001777_8 is loaded into the A-register.

With indirect addressing, then, 15-bit addresses can be used which allow us to refer to $2^{15} = 32{,}768_{10}$ memory locations with addresses from 0 to 77777_8. This is done at the expense of using 2 words for each instruction, one for the instruction and one for the address.

If the contents of the instruction address (location 5 in the above example) also contains a 1 in bit 15, the D/I bit, the contents of the 15-bit address is used as an address, and so on to any level.

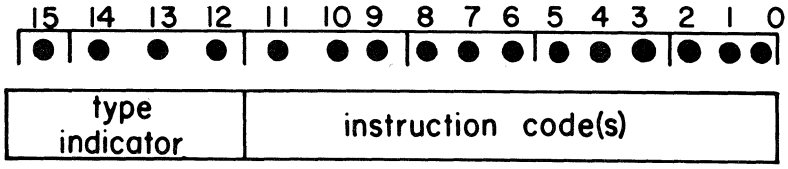
For example:



The LDA instruction refers to location 4777_8. Since indirect addressing is specified, location 4777_8 is assumed to be an address. However, location 4777_8 contains a D/I bit which is set to 1, and (4777_8) is treated not as an address, but as the address of an address. $(4777_8) = 1777_8$, the address of the address of the operand. $(7777_8) = 7777_8$, the address of the operand. $(7777_8) = 177527_8$, the actual operand which is loaded into the A-register.

3.1.3 REGISTER REFERENCE INSTRUCTIONS

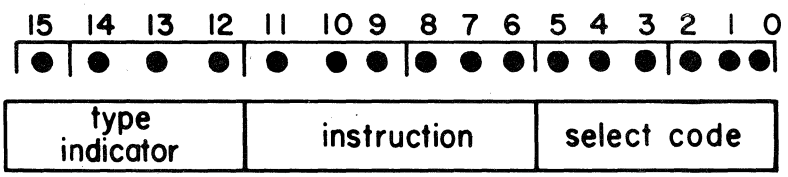
Instructions which manipulate registers are formatted as follows:



The type indicator is set to all zeros to indicate register reference instructions. The other 12 bits specify the command or combination of commands.

3.1.4 INPUT/OUTPUT INSTRUCTIONS

Instructions which control data transfer between the computer and input/output devices are formatted as follows:



The type indicator is set to 1000 to indicate input/output instructions.

The instruction portion of an I/O (input/output) command defines the operation to be performed.

I/O CHANNEL

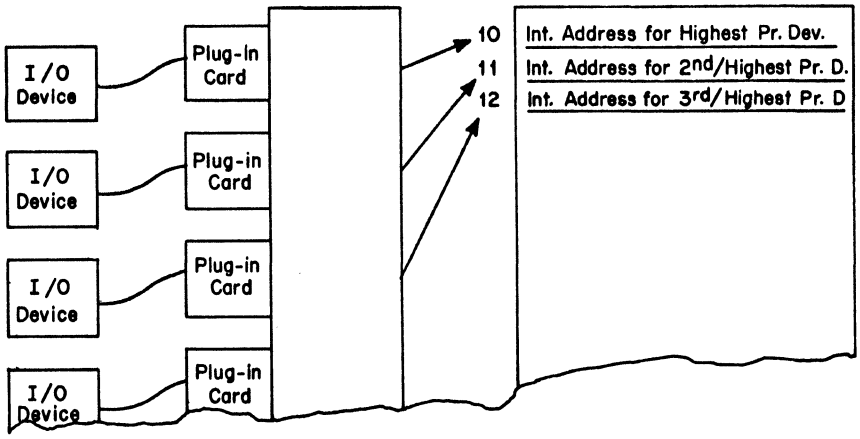
Data transfer takes place through the I/O hardware system. Up to this point, the discussion of computer hardware has been centered around the manipulation of data between the computer memory and the arithmetic registers. The I/O hardware system provides interface (a common boundary, or meeting point) between the central processor and external input/output devices; this interface allows transfer of data between the computer and external devices.

Up to sixteen I/O devices may be connected to the HP 2116 main unit; an HP 2155A I/O extender may be added to increase the total capability to 48 devices. Each device is connected to the computer through an interface component, consisting of a card and cable which is plugged into a slot in the main frame of the computer. Each interface card, or channel, consists of:

(a) an I/O buffer for temporary storage of data as it is being transferred. The buffer may be up to 16 bits in length; the actual length depends on the device being used. Since transfer of data to or from an external device takes a comparatively long time, the I/O buffer eliminates the necessity of tying up one of the working registers while the I/O operation is taking place. Thus, once an I/O operation is initiated, other instructions may be executed while the operation is taking place.

(b) a control bit which, in effect, "turns on" the I/O channel. When set, it enables the connected device to perform its I/O function and allows the flag to cause an interrupt.

(c) an I/O flag bit. This bit is set to 1 by a signal from an I/O device when an operation has been completed. When the interrupt system is enabled, setting the flag causes an interrupt. The currently executing instruction is interrupted and control is passed to an interrupt location associated with the device. The interrupt locations are in low order memory, in locations 10_8 through 77_8. The device having the highest priority is assigned location 10, the device having the next highest priority is assigned location 11, and so forth. Priority is determined by the slot in which the interface card for the device is placed.



The flag bit, when set, inhibits all interrupts on lower priority devices.

SELECT CODE

As shown in the input/output instruction format, bits 5-0 form a select code. This code provides the necessary reference to I/O device or function. The select codes correspond directly to the interrupt addresses of the I/O devices, and also to functions having interrupt addresses, such as Power Failure Interrupt.

Select Code Assignments

Select Code	Interrupt Location	Assignment
00	none	Interrupt System Disable/Enable
01	none	Switch Register or Overflow
02	none	DMA Channel 1 Initialize
03	none	DMA Channel 2 Initialize
04	4	Power Failure Interrupt
05	5	Memory Protect Interrupt
06	6	DMA Channel 1 Completion Interrupt
07	7	DMA Channel 2 Completion Interrupt
10	10	I/O Device, highest priority
11	11	I/O Device 2nd highest priority
.	.	.
.	.	.
.	.	.
77	77	I/O Device, lowest priority

As shown in the table above, some select codes are reserved for specific uses while others are available for assignment to any optional I/O device. The first five (octal codes 00-04) are reserved for non-interrupting functions. Select code 00 is reserved for enabling or disabling the interrupt system; certain I/O instructions using this select code set or clear the flag bit for all I/O devices. For certain input instructions, select code 01 refers to the 16 toggle switches on the computer console known as the Switch Register. Select codes 02, 03, 06, and 07 are reserved for use by Direct Memory Access, Option M11. Direct Memory Access is a hardware option which allows the transfer of blocks of data directly between an external I/O device and memory. As discussed above, the transfer of data between external device/buffer/working register takes place one element of data at a time. The length of the element depends upon the I/O device. Select code 05 is the highest priority interrupt, reserved for Power Failure Control.

3.2 REGISTERS

The HP 2116 contains 7 working registers:

T-Register

The 16-bit transfer register holds all data as it is transferred between memory and other registers in the control element.

P-Register

The 16-bit program counter holds the address of the instruction currently being executed. Only bits 0-14 are used. The P-register is automatically incremented after execution of each instruction.

M-Register

The 16-bit memory address register contains the address of the memory location currently being read from or written into.

A-Register

The 16-bit A-register is an accumulator and holds operands and the results of arithmetic and logical operations performed by programmed instructions. This register may be addressed by any memory reference instruction as location 00000, permitting inter-register operations such as "add B to A" with a single word instruction.

B-Register

The 16-bit B-register is a second accumulator which can be used in the same manner as the A-register, with the exceptions of the logical "and", "inclusive or", and Exclusive or" operations. The B-register may be referenced by any memory reference instruction as location 00001 for inter-register operations with A.

E-Register

The 1-bit extend register indicates a carry from bit 15 of the A or B-registers by any add or increment instruction. The E-register can be set, complemented, or tested; it can also be rotated in conjunction with the A- or B-registers.

OV-Register

The 1-bit overflow register indicates that an add or increment

instruction referring to the A- or B-register has caused one of these accumulators to exceed the maximum positive or negative number, $+32,767_{10}$ to $-32,768_{10}$. For positive numbers, this occurs when a carry is made from bit 14 to bit 15, implying that the result of the addition of two positive numbers is a negative number. For negative numbers, overflow occurs when a carry is not made from bit 14 to bit 15, implying that the result of the addition of two negative numbers is a positive number.

In both of the following examples, the OV-register would be set. In the later example the carry from bit 15 causes the E-register to be set.

Positive Overflow

(A or B) = 0 110 001 101 000 111

(memory) = 0 100 111 111 111 000

result in
A or B = 1 011 001 100 111 111

Negative Overflow

(A or B) = 1 001 110 000 111 101

(memory) = 1 011 000 000 000 000

result in
A or B = 0 100 110 000 111 101

[1] ← (E-register)

The OV-register can be cleared, set, or tested; a second overflow does not change the OV-register unless it has been cleared first.

3.3 OPERATION SEQUENCE

The HP 2116 operates using any of the following machine phases: Fetch, Indirect, Execute, Interrupt, and Halt. Each phase takes 1.6 microseconds, called a machine cycle, with the exception of the ISZ instruction, whose Execute phase takes 2.0 microseconds, and the Halt phase, which may be held indefinitely until the halt is terminated.

3.3.1 FETCH PHASE

The M-register is set equal to the P-register. The instruction whose address is indicated by the contents of the M-register is transferred to the T-register. Bits 15-10 of the instruction are transferred to a special instruction register. Processing continues according to the type of instruction:

Memory Reference Instructions

Bits 9-0 of the T-register are transferred to the M-register. (In the case of the JMP instruction, bits 9-0 of the T-register are transferred to bits 9-0 of the M- and P-register. The Fetch phase is then re-initiated if the D/I bit of the instruction =0; if D/I = 1, processing continues with the Execute phase.) If the Z/C bit of the instruction = 0, bits 15-10 of the M-register are cleared to zero; this causes reference to the zero (base) page. If Z/C = 1, bits 15-10 remain the same as bits 15-10 of the P-register; this causes reference to the current page. If the D/I bit of the instruction equals 1, the Indirect phase is initiated; otherwise, the Execute phase is initiated.

Register-Reference and Input/Output Instructions

These instructions require only one machine cycle; they are executed at this point. The P-register is incremented, and the fetch phase re-initiated.

3.3.2

INDIRECT PHASE The contents of the location whose address is specified by the contents of the M-register are transferred to the T-register, then to the M-register. If bit 15 of the T-register = 0, the execute phase is initiated; if bit 15 = 1, the indirect phase is re-initiated. (In the case of the JMP instruction, the contents of the T-register are also transferred to the P-register when bit 15 = 0, and the Fetch phase is re-initiated.)

3.3.3

EXECUTE PHASE The instruction is executed, the P-register is incremented, and the Fetch phase re-initiated.

3.3.4

INTERRUPT PHASE

The machine enters the interrupt phase when an interrupt occurs on an I/O device. The normal program sequence is halted, and the computer fetches the next instruction from one of the interrupt locations. The P-register is decremented by 1, and bits 15-6 of the M-register are cleared to zero. Bits 5-0 of the M-register are set to the select code of the interrupting device. The Fetch phase is re-initiated.

3.3.5

HALT PHASE The machine enters the halt phase when a HLT instruction is executed, or when the HALT button on the computer console is pushed. Machine processing is terminated and the interrupt phase is inhibited. Processing continues when the RUN button on the computer console is pushed.

REVIEW

1. The HP 2116 recognizes three basic types of instructions; these are:

2. The memory in the HP 2116A is divided into theoretical 1,024-word blocks called _____.

3. What is this division of memory based upon?

4. The Z/C bit allows reference to memory locations in the _____ page or the _____ page.

5. Indirect addressing is indicated by _____.

6. Indirect addressing allows reference to _____ memory locations.

7. The I/O hardware system provides interface between the _____ and the _____.

8. What are the three components of an I/O channel through which a programmer communicates with an external device?

9. What determines the interrupt priority of an I/O device?

THE ASSEMBLER 4

It has been illustrated that a computer program is a sequence of instructions which, when executed by the computer, solves a specific problem. The Assembler for the HP 2116 is itself a program: a sequence of instructions solving the problem of how to write computer programs more easily.

The assembler translates programs written in a symbolic language consisting of mnemonic operation codes, operands, and labels. The symbolic program which is input to the computer to be translated by the Assembler is called the source program. The translated binary program which is output as a result of the assembly process is called the object program. The object program may then be input to the computer for execution.

4.1 OPERATION CODES

Mnemonic operation codes are recognized by the Assembler to be translated as machine instructions or pseudo instructions.

Machine instructions are those built into the computer - the Assembler translates these instructions into the binary code which can be directly executed by the computer. For example, LDA is translated into the bit combination which is interpreted as "load the A-register."

Pseudo instructions: (1) provide information to the Assembler about the program being assembled, (2) allow definition of storage areas and constants, and (3) provide calls to arithmetic subroutines which perform often-used functions not available with any one machine instruction. For example:

(1) END tells the Assembler it has reached the end of the source program.

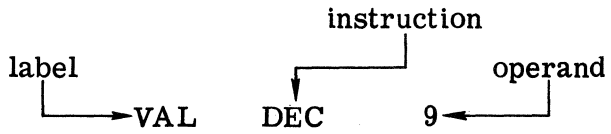
(2) DEC allows the user to define one or more decimal constants.

BSS allows the user to reserve a block of storage locations.

(3) FMP allows the user to multiply two values. This function is not available as one machine instruction; however, this code calls a subroutine (a group of instructions) which performs multiplication.

4.2 LABELS

A label for an instruction provides the ability to refer to the instruction or the value or storage area generated by the instruction. For example:



As instructions are input to be translated, the Assembler assigns the instructions to consecutive memory locations in the order they are input and maintains a table relating symbolic labels to the location or address assigned. In the above example, VAL would be related to the memory address where the decimal 9 generated by the DEC 9 pseudo instruction is stored.

4.3 OPERANDS

Some instructions require the designation of an operand. In some cases, the operand value is specified in the instruction, as the 9 is specified in the above example. In other cases, an operand address is specified. Symbolic operand addresses may be used, provided these symbols have been defined somewhere within the program. For example:

```
VAL   DEC   9
        .
        .
        .
      LDA   VAL
```

the symbolic operand, VAL, is defined by the label VAL in the DEC instruction, above.

The Assembler searches the symbol table for the address associated with VAL and uses this address in translating the instruction.

4.4 ABSOLUTE PROGRAMS

An absolute program is one whose addresses are not modified as a result of loading at object program execution time. For the program to execute correctly, the object program must be loaded into the same memory locations each time it is used. Consider the previous example:

```
      .
      .
      .
VAL   DEC   9
      .
      .
      .
      LDA   VAL
```

Suppose the starting location for the program had been set at assembly time to be 100_8, and the DEC 9 instruction translated to be at location 121_8. The LDA VAL instruction, then, has been translated as "load the A-register with the contents of location 121_8."

If it were possible for the object program to be loaded for execution starting at location 130_8, the decimal 9 resulting from the DEC 9 instruction would be at location 151_8. Therefore, the LDA VAL instruction, translated to load A with 121_8, is incorrect.

Absolute programs, then, must be loaded into the locations determined at assembly time, or all memory reference instructions will be incorrect.

4.5 RELOCATABLE PROGRAMS

When the user requests a relocatable assembly, the Assembler assigns relative addresses to instructions. The first instruction requiring memory space is assigned relative location 0, the second, relative location 1, and so on. Then, at the time the translated program is loaded into the machine for execution, the BCS Relocating Loader (see Section 5.12) adds the relative address to the starting address for each instruction using an operand address. The first available location is determined by the Relocating Loader and used as the starting location. For example:

Mnemonic Label	Relative Address	Opcode	Mnemonic Operand Address	Relative Operand Address
	0	LDA	QUAN	5
	1	ADA	ETHEL	6
	2	AND	MASK	7
	3	STA	QUAN	5
	4	HLT		
QUAN	5	BSS	1	
ETHEL	6	DEC	7	
MASK	7	OCT	777	

Suppose that the first available starting address determined by the loader is 2000_8. The Relocating Loader modifies the operand addresses by adding 2000_8.

Mnemonic Label	Location	Opcode	Mnemonic Operand	Actual Operand Address
	2000	LDA	QUAN	2000 + 5 = 2005
	2001	ADA	ETHEL	2000 + 6 = 2006
	2002	AND	MASK	2000 + 7 = 2007
	2003	STA	QUAN	2000 + 5 = 2005
	2004	HLT		
QUAN	2005	BSS	1	
ETHEL	2006	DEC	7	
MASK	2007	OCT	000777	

No matter where the program is loaded, the modified operand addresses always refer to the desired operands.

If the Relocating Loader encounters a Memory Reference instruction referring to a location in a page other than the current page, or page 0, a full 15-bit address is placed in an available location in the base page. The relocatable loader then provides an indirect reference to this location, which is then used as the operand address of the instruction. The same word in the base page is used if other similar references are made to the same location.

4.6 PROGRAM LOCATION COUNTERS

The program location counter is an Assembler-maintained counter which implements the absolute and relative address assignment discussed in the previous two sections.

When an absolute assembly is requested by the user, the value of the program location counter is set to the value indicated by the user in his request for an absolute program (See ORG, Section 9.1.2). The first instruction requiring memory space is associated with this absolute address value; the next instruction requiring memory space is associated with this value plus one, and so forth.

When a relocatable assembly is requested by the user, the program location counter is set to zero. The first instruction requiring memory space is associated with relative address zero, the next instruction requiring memory space with relative address one, and so forth.

Two other counters, the base page location counter, and the common location counter are maintained by the Assembler. The base page location counter is maintained for assigning instructions and data from a relocatable program to contiguous locations base page, at the user's request (see ORB, section 9.1.4). The common location counter is maintained for assigning data from a relocatable program to an area of common storage. Data in a common storage area may be referred to by different programs. (see COM, section 9.2.1)

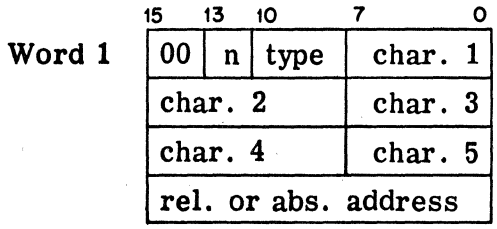
4.7 ASSEMBLER PROCESSING

The source program, punched on paper tape or prepared on some other medium, is input to the computer for translation. The Assembler performs its translation in two or three examinations of the source code. Each examination is called a pass. If any errors are found during these passes, the Assembler issues diagnostic messages. These messages are listed in Section 11.5.

4.7.1 PASS ONE

During the first pass, the symbol table is generated. Upon request, the symbol table may be listed during the first pass.

The format of a symbol table entry in memory is as follows:



00	not used	
n	number of words in entry (2-4)	
type	0	absolute
	1	relocatable
	2	base page relocatable
	3	common
	4	external

The length of the label symbol affects the size of the entry. A one-character symbol requires only two words; a full five-character symbol requires four words. There is a specific amount of storage available for the symbol table. When the number and length of the symbol table entries exceeds the amount of storage available, the symbol table will overflow. When this occurs, it is necessary to reduce the size of the table by reducing the number of labels or their length.

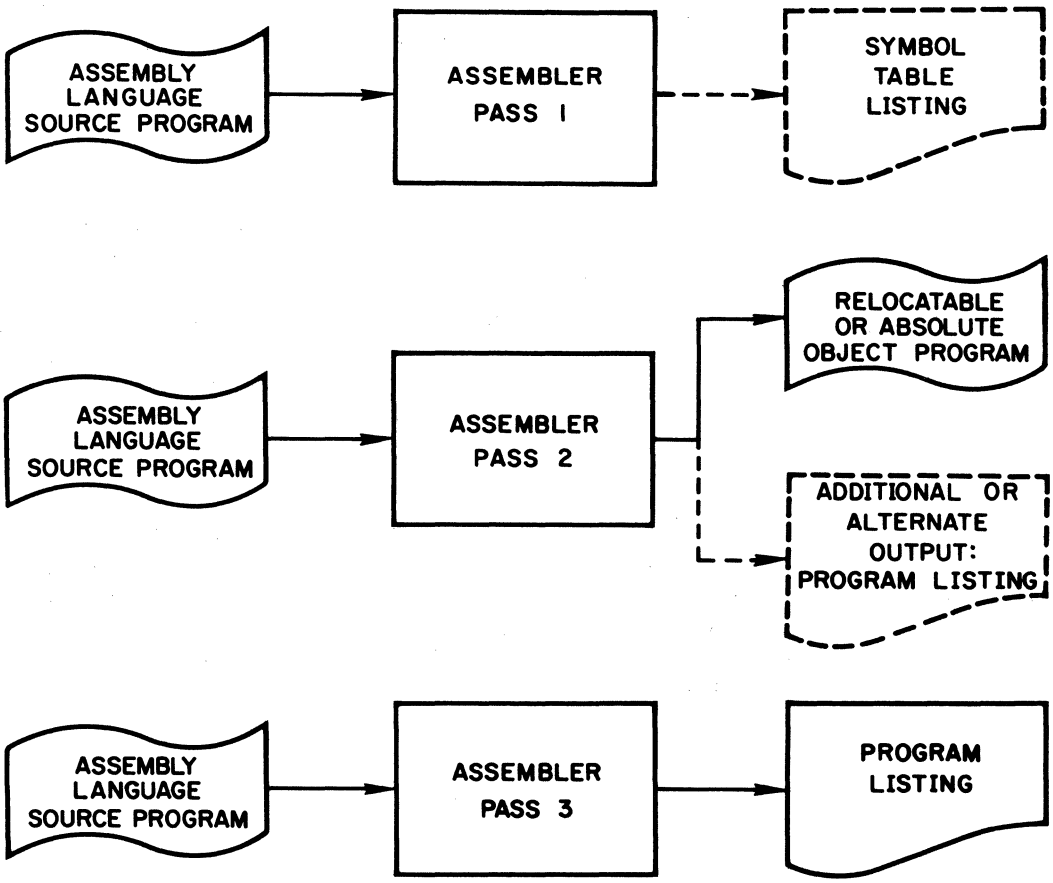
The Assembler is designed such that when an absolute assembly is requested, the portion of the Assembler which provides relocatability is overlaid, or destroyed, by the symbol table. Thus, an absolute assembly allows a larger symbol table than a relocatable assembly. If several absolute and relocatable programs are being assembled consecutively, without re-loading the Assembler, the relocatable programs must be input before the absolute programs.

4.7.2 PASS TWO

After the first pass, the source program is reloaded and re-examined. During the second pass, the assembly is completed; operation codes are translated, and operand addresses are generated where specified. A translated binary object program or a program listing may be requested as output. Both may be requested if the necessary output devices are available.

4.7.3 PASS THREE

If both a program listing and an object program are requested, and only one punch output device is available, the object program is punched on the second pass, and the source program is input for a third pass. The program listing is generated on this third pass. The following diagram illustrates Assembler processing.



ASSEMBLER PROCESSING

REVIEW

1. The Assembler converts a symbolic _____ program into a translated binary _____ program which may be executed by the computer.

2. What are the two general types of instructions available to the Assembly-language programmer?

3. The Assembler creates a _____ which is used to relate symbolic labels to the address assigned to them.

4. Define an absolute program.

5. Define a relocatable program.

6. What is a "pass"?

7. The HP 2116 Assembler requires how many passes to complete an Assembly?

BASIC CONTROL SYSTEM 5

The Basic Control System (BCS) for the HP 2116 is a computer program which provides capabilities of use to both the assembly language programmer and the compiler language programmer.† BCS is constructed of several separate programs, each of which may be modified to meet the particular hardware requirements at an installation.

Some of the capabilities relating to the Assembler include loading programs, simplified I/O (Input/Output), and debugging (error detection) aids.

5.1 LOADING PROGRAMS

The manner in which programs of different types are brought into memory and given control forms an important part of the translation process. Two loader programs are used: the Basic Binary Loader and the Relocating Loader.

5.1.1 BASIC BINARY LOADER

The Basic Binary Loader is responsible for loading into memory all translated (binary) absolute programs. In addition to user object programs, this includes standard software systems that are in absolute form: BCS, FORTRAN, the Assembler, and so forth.

The Basic Binary Loader is not a part of BCS; it is a binary program which resides permanently in the last 64_{10} locations in memory. The Basic Binary Loader is loaded directly into memory, one location at a time, by manually setting the toggle switches on the computer console.

If the Basic Binary Disc Loader is used instead of the Basic Binary Loader, the operator must press PRESET before RUN.

---

† Only those aspects of BCS which relate to the Assembler and particularly the novice assembly language programmer are discussed in this manual. For a detailed description of BCS with complete operating instructions, refer to the Basic Control System Programmer's Reference Manual.

5.1.2 RELOCATING LOADER

The Relocating Loader is the portion of BCS which is responsible for loading translated (binary) relocatable programs and in the process, modifying the relative addresses provided by the Assembler such that they refer to the correct memory locations once the program is loaded. Memory references which cross page boundaries are also handled by the Relocating Loader. A full 15-bit address is placed in the base page and an indirect reference to the base page location is inserted in the Memory Reference instruction.

5.1.3 LOADING PROCESS

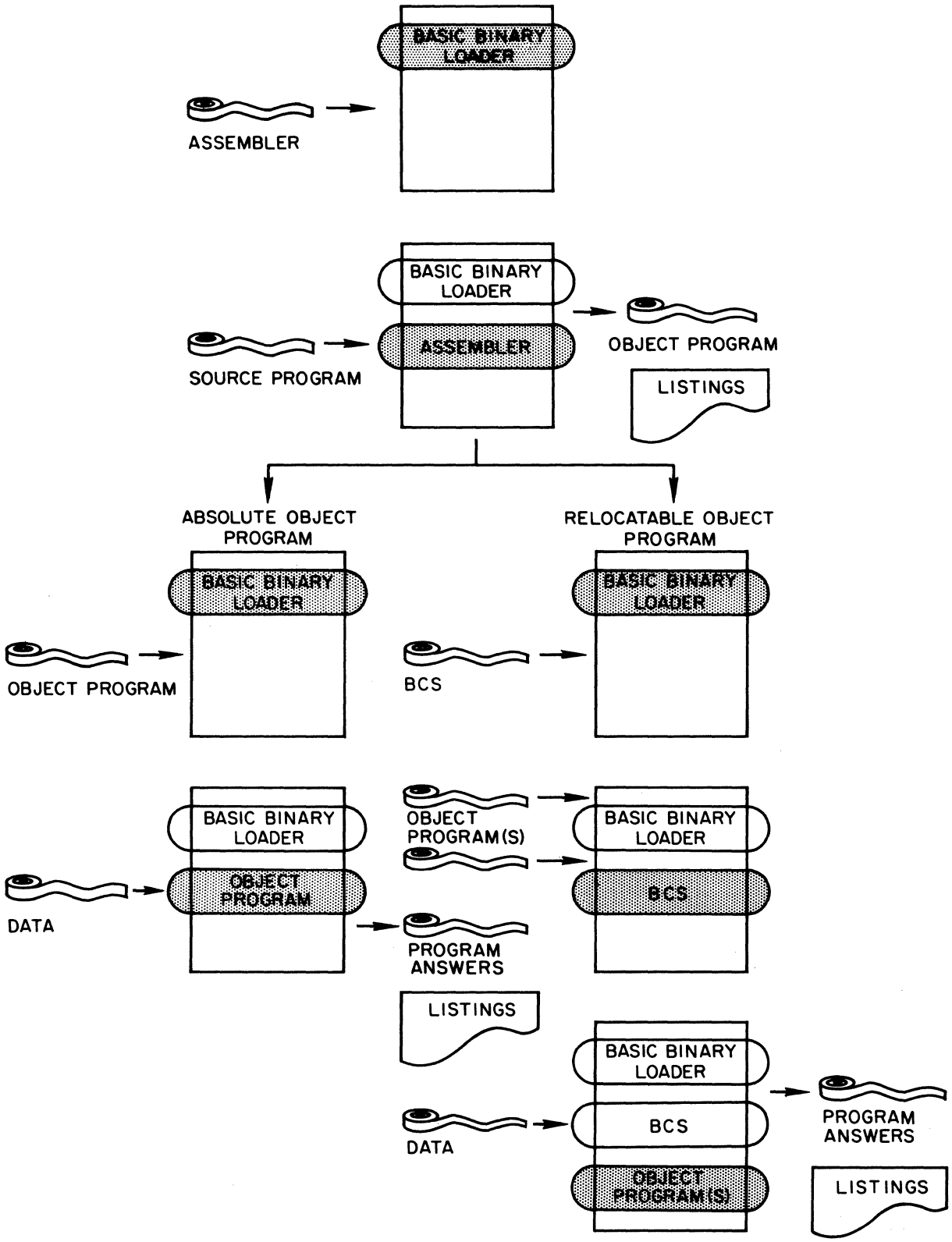
A source program may be coded such that it is translated as an absolute or relocatable program. The process of loading and executing differs somewhat in each case. First, the Assembler is loaded, using the Basic Binary Loader. When the loaded Assembler is given control, through manipulation of the console switches, it reads and translates the source program, producing an object program. If absolute, the object program may be loaded immediately into memory using the Basic Binary Loader, and executed. If relocatable, BCS must first be loaded, using the Basic Binary Loader. The Relocating Loader may then be requested to load the object program and give it control for execution.

5.2 INPUT/ OUTPUT

As illustrated by the previous sections, there are many ways to transfer information to and from a computer. The previous sections of this chapter described means of loading computer programs. These computer programs may in turn read data and write data to and from areas in the computer for processing.

The difference between loading data and reading data is mainly one of terminology, depending upon the purpose of the data. A program is loaded into memory for the purpose of execution. Data is read into a computer to be manipulated by an executing program. In short, data which is loaded acts; data which is read is acted upon. For example, the Assembler is loaded into the computer for execution; it then reads the source program and translates (manipulates) it, producing an object

LOADING PROCESS



The Shaded Blocks Indicate The Executing Program

program. The object program is then loaded into the computer for execution. This object program may in turn read data which it processes to form the results intended.

The Assembler contains input/output instructions which may be used to transfer data between the computer and various input/output devices. Input/output is a complicated process, however, and many machine instructions must be used to complete an I/O operation. The Basic Control System simplifies input/output by allowing the user to, in effect, tell BCS what is to be done and "letting BCS do it". This is accomplished through calling sequences in assembly language. The Input/Output Control program (.IOC.) of the Basic Control System interprets the call, initiates the operation, and returns control to the program making the request.

5.3 DEBUGGING AIDS

BCS provides various facilities for program testing, error detection, and error correction -- a process generally known to programmers by the title "debugging". The portion of BCS which provides these aids is the Debugging system, a relocatable program which is loaded into memory along with the user's relocatable object program.

The Debugging system supervises the execution of the user's object program; it interprets each instruction, takes action if indicated by a Debugging system control statement, and causes the instruction to be executed.

With the control statements, the user may modify the contents of storage locations and registers, stop execution of the object program at a certain point, display contents of registers and memory locations, and terminate the debugging process.

REVIEW

1. What facilities does BCS provide to the Assembly-language programmer?

2. What are the two loading programs available to the programmer to load object programs? What type programs do each load?

3. Which of the above loading programs is a part of the Basic Control System?

4. What is the difference between data that is read or written and data which is loaded?

5. Define "debugging."

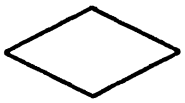
PROBLEM ANALYSIS (FLOWCHARTING) 6

The first and most important part of writing a computer program is analysis: reducing a problem's solution into logical steps. Flowcharting provides a helpful tool for this purpose. A flowchart is a graphical representation of a solution process. Flowcharts illustrate logic; errors in program logic can be found and corrected at the flowchart stage, saving wasted coding efforts. Flowcharts not only aid in preliminary design of a program, they provide valuable documentation after the program has been written.

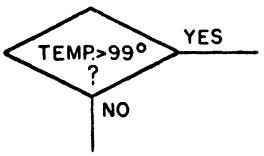
The basic flowchart symbols are given below:

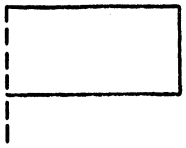
Input/Output. This symbol represents the transfer of data between the computer and an input/output device. For example:

Procedure. This symbol represents a process or operation to be performed. For example:

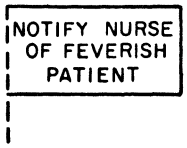


Decision. This symbol represents the determination of a factor from which several paths may be taken. For example:

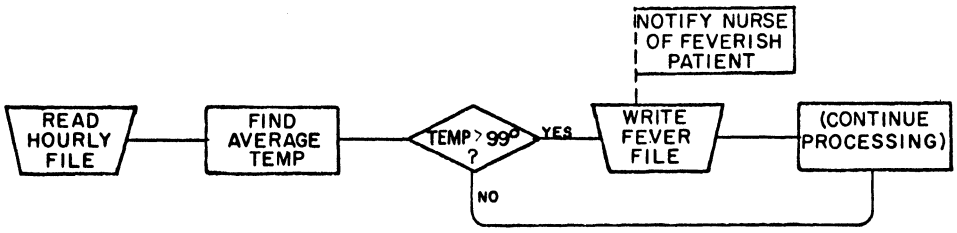




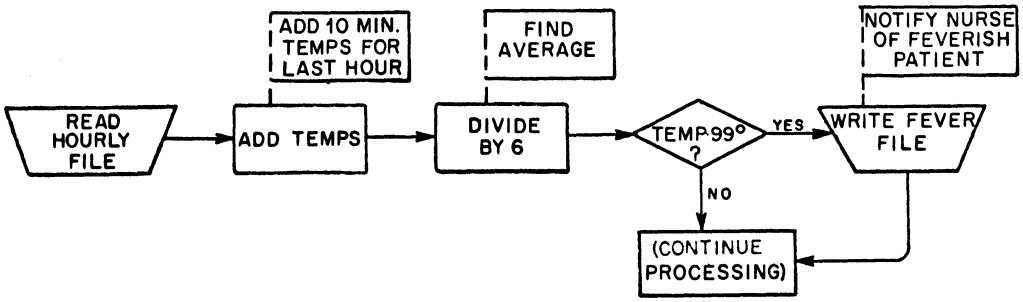
Annotation. This symbol provides explanatory or clarification notes. For example:



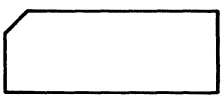
These and other specialized symbols discussed below are connected by directional lines to form a flowchart:



Normal flow direction is from left to right or top to bottom; however, when this is impossible or particularly cumbersome, arrows may be used to clarify opposite flow directions. For example, if we elaborate on the previous example:



Specialized symbols include:



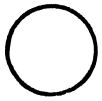
Punched card symbol.

Magnetic tape symbol.

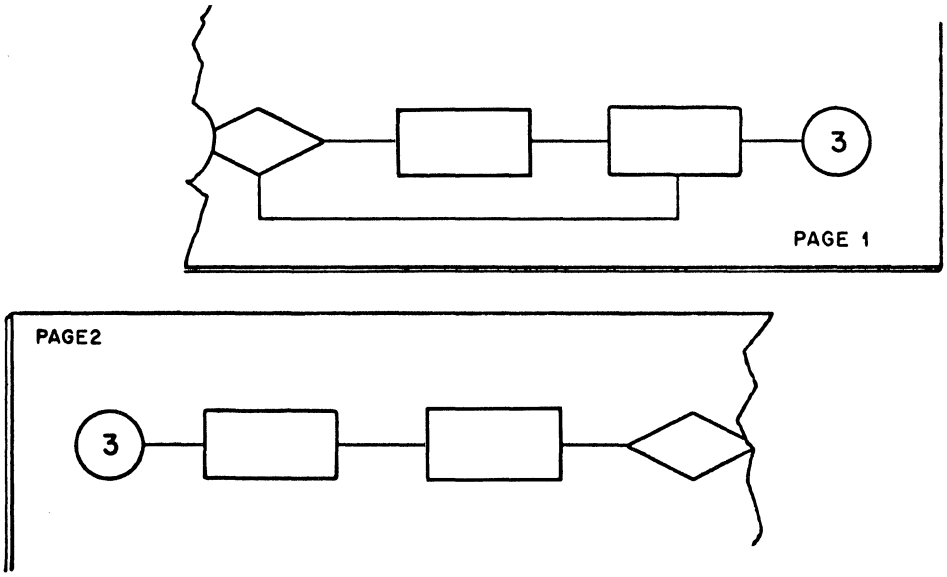
Punched tape symbol.

Document symbol.

Predefined process symbol. e.g., a subroutine used a number of times in the same program, or a library program.



Connector symbol. When flowlines are broken due to page limitations or for other reasons, this symbol may be used to indicate the separation. For example:



Flowcharts may be as simple or detailed as desired. The programmer may start with a very general flowchart and, as the problem's solution becomes more clear, develop a very detailed chart. The flowchart is a tool for the programmer's benefit, and may be used in whatever manner he sees fit.

REVIEW

1. Flowcharting is a helpful tool in problem _____.

2. Which symbol represents a decision?

3. "Long Shot" Al, the inveterate horse racing enthusiast, came to the track one day with $4.00 and a heart full of hope. Scanning the program for the first race, he came to the following plan: "If Orphan Sandy is at 10-to-1 or better, I'll put $2.00 on her to win, and $2.00 on Shotgun to place. If less than 10-to-1, I'll sink the whole $4.00 on Orphan Sandy to win. If I lose the first race, I'll walk home. If I make more than $2.00 and less than $4.00, I'll flip a coin to see if I stay and bet again or go home. Heads, I stay; tails, I go home. If I come out ahead, I'll definitely stay and bet some more."

   Draw a flowchart of the above plan.

4. Devise a flowchart illustrating the process of testing 200 quantities stored on a paper tape device for being positive, negative, or zero. A count is to be kept of the number of quantities in each group and this count is to be printed on a teleprinter.

INSTRUCTION FORMAT 7

Source language instructions are recognized by the Assembler in a certain format. An instruction may be specified in as many as four fields, in the following order -- the Label, Op Code, and Operand fields described in Chapter 4 and a Remarks field allowing the user to specify explanatory comments if he wishes.

Fields are separated by one or more spaces; the statement is terminated by an end-of-statement mark. On paper tape, the end-of-statement mark consists of a carriage return, (CR) , and a line feed, (LF)

A statement may contain up to 80 characters including blanks, but excluding the end-of-statement mark. Fields beginning in character positions 73-80 are not processed by the Assembler.

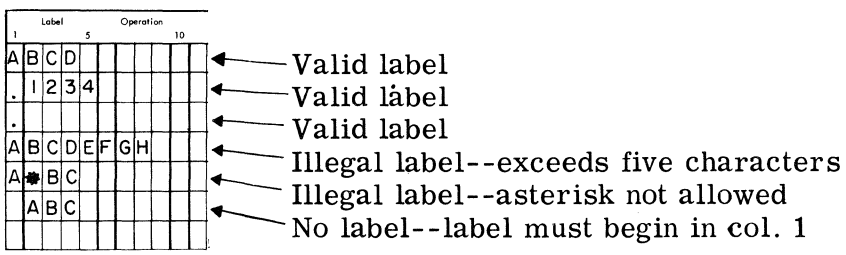
7.1 LABEL FIELD

This field begins in character position one, immediately following the end-of-statement mark for the previous statement. A space in character position one indicates the statement has no label.

7.1.1 LABEL SYMBOL

A label symbol may be constructed of from one to five alphanumeric characters, A through Z, 0 through 9, and the period. The first character of a label must be alphabetic or a period.

Examples:



Each label must be unique within the program; no two statements may have the same label

7.1.2 ASTERISK

An asterisk in character position one indicates that the entire statement is a comment. Positions 2 through 80 are then available for programmer's remarks. Only positions 1 through 68 are printed as part of the assembly listing on the HP 2752A Teleprinter, however. An asterisk within the label field is illegal in any character position other than one.

Comments are not translated by the Assembler as part of the object program; that is, they do not take up memory space at execution time.

Example:

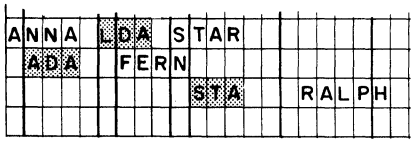
```
Label     Operation     Operand                                   Comments
1    5    10    15    20    25    30    35    40    45
*  THIS  IS  A  COMMENT--NOT  AN  INSTRUCTION.
*  EACH  COMMENT  LINE  MUST  HAVE  AN  *  IN  COL.  1
```

7.2 OP CODE FIELD

The operation code field follows the label field and is separated from it by at least one space. If there is no label, the operation code may begin anywhere after character position one. The op code field is terminated by a space immediately following an operation code. Specific operation codes are discussed in Chapters 8 and 9.

Examples:

```
Label     Operation     Operand
1    5    10    15    20
ANNA      LDA   STAR
          ADA   FERN
          STA   RALPH
```



Both of these sequences of code would be translated and executed correctly. However, the first would be much easier to read, both on the coding sheet and on the assembly listing.

7.3 OPERAND FIELD

The operand field follows the op code field and is separated from it by at least one space. It is terminated by a space (except when the space follows a plus sign, a minus sign, a comma, or a left parenthesis) or by an end-of-statement mark if the remarks field is omitted.

The operand field may contain an expression or a literal; an expression consists of one of the following:

1) symbolic term

2) numeric term

3) asterisk

4) combination of symbolic terms, numeric terms, or asterisk, joined by arithmetic operators + and -.

An operand expression may be absolute or relocatable. In an absolute program all operand expressions are considered absolute. A relocatable program may contain absolute or relocatable operand expressions; however, the absolute address expressions must have a value less than 77_8.

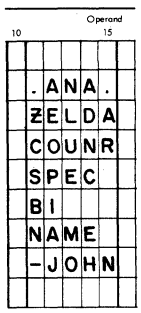
In some cases, an expression may be followed by an indicator. For example, the operand field of Memory Reference instructions may be followed by a , I to specify indirect addressing. These indicators are discussed with the instructions with which they may be used.

7.3.1 SYMBOLIC TERM

A symbolic term is constructed using the same rules as for a label -- one to five characters in length, consisting of A through Z, 0 through 9, and the period. The first character must be alphabetic or a period.

A symbol used in the operand field must be defined elsewhere in the program as a label of a machine instruction or a BSS, ASC, DEC, OCT, DEF, ABS, EQU, or arithmetic subroutine pseudo instruction. In the special case of the COM and EXT pseudo instructions, an operand term defines a symbol which may be used as an operand term in other instructions.

A symbolic term may be preceded by a plus or minus sign. If preceded by a plus or no sign, the associated value is used. If preceded by a minus sign, the two's complement of the associated value is used. A single negative term may be used only with the ABS pseudo operation.



Valid symbolic terms, provided they have been defined.

Valid only with ABS pseudo, provided John has been defined.

Absolute or Relocatable Symbolic Terms

A symbolic term may be absolute or relocatable. If the program is defined as absolute, or if a symbol has been defined as absolute in an EQU pseudo instruction, the value assigned to the symbol by the Assembler remains fixed; the term is absolute. If the program is defined as relocatable, the actual value of the symbol is established on loading; the term is relocatable.

A relocatable term may be program relocatable, base page relocatable, or common relocatable. A symbol that names an area of common storage (via the COM pseudo) is a common relocatable term. A symbol that is allocated to the base page (via the ORB pseudo) is a base page relocatable term. A symbol that is defined in any other manner is a program relocatable term.

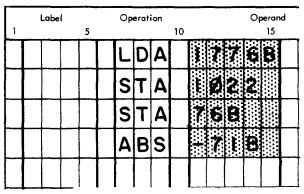
7.3.2 NUMERIC TERM

A numeric term may be decimal or octal. In an absolute program, the maximum value of a single numeric operand depends on the type of machine or pseudo instruction:

Pseudo instructions	$32{,}767_{10}$	or	177777_8
Memory Reference instructions	1023_{10}	or	1777_8
Input/Output instructions	63_{10}	or	77_8

If a numeric term is preceded by a plus or no sign, the binary equivalent of the number is used in the object code. If preceded by a minus sign, the two's complement of the binary equivalent is used. A single negative numeric term may only be used with the ABS pseudo operation. An octal number is followed by the letter B; for example, 377B -177777B.

Examples:

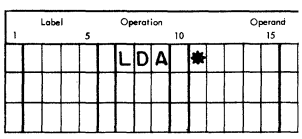


Valid for absolute program
Valid for absolute program
Valid for absolute or relocatable program
Valid for ABS pseudo instruction

7.3.3 ASTERISK

An asterisk in the operand field refers to the value in the program location counter (or base page location counter) at the time the source program statement is encountered. The asterisk is assigned a relocatable value in a relocatable program, an absolute value in an absolute program.

Example:



When this instruction is executed, the A-register will be loaded with the translated binary representation of the instruction itself.

7.3.4 COMBINATION EXPRESSION

Numeric terms, symbolic terms, and the asterisk may be combined using the arithmetic operators + and - to form operands. These expressions are either absolute or relocatable depending upon the manner in which their absolute and/or relocatable terms are combined.

Decimal and octal integers, and symbols defined as being absolute in an EQU pseudo instruction are absolute terms.

The asterisk and all symbols that are defined in the program are assigned relocatable or absolute values, depending on the type of assembly.

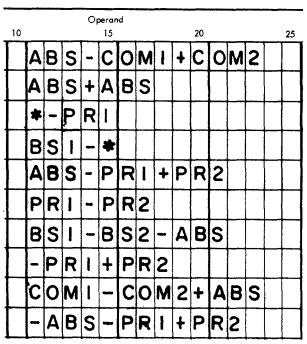
Absolute Combinations

An absolute combination may be any arithmetic combination of absolute terms. It may also contain relocatable terms alone or in combination with absolute terms. If relocatable terms do appear, there must be an even number of them; they must be of the same type (program, common, or base page relocatable), and they must be paired by sign (a negative term for each positive term). The paired terms do not have to be contiguous, that is, next to each other in the combination. The pairing of terms by type cancels the effect of relocation; the value represented by the pair remains constant.

An absolute expression reduces to a single absolute value. The value of an absolute combination may be negative only for ABS pseudo operations.

Examples:

If PR1 and PR2 are program relocatable terms; BS1 and BS2, base page relocatable; COM1 and COM2, common relocatable; and ABS an absolute term, then the following are absolute terms:



The asterisk is base page relocatable or program relocatable depending on the location of the instruction.

Relocatable Combinations

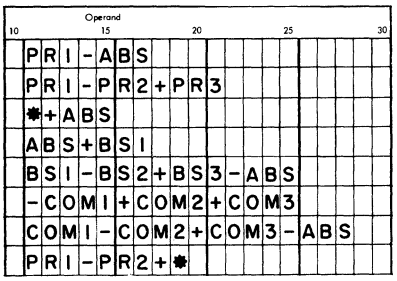
A relocatable combination is one whose value is changed by the loader. All relocatable combinations must have a positive value.

A relocatable expression may contain any odd number of relocatable terms, alone, or in combination with absolute terms. All relocatable terms must be of the same type. Terms must be paired by sign with the odd term being positive.

A relocatable combination reduces to a single positive relocatable value, adjusted by the values represented by the absolute terms and paired relocatable terms associated with it.

Examples:

If PR1, PR2, and PR3 are program relocatable terms; BS1, BS2, and BS3, base page relocatable terms; COM1, COM2, and COM3, common relocatable; and ABS an absolute term, then the following are relocatable terms:



7.3.5 LITERALS

Literals provide a simplified means of defining constants in the source program. They are processed by the Assembler provided for 8K or larger machines (8,192-word memory or larger); literals may be used only in relocatable programs.

Literals may be used in the operand field of certain instructions to specify an actual operand value, rather than an operand address. The literal values specified in the source program are preceded by an equal sign and an identifier. The equal sign signifies that the value is a literal, and not an address expression; the identifier defines the type of literal:

=D one-word decimal integer within the range -32,767 through 32,767.

=F two-word floating point decimal number. Any positive or negative real number within the approximate range 10^{-38} to 10^{38}, and zero. Decimal numbers with fractional values must be specified with a decimal point (32.75). Decimal numbers without fractional values may be specified with or without a decimal point (32 or 32.). (Section 9.4.3 describes the manner in which floating point numbers are stored in memory).

=B octal integer; a signed or unsigned number consisting of one to six octal digits $b_1b_2b_3b_4b_5b_6$ where b_1 may be 0 or 1, b_2-b_6 may be 0-7.

=A two ASCII characters; blank fill is used for a Ø or if only one letter follows the A.

=L an expression which, when evaluated, will result in an absolute value. All symbols used must be previously defined.

The literal value is specified immediately after the identifier; no spaces may intervene.

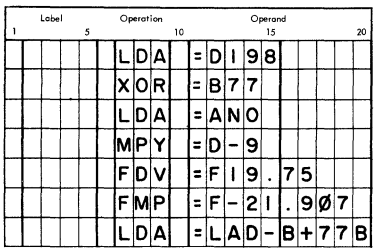
Literals may be used as operands with the following instructions only:

ADA ADB AND MPY LDA LDB XOR DIV CPA CPB IOR	May use = D, = B, = A, and L only
DLD FAD FMP FDV FSB	May use = F only

Only one literal may be specified in an operand field. The Assembler translates the literal into its binary value, assigns the value to a memory location, and translates the instruction so that it refers to the location where the literal value is stored. The Assembler assigns the literals to the memory locations immediately following the last instruction of the program. These locations are printed on the source program listing during pass 2 of the assembly, unless the SUP pseudo instruction is specified to suppress this listing. (See Example below.)

If the same literal is used in more than one instruction, only one value is generated, and all instructions using this literal refer to the same location.

Examples:



A loaded with binary equivalent of 198_{10}.
Logical product of (A) and 000077_8.
A loaded with ASCII characters NO.
(A) times -9_{10}.
(AB) divided by 19.75_{10}.
(AB) multiplied by -21.907_{10}.
A loaded with the result of value of AD - value of B + 77_8.

The listing segment shown below was produced from a source program using the following literals:

```
0002   00000                      NAM  START
0003   00036              A       EQU  30
0004   00050              B       EQU  40
0005   00000  000000      LOC     BSS  12
0006   00014  000000      START   NOP
0007   00015  062012R             LDA  =D20
0008   00016  072000R             STA  LOC
0009   00017  062013R             LDA  =B75
0010   00020  072001R             STA  LOC+1
0011   00021  016001X             DLD  =F10.0
       00022  000014R
0012   00023  016002X             DST  LOC+2
       00024  000002R
0013   00025  016001X             DLD  =F-10.0
       00026  000016R
0014   00027  016002X             DST  LOC+4
       00030  000004R
```

```
0015   00031 016001X        DLD =F3.7
       00032 000020R
0016   00033 016002X        DST LOC+6
       00034 000006R
0017   00035 062022R        LDA =LA-B+100
0018   00036 072010R        STA LOC+8
0019   00037 062023R        LDA =APA
0020   00040 072011R        STA LOC+9
0021   00041 062023R        LDA =B50101
0022   00042 072012R        STA LOC+10
0023   00043 062023R        LDA =D20545
0024   00044 072013R        STA LOC+11
0025   00045 126014R        JMP START,I  <-- Last instruction requiring memory space
       00046 000024  <-- Octal representation of 20(10).
       00047 000075  <-- Octal representation of 75(8).
       00050 050000 }
       00051 000010 }  <-- 10.1 in floating point format
       00052 130000 }
       00053 000010 }  <-- -10.1 in floating point format.
       00054 073146 }
       00055 063004 }  <-- 3.7 in floating point format.
       00056 000132  <-- Octal value of expression: A - B + 100.
       00057 050101  <-- Octal value of PA, 50101(8) and 20545(10).
0026                        END
** NO ERRORS*
```

7.4 COMMENTS FIELD

The comments field allows the user to transcribe comments on the list output produced by the Assembler. The field follows the Operand field and is separated from it by at least one space. The end-of statement mark, (CR) (LF) , or the 80th character in the entire statement terminates the field. If the listing is to be produced on the 2752A Teleprinter, the total statement length, excluding the end-of-statement mark, should not exceed 52 characters, the width of the source language portion of the listing. Statements consisting solely of comments may contain up to 68 characters including the asterisk in the first position. On the list output, statements consisting entirely of comments begin 16 positions to the left of the source portion of statements containing instructions.

The comments field should be omitted on the NAM and END pseudo operations or in the following input/output statements without operands; SOC, SOS, and HLT. If comments are used the Assembler attempts to interpret them as an operand.

7.5 MANUAL NOTATION

Notation used in this manual to represent source language instructions are as follows:

Symbols expressed in lower case are to be supplied by the user. For example,

m	memory address -- an expression (Section 7.3).
sc	select code -- an expression
lname	a label symbol

Bracket [ ] indicate a field or portion of a field that is optional.

Braces { } indicate that one of the included set may be selected.

7.6 CODING CONVENTIONS

To ensure maximum legibility, most programmers code source programs in capital letters on coding sheets provided for this purpose. To distinguish between certain characters, the following conventions have been established:

alphabetic	I_____ı
numeric	1_____ı
alphabetic	O_____o
numeric	zero___ø
alphabetic	Z_____ƶ
numeric	2_____ƨ

REVIEW

1. What are the four fields of an instruction, in the order they are specified?

2. What is the permissible length of an operand symbolic term?

3. What characters may be used to construct a valid label?

4. How do you specify an unlabeled instruction?

5. What is the method of coding a statement consisting entirely of comments?

6. Which of the following labels are illegal?

   (a) ABC
   (b) A.B.C.
   (c) 2AB
   (d) .BC2
   (e) BA*
   (f) ROTATE

7. What characters terminate a statement?

8. What is the range for numeric operand terms in the following instructions?

   (a) Pseudo instructions
   (b) Memory reference instructions
   (c) Input/Output instructions

9. What character may be appended to a numeric operand term to distinguish the term as octal?

10. What is the function of the asterisk in the operand field?

11. Which of the following literals are illegal?

(a)	=B927	(e)	=L77
(b)	=D926	(f)	=F32
(c)	=D9.35	(g)	=B777
(d)	=AAA	(h)	=329

MACHINE INSTRUCTIONS 8

These instructions are the machine's instruction repertoire; the assembler translates mnemonic labels, operation codes, and operands to the instruction format as described in Chapter 3.

8.1 MEMORY REFERENCE

Memory reference instructions are those which refer to locations in memory. They include instructions to perform arithmetic and logical operations and instructions which alter the sequence of execution.

8.1.1 LDA/LDB

These instructions load the A or B register with the contents of the specified address, or with the specified literal. The contents of the address are unchanged.

Label	Op Code	Operand
	{LDA / LDB}	{m [,I] / lit}

m — absolute or relative address expression

I — indirect addressing indicator

lit — literal value

Examples:

	Before Execution	After Execution
LDA ALFRE	(A)=012312_8	(A)=001767_8
	(ALFRE) =001767_8	(ALFRE) =001767_8
LDB ALFRE, I	(B)=076543_8	(B)=155555_8
	(ALFRE) =002316_8	(ALFRE) =002316_8
	(2316_8)=155555_8	(2316_8)=155555_8
LDA =77B	(A)=015762_8	(A)=000077_8

8.1.2 STA/STB

These instructions store the contents of the A or B register in the specified address.

Label	Op Code	Operand
	{ STA / STB }	m[,I]

m absolute or relative address

I indirect addressing indicator

Examples:

	Before Execution	After Execution
STA PLACE	(A)=017653_8	(A)=017653_8
	(PLACE)=054327_8	(PLACE)=017653_8
STA PLACE,I	(A)=153455_8	(A)=153455_8
	(PLACE)=100677_8	(PLACE)=100677_8
	(677_8)=000534_8	(677_8)=000534_8
	(534_8)=177777_8	(534_8)=153455_8

The 1 in bit 15 of location PLACE means that the contents of the location specified by the right-most 15 bits is to be used as an indirect address.

8.1.3 ADA/ADB

Adds the contents of the A or B register to the contents of the specified address or to the literal, storing the result in the A or B register. The contents of the address are unchanged.

Label	Op Code	Operand
	{ ADA / ADB }	{ m[,I] / lit }

m absolute or relative address expression or literal

I indirect address indicator

lit literal value

Examples:

	Before Execution	After Execution
ADA EGAD	(A)=001702_8	(A)=001727_8
	(EGAD)=000025_8	(EGAD)=000025_8
ADA EGAD,I	(A)=001702_8	(A)=005431_8
	(EGAD)=001652_8	(EGAD)=001652_8
	(1652_8)=003527_8	(1652_8)=003527_8
ADA =D16	(A)=005320_8	(A)=005340_8

8.1.4 AND

Forms the logical product of the contents of the A-register and the specified address or literal and stores the result in the A-register. The logical product of two bits is defined as follows:

$0 \wedge 0 = 0$
$0 \wedge 1 = 0$
$1 \wedge 0 = 0$
$1 \wedge 1 = 1$

Label	Op Code	Operand
	AND	$\left\{\begin{matrix} m[,I] \\ lit \end{matrix}\right\}$

m	absolute or relative address
I	indirect address indicator
lit	literal value

Examples:

	Before Execution	After Execution
AND MASK	(A)=037654_8	(A)=000654_8
	(MASK)=000777_8	(MASK)=000777_8
AND MASK,I	(A)=037654_8	(A)=037400_8
	(MASK)=000777_8	(MASK)=000777_8
	(777)=177500_8	(777)=177500_8
AND =77B	(A)=053461_8	(A)=000061_8

8.1.5 XOR

Forms the logical "exclusive or" of the contents of the A register and the specified address or literal and stores the result in the A register. The "exclusive or" operation for two bits is defined as follows:

$0 \veebar 0 = 0$
$0 \veebar 1 = 1$
$1 \veebar 0 = 1$
$1 \veebar 1 = 0$

Label	Op Code	Operand
	XOR	{ m[,I] / lit }

m	absolute or relative address
I	indirect address indicator
lit	literal value

Examples:

	Before Execution	After Execution
XOR ZELDA	(A)=	(A)=
	$0\ 011\ 010\ 110\ 001\ 111_2$	$1\ 100\ 101\ 001\ 110\ 000_2$
	(ZELDA)=	(ZELDA)=
	$1\ 111\ 111\ 111\ 111\ 111_2$	$1\ 111\ 111\ 111\ 111\ 111_2$

Note that by taking the exclusive or with a mask of all 1's, the 1's complement is formed.

XOR SCOTT,I	(A)=	(A)=
	$0\ 011\ 010\ 110\ 001\ 111_2$	$0\ 110\ 000\ 011\ 011\ 010_2$
	(SCOTT)=	(SCOTT)=
	$0\ 000\ 001\ 010\ 011\ 111_2$	$0\ 000\ 001\ 010\ 011\ 111_2$
	(1237_8)=	(1237_8)=
	$0\ 101\ 010\ 101\ 010\ 101_2$	$0\ 101\ 010\ 101\ 010\ 101_2$
XOR =777B	(A)=	
	$0\ 010\ 011\ 100\ 101\ 110_2$	$0\ 010\ 011\ 011\ 010\ 001_2$

8.1.6 IOR

Forms the logical "inclusive or" of the contents of the A register and the specified memory location and stores the result in the A register. The "inclusive or" operation for two bits is defined as follows:

$0 \vee 0 = 0$
$0 \vee 1 = 1$
$1 \vee 0 = 1$
$1 \vee 1 = 1$

Label	Op Code	Operand
	IOR	$\left\{ \begin{matrix} m[,I] \\ lit \end{matrix} \right\}$

m	absolute or relative address
I	indirect address indicator
lit	literal value

Examples:

	Before Execution	After Execution
IOR CODE	(A)=	(A)=
	$0\ 000\ 000\ 111\ 101\ 111_2$	$1\ 111\ 111\ 111\ 101\ 111_2$
	(CODE)=	
	$1\ 111\ 111\ 111\ 000\ 000_2$	$1\ 111\ 111\ 111\ 000\ 000_2$
IOR CODE,I	(A)=	(A)=
	$0\ 000\ 000\ 111\ 101\ 111_2$	$0\ 000\ 000\ 111\ 101\ 111_2$
	(CODE)=	(CODE)=
	$0\ 000\ 001\ 110\ 101\ 000_2$	$0\ 000\ 001\ 110\ 101\ 000_2$
	(1650_8)=	(1650_8)=
	$0\ 000\ 000\ 000\ 001\ 111_2$	$0\ 000\ 000\ 000\ 001\ 111_2$
IOR =177777B	(A)=	(A)=
	$0\ 101\ 110\ 111\ 100\ 011_2$	$1\ 111\ 111\ 111\ 111\ 111_2$

8.1.7 JMP

Alters sequence of execution. The next instruction to be executed is located at the specified memory location.

Label	Op Code	Operand
	JMP	m[,I]

m — absolute or relative address expression specifying next statement to be executed.

I — indirect address indicator; if specified, the contents of memory location m are used as the address containing the next statement to be executed.

Examples:

```
Label  Operation  Operand       Comments
       IOR  GLAD                NOTE THAT THIS SEQUENCE OF
LOOP   LDA  FERN                INSTRUCTIONS RESULTS IN A NEVER-
       STA  FERN+1              ENDING LOOP--THE COMPUTER WILL
       AND  MASK                KEEP EXECUTING THE STATEMENTS
       STA  FERN+2              FROM LOOP TO OOPS INDEFINITELY
OOPS   JMP  LOOP
        .
        .
        .
       LDA  FERN                THE JUMP IS TO THE LOCATION
       ADA  ADRM                SPECIFIED BY THE CONTENTS OF
       STA  FERN                LOCATION FERN WHICH HAS BEEN
       JMP  FERN,I              MODIFIED BY THE CONTENTS OF
                                LOCATION ADRM
```

8.1.8 JSB

The jump subroutine instruction generates a return address by adding 1 to the contents of the program location counter. This address is stored in the specified address and control transfers to the specified address+1. The user returns control to the main program by a JMP indirect to the first location of the subroutine.

Label	Op Code	Operand
	JSB	m[,I]

m absolute or relative address

I indirect addressing indicator

This instruction is particularly useful for a utility subroutine which may be called at several points in the main program.

Example:

Main Program

```
Label      Operation      Operand
                  .
                  .
                  .
           LDA    BRUCE
           ADA    PLACE
           STA    BRUCE
           JSB    AGHA
           LDA    FREEN
           XOR    EXCL
                  .
                  .
                  .
           JSB    AGHA
           LDA    FISBY
```

Subroutine

```
Label      Operation      Operand
AGHA       LDA    NONE
           LDA    ADDR
           AND    MASK
           STA    PLACE
           JMP    AGHA,I
```

The first time the JSB to AGHA is executed, the address for the LDA FREEN instruction is placed in AGHA and execution transfers to location AGHA+1, the LDA ADDR instruction. The LDA NONE instruction is never executed; it is destroyed when the return address is inserted in that location. The JMP AGHA, I at the end of the subroutine transfers control back to the main program at the LDA FREEN instruction. The next time AGHA is called, the address for the LDA FISBY instruction is placed in AGHA and execution transfers to location AGHA+1. The JMP AGHA, I then transfers control back to the main program at the LDA FISBY instruction.

8.1.9

ISZ

Increment and skip if zero. ISZ adds one to the contents of the specified address. If this quantity is then zero, the next instruction in memory is bypassed.

Label	Op Code	Operand
	ISZ	m[,I]

m	absolute or relative address
I	indirect addressing indicator

This instruction is particularly useful in executing a loop a specific number of times before continuing with processing.

Example:

```
Label    Operation    Operand                    Comments
1     5       10       15       20       25       30       35       40       45       50
       .
       .
       .
EMIL   LDA  QUAN        COUNT HAS BEEN PREVIOUSLY
       ADA  SUM         DEFINED TO CONTAIN -7. THE ISZ
       .                INSTRUCTION ADDS 1 TO COUNT
       .                DURING EACH PASS THROUGH THE
       .                LOOP. WHEN THE SEQUENCE HAS
       ISZ  COUNT       BEEN EXECUTED A TOTAL OF SEVEN
       JMP  EMIL        TIMES, COUNT=Ø AND THE JMP EMIL
       LDB  RALPH       INSTRUCTION IS BYPASSED--CONTROL
       STB  SAM         PASSES TO LDB RALPH.
```

8.1.10

CPA/CPB

This instruction compares the contents of the A- or B-register and the contents of the specified address or the literal. If they are not equal, the next instruction is skipped. If they are equal, the next instruction is executed.

Label	Op Code	Operand
	CPA	m[,I]
	CPB	lit

m	absolute or relative address
I	indirect addressing indicator
lit	literal value

Examples:

```
Label   Operation  Operand       Comments
        .
        .
        .
        LDA  BUD       THE CONTENTS OF LOCATION BUD ARE
        CPA  COUNT     COMPARED WITH THE CONTENTS OF
        JMP  ZELLA     LOCATION COUNT. IF THEY ARE
        ADA  ONE       EQUAL, THE JMP ZELLA INSTRUCTION
        STA  BUD       IS EXECUTED. IF THEY ARE NOT
        .              EQUAL, JMP ZELLA IS NOT EXECUTED
        .              --EXECUTION CONTINUES WITH
        .              ADA ONE.
        .
        LCB  HELEN     THE CONTENTS OF LOCATION HELEN
        CPB  =D17      ARE COMPARED TO 17. IF EQUAL,
        JMP  TOM       THE JMP TOM INSTRUCTION IS EXEC-
        ADB  =D1       UTED. IF UNEQUAL, EXECUTION
        STB  HELEN     CONTINUES WITH ADA =D1.
        JMP  *-1Ø
```

8.2 REGISTER REFERENCE

The register reference instructions manipulate the working registers A, B, and E. They include a Shift-Rotate group and an Alter-Skip group.

8.2.1 SHIFT-ROTATE GROUP

This group contains 19 basic instructions that can be combined to produce more than 500 different single cycle operations.

CLE — Clear E-register (set to zero).

ALS/BLS — Shift A- or B-register left one bit, place a zero in the least significant bit (bit 0). Sign bit (bit 15) is unaltered.

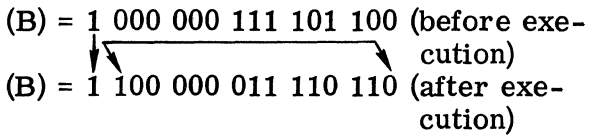
Example, BLS:

(B) = 1 011 100 010 110 100 (before execution)

(B) = 1 111 000 101 101 000 (after execution)

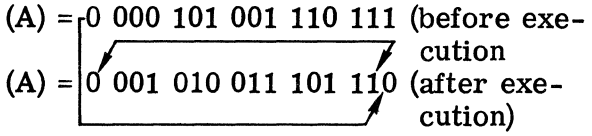
ARS/BRS Shift A- or B-register right one bit, extend sign bit. Sign bit is unaltered:

Example, BRS:



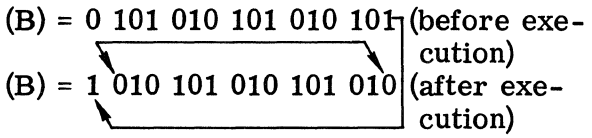
RAL/RBL Rotate A- or B-register left one bit.

Example, RAL:



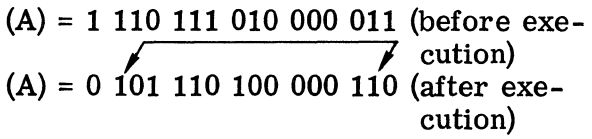
RAR/RBR Rotate A- or B-register right one bit.

Example, RBR:



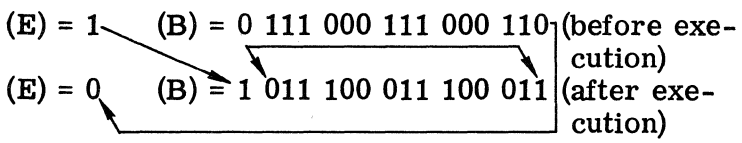
ALR/BLR Shift A- or B-register left one bit, clear the sign bit, place a zero in the least significant bit.

Example, ALR:



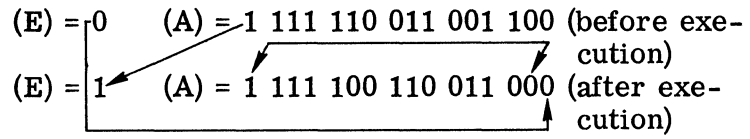
ERA/ERB Rotate E and A- or B-register right one bit.

Example, ERB:



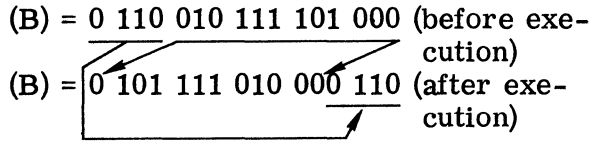
ELA/ELB Rotate E and A- or B-register left one bit.

Example, ELA:



ALF/BLF Rotate A- or B-register left four bits.

Example, BLF:



SLA/SLB Skip the next instruction if the least significant bit of the A- or B-register is zero.

When combined, the instructions must be given in the order shown below, separated by commas. Instructions for the A-register may not be combined with instructions for the B-register.

Label	Op Code	Operand

[{ ALS, ARS, RAL, RAR, ALR, ALF, ERA, ELA }] [,CLE] [,SLA] [, { ALS, ARS, RAL, RAR, ALR, ALF, ERA, ELA }]

[{ BLS, BRS, RBL, RBR, BLR, BLF, ERB, ELB }] [,CLE] [,SLB] [, { BLS, BRS, RBL, RBR, BLR, BLF, ERB, ELB }]

Any combination as shown above requires only one machine cycle for execution. For example:

ARS

ARS (coded on two lines) requires two machine cycles,

but

ARS, ARS (coded on one line) requires one machine cycle.

Examples:

```
Label   Operation   Operand            Comments
       LDA ALICE         IF ALICE CONTAINS A Ø IN THE
       SLA               LEAST SIGNIFICANT BIT,THE
       LDB ANGLE         LDB ANGLE INSTRUCTION IS
       STA ANGLE         SKIPPED.
        .
        .
        .                ROTATE QUANTITY IN A LEFT ONE
       RAL,SLA           BIT AND TEST BIT Ø. IF NON-ZERO,
       JMP NEGTV         THE ORIGINAL QUANTITY IN A WAS
       JMP POSTV         NEGATIVE AND JMP NEGTV IS EXEC-
        .                UTED. IF ZERO, THE ORIGINAL
        .                QUANTITY IN A WAS POSITIVE--JMP
        .                NEGTV IS SKIPPED AND JMP POSTV
        .                IS EXECUTED.
        .
       RBR,BLF           ROTATE QUANTITY IN B LEFT 3 BITS
        .
        .
        .
       ERA,CLE,SLA,ELA        SETS BIT Ø OFF AND JUMPS TO
       JMP BION               LOCATION BION IF BIT 1=1--
       JMP BIOFF              TO LOCATION BIOFF IF
                              BIT 1=Ø.
```

8.2.2 ALTER-SKIP GROUP

This group contains 19 basic instructions that can be combined to produce more than 700 different single cycle operations.

CLA/CLB Clear the A- or B-register to zeros.

Example, CLA:

(A) = 0 010 111 001 110 101 (before execution)

(A) = 0 000 000 000 000 000 (after execution)

CMA/CMB Complement the contents of the A- or B-register, one's complement form.

Example, CMB:

(B) = 0 000 010 101 111 000 (before execution)
(B) = 1 111 101 010 000 111 (after execution)

CCA/CCB Clear, then one's-complement the A- or B-register (set to one's).

Example, CCA:

(A) = 1 111 000 010 101 011 (before execution)
(A) = 1 111 111 111 111 111 (after execution)

CME Complement the E-register.

CLE Clear the E-register (set to zero).

CCE Clear, then one's complement the E-register.

SEZ Skip next instruction if E is zero.

SSA/SSB Skip next instruction if sign of A- or B-register is positive; that is, if bit 15=0.

INA/INB Increment the contents of the A- or B-register by one.

Example, INB:

(B) = 0 001 101 110 010 111 (before execution)
(B) = 0 001 101 110 011 000 (after execution)

SZA/SZB Skip the next instruction if the contents of the A- or B-register is all zeros.

SLA/SLB Skip the next instruction if the least significant bit (bit 0) of the A- or B-register is zero.

RSS Reverse the sense of the skip instructions preceding the RSS in the statement. That is, the SSA/SSB, SZA/SZB, SLA/SLB instructions skip on 1's when followed in the same statement by an RSS instruction.

When combined, the instructions must be given in the order shown below, separated by commas. Instructions for the A-register may not be combined with instructions for the B-register.

<u>Label</u>   <u>Op Code</u>

```
[{CLA | CMA | CCA}] [,SEZ] [,{CLE | CME | CCE}] [,SSA][,SLA][,INA][,SZA][,RSS]

[{CLB | CMB | CCB}] [,SEZ] [,{CLE | CME | CCE}] [,SSB][,SLB][,INB][,SZB][,RSS]
```

Any combination as shown above requires only one machine cycle for execution. If more than one skip instruction is used in a statement, any true condition will cause the skip to occur. The only exception is SSA, SLA, RSS or SSB, SLB, RSS. The indicated register must contain a quantity which is negative <u>and</u> odd (bit 0=1) for the skip to occur. An RSS following more than one skip instruction in a statement reverses the sense of all the skip instructions.

Examples:

```
Label   Operation  Operand               Comments
        LDA        LANI                  SIGN BIT OF LANI TESTED FOR Ø OR
        SSA                              1. IF Ø, LANI IS POSITVE--JMP
        JMP        NEGPL                 POSPL IS EXECUTED. IF 1, LANI IS
        JMP        POSPL                 NEGATIVE--JMP NEGPL IS EXECUTED

        LDA        DICK                  TESTS (DICK) FOR ODD OR EVEN
        SLA                              (BIT Ø=1 OR Ø). IF ODD, JMP
        JMP        ODRTN                 ODRTN IS EXECUTED. IF EVEN, JMP
        JMP        EVRTN                 EVRTN IS EXECUTED.

        LDA        CAROL                 IF (CAROL) ARE NEGATIVE, A JUMP
        SSA,RSS                          IS MADE TO XYZ. IF POSITIVE, THE
        CMA,INA                          VALUE IS CONVERTED TO NEGATIVE
        JMP        XYZ                   AND A JUMP IS MADE TO XYZ.
```

```
        LDA  ED          COMPARES CONTENTS OF LOCATIONS
        CMA,INA          ED AND QUAN. IF (ED) LESS THAN
        ADA  QUAN        (QUAN), A JUMP IS MADE TO LT.RT
        SSA,RSS          IF (ED) GREATER THAN (QUAN), A
        JMP  LT.RT       JUMP TO GT.RT OCCURS. IF (ED)
        SZA              EQUALS (QUAN), A JUMP TO EQ.RT
        JMP  GR.RT       OCCURS.
        JMP  EQ.RT

        LDB  JOE         TESTS (JOE) FOR NEGATIVE AND
        SSB,SLB,RSS      BIT Ø=1. IF TRUE, CMB,INB IS
        CMB,INB          SKIPPED--EXECUTION CONTINUES
        LDA  FOWL        WITH LDA FOWL. IF FALSE, THE
                         NUMBER IS COMPLEMENTED AND
                         EXECUTION CONTINUES WITH LDA
                         FOWL.
```

8.2.3 NOP

When a no-operation instruction is encountered in a program, no action takes place; the computer goes on to the next instruction. A full memory cycle is used in executing a no-operation instruction.

Label	Op Code	Operand
	NOP	(not used)

A subroutine to be entered by a JSB instruction should have a NOP as the first statement. A NOP statement causes the assembler to generate a word of zeros.

8.3 INPUT/OUTPUT INSTRUCTIONS

The input/output instructions: (1) allow the transfer of data between the computer and an external device, (2) enable or disable external interrupt, (3) check the status of I/O devices, and (4) check for arithmetic overflow condition.

Because of the variety of ordinary input/output devices and specialized HP data acquisition devices which may be attached to the HP 2116A, one particular instruction may have a number of different results.

Very generally speaking, the STC instruction "turns on" the control bit, transferring or enabling the transfer of one data

element. The size and format of an element depends largely on the type of data the device is expected to convey. For example, a digital voltmeter provides a data element in the form of a two-word binary representation of a decimal number, giving the number of volts measured. A teleprinter machine provides an element in the form of a binary representation of an ASCII character. The size and format of a data element for HP devices connectable to the HP 2116A is given in Appendix D, with samples of coding.

The instructions LIA, LIB, MIA, MIB, OTA, and OTB control the transfer of data between the channel buffer and the A- and B-registers. The flag bit is set automatically when data transmission between the device and the channel buffer is completed. Instructions are also available to set, clear, or test the flag bit. If the interrupt system is enabled, and the control bit is set, setting the flag bit causes program interrupt to occur; control transfers to the interrupt location related to the channel. If the interrupt system is disabled, no interrupt can occur; in this case the programmer may use the instructions which test the flag to determine when transfer is completed. The flag bit may be cleared by specifying the two characters ,C following the select code in most I/O instructions.

8.3.1 STC

This instruction transfers or enables the transfer of one data element between the channel buffer and the device.

Label	Op Code	Operand
	STC	sc[,C]

The set control instruction sets the control bit for the channel indicated by the select code (sc). The C option clears the flag bit before any transmission initiated by the STC is completed.

If sc=1, the statement is treated as a NOP (no-operation) instruction.

8.3.2 CLC

The clear control instruction clears (sets to zero) the control bit for the channel specified by the select code (sc), effectively disconnecting the device.

Label	Op Code	Operand
	CLC	sc [, C]

When the control bit is cleared, interrupt on the channel is disabled, although the flag bit may still be set by the device. If sc=0, control bits for all channels are cleared, and all flags are set; all devices are disconnected. If sc=1, this statement is treated as a NOP (no-operation) instruction.

The C option clears the flag bit for the channel.

8.3.3 LIA/LIB

These instructions clear, then load the A- or B-register with the contents of the I/O buffer indicated by sc.

Label	Op Code	Operand
	{LIA / LIB}	sc [, C]

If sc=1, the contents of the Switch Register are loaded into A or B. If C is specified when sc=1, the Overflow bit is cleared after transfer from the switch register is complete. Otherwise, C clears the flag bit for the channel.

8.3.4 MIA/MIB

These instructions merge ("inclusive or") the contents of the I/O buffer indicated by sc into the A- or B-register.

Label	Op Code	Operand
	{MIA / MIB}	sc [, C]

If sc=1, the contents of the Switch Register are merged into A or B. If C is specified when sc=1, the Overflow bit is cleared after the merge is complete. Otherwise, C clears the flag bit for the channel.

8.3.5 OTA/OTB

Label	Op Code	Operand
	{OTA / OTB}	sc [, C]

This instruction causes the contents of the A- or B-register to be output to the I/O buffer indicated by sc. The C option clears the flag bit for the channel.

8.3.6
STF

Label	Op Code	Operand
	STF	sc

The set flag instruction sets the flag bit of the channel indicated by sc. If sc=0, the entire interrupt system is enabled; if sc=1, the overflow bit is set.

8.3.7
CLF

Label	Op Code	Operand
	CLF	sc

The clear flag instruction clears the flag bit of the channel indicated by sc. If sc=0, the entire interrupt system is disabled; if sc=1, the overflow bit is cleared to zero.

8.3.8
SFC

Label	Op Code	Operand
	SFC	sc

The skip if flag clear instruction skips the instruction immediately following if the flag bit for channel sc is zero.

8.3.9
SFS

Label	Op Code	Operand
	SFS	sc

The skip if flag set instruction skips the instruction immediately following if the flag bit for the channel indicated by sc is one.

8.3.10
CLO,STO
SOC,SOS

In addition to using a select code of 1, the overflow bit may be accessed by the following instructions.

Label	Op Code	Operand
	CLO	(not used)

This instruction clears the overflow bit to zero.

Label	Op Code	Operand
	STO	(not used)

This instruction sets the overflow bit to one.

Label	Op Code	Operand
	SOC	[C]

The skip if overflow clear instruction skips the instruction immediately following if the overflow bit is zero. The C option clears the overflow bit after the test is made. If C is not used, comments must be omitted.

Label	Op Code	Operand
	SOS	[C]

The skip if overflow set instruction skips the instruction immediately following if the overflow bit is one. The C option clears the overflow bit after the test is made. If C is not used, comments must be omitted.

8.3.11 HALT

The halt instruction stops computer processing.

Label	Op Code	Operand
	HLT	[sc [, C]]

The computer stops processing and holds the setting of the flag bit for the channel designated by sc. If the C option is specified, the flag bit for the channel is cleared.

The HLT instruction is displayed in the T-register and the P-register indicates the HLT location plus one.

If neither the sc nor the C option is used, the comments must be omitted.

8.4 EXTENDED ARITHMETIC UNIT INSTRUCTIONS

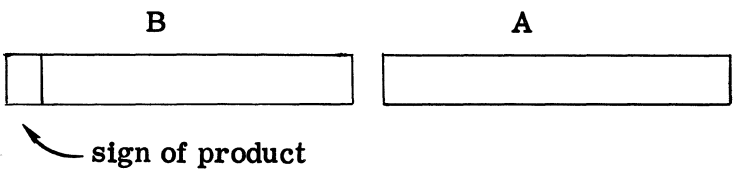
When the Extended Arithmetic Unit option is included in the computer configuration, additional arithmetic and shift capabilities are available. Four of these instructions (MPY, DIV, DLD, and DST) cause two computer words to be generated; the first word is the instruction code, and the second, a 15-bit operand address. When assembled for configurations without the EAU option, these four instructions result in calls to subroutines. The remaining mnemonics, if used in a non-EAU Assembler, would be considered as operation code errors.

8.4.1 MPY

This instruction multiplies the contents of the A-register by the contents of a memory location and stores the product in registers B and A.

Label	Op Code	Operand
	MPY	{ m [,I] } { lit }
	m	absolute or relative address expression
	I	Indirect addressing indicator
	lit	literal value

The result is stored right-justified in the combined B and A registers:



For example:

	Before Execution	After Execution
MPY	(A) = 000173_8	(B) = 000000
	(VALUE) = 000034_8	(A) = 006564_8
	(B) = any quantity	(VALUE) = 000034_8
MPY DANTE	(A) = 101325_8	(B) = 177103_8
	(DANTE) = 061111_8	(A) = 172275_8
	(B) = any value	(DANTE) = 061111_8
MPY =D20	(A) = 000075_8	(B) = 000000
		(A) = 002304

Note that in the second example, the negative answer (in eight's complement form) is really 1774354275. Split into the two 16-bit registers and right-justified, it is represented as shown above.

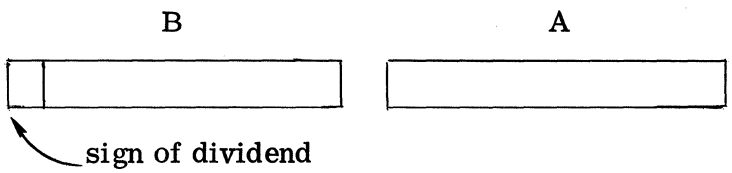
8.4.2 DIV

DIV divides the contents of B and A by the contents of a memory location and stores the result; the quotient is stored in A and the remainder in B.

Label	OP Code	Operand
	DIV	{ m [, I] } { lit }
	m	absolute or relative address.
	I	indirect addressing indicator.
	lit	Literal value

The Overflow bit is set if the divisor equals zero or if the dividend exceeds the A-register, otherwise, exit with Overflow bit cleared.

Initially, the dividend is stored right-justified in the combined B- and A-registers:



For example:

	Before Execution	After Execution
DIV ALAN	(B) = 000000	(B) = 000000
	(A) = 054147_8	(A) = 000563_8
	(ALAN) = 000075_8	(ALAN) = 000075_8
DIV =B73	(B) = 000000	(B) = 000002_8
	(A) = 000075_8	(A) = 000001_8

8.4.3 DLD

DLD loads the A and B registers with the contents of two consecutive words in memory.

Label	Op Code	Operand
	DLD	{ m [, I] / lit }
	m	location of first word---the contents of this location is loaded into the A-register. Location m+1 is loaded into B-register.
	I	indirect addressing indicator
	lit	literal value (F only)

For example:

	Before Execution	After Execution
DLD FLPT	(A) = any quantity	(A) = 017777_8
	(B) = any quantity	(B) = 177400_8
	(FLPT) = 017777_8	(FLPT) = 017777_8
	(FLPT+1) = 177400_8	(FLPT+1) = 177400_8
DLD IND, I	(A) = any quantity	(A) = 035467_8
	(B) = any quantity	(B) = 054100_8
	(IND) = 002177_8	(IND) = 002177_8
	(2177_8) = 035467_8	(2177_8) = 035467_8
	(2200_8) = 054100_8	(2200_8) = 054100_8

8.4.4 DST

DST stores the contents of the A and B registers into two consecutive memory locations.

Label	Op Code	Operand
	DST	m[, I]
	m	location of first word--the contents of the A-register is stored in this location. The contents of the B-register is stored in location m+1.
	I	indirect addressing indicator.

For example:

	Before Execution	After Execution
DST TROUT	(A) = 000042_8	(A) = 000042_8
	(B) = 177401_8	(B) = 177401_8
	(TROUT) = any quantity	(TROUT) = 000042_8
	(TROUT + 1) = any quantity	(TROUT + 1) = 177401_8
DST IVAN, I	(A) = 017532_8	(A) = 017532_8
	(B) = 152525_8 (Note 1 in column 15)	(B) = 152525_8
	(IVAN) = 102027_8	(IVAN) = 102027_8
	(2027_8) = 002777_8	(2027_8) = 002777_8
	(2777_8) = 000000	(2777_8) = 17532_8
	(3000_8) = 017000_8	(3000_8) = 152525_8

8.4.5 SHIFT-ROTATE INSTRUCTIONS

The EAU Shift-Rotate instructions provide the capability to shift or rotate the B- and A-registers 1 to 16 bit positions.

ASR n Arithmetically shift the B- and A-registers right n bits. Sign bit (bit 15 of B) is extended.

Example, ASR 5:

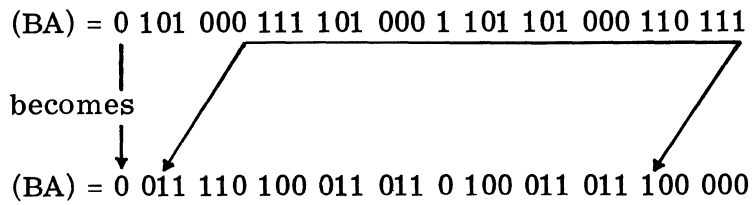
(BA) = 1 011 000 101 000 101 0 101 101 011 100 111

becomes

(BA) = 1 111 110 110 001 010 0 010 101 011 010 111

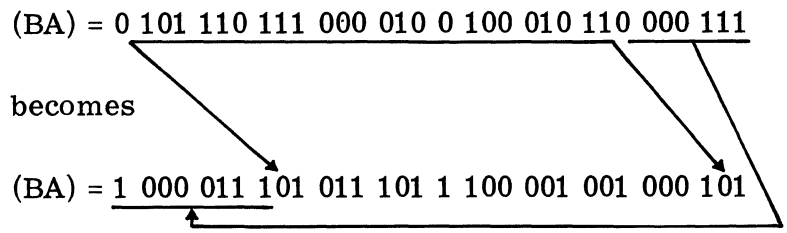
ASL n Arithmetically shift the B- and A-registers left n bits. Place zeros into the least significant bits. The sign bit (bit 15 of B) is unaltered. The Overflow bit is set if bit 14 differs from bit 15 before each shift, otherwise, exit with Overflow bit cleared.

Example, ASL 5:



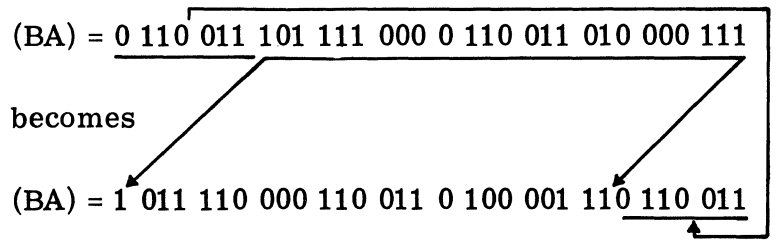
RRR n Rotate the B-and A-registers right n bits.

Example, RRR 8:



RRL n Rotate the B- and A-registers left n bits.

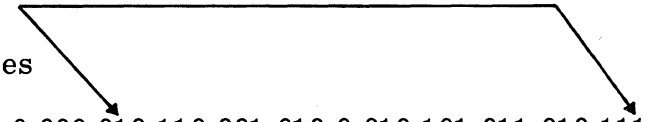
Example, RRL 7:



LSR n Logically shift the B- and A-registers right n bits. Place zeros into the most significant bits.

Example, LSR 5.

(BA) = 1 011 000 101 000 101 0 101 101 011 100 111



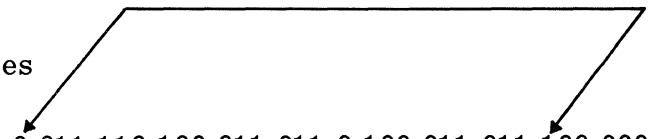
becomes

(BA) = 0 000 010 110 001 010 0 010 101 011 010 111

LSL n Logically shift the B- and A-registers left n bits. Place zeros into the least significant bits.

Example, LSL 5:

(BA) = 0 101 000 111 101 000 1 101 101 000 110 111



becomes

(BA) = 0 011 110 100 011 011 0 100 011 011 100 000

1. The CPA instruction skips on what condition?

2. The ISZ instruction skips on what condition?

3. To what location does the instruction JSB 227B (absolute) pass control?

4. Assume the above instruction is executed at absolute location 137B. What is placed in location 227B?

5. Assume that 50 values are stored in consecutive locations beginning at relative address TAG. What is the relative address of the last value?

6. How many shift-rotate instructions can be combined in one line of coding? Alter-skip instructions?

7. What is wrong with the following combinations?
   (a) ALS, CLE, RBL
   (b) CLA, ALS
   (c) BLS, BLS, CLF
   (d) ALF ALF

8. What is a good first instruction for a routine which is to be entered by a JSB instruction?

9. Give an instruction which enables the interrupt system.

10. When the interrupt system is enabled, the control bit for the device is set, and the device sets its associated flag bit on, what happens?

11. An STC instruction is required to enable the transmission of:
   (a) an element of data
   (b) one computer word
   (c) a character

12. Data transfers between a channel buffer and a device are controlled by which instructions?

13. The instruction HLT 11B, C is stored in absolute location 3767_8. What is displayed in the P-register when this instruction is executed?

14. Write a sequence of code which will add the values stored in CAT and DOG and store the result in SUM.

15. Calculate X = Y + Z and compare X to Q. If unequal, calculate X + W + Q and store the result in R2. If equal, calculate X + W and store in R1.

16. Test bits 3, 5, and 9 of location TEST; if all are on (=1), jump to location ON. If not all are on, jump to location OFF. Define any constants necessary by giving label and value; for example, CONST = 19_{10}.

17. Calculate X - Y. If the result is odd, jump to a subroutine which tests the results for positive or negative. If the result is negative, convert to positive and return to the main program. If the result is positive, return to the main program. If X - Y is even, the program is to continue in sequence.

PSEUDO INSTRUCTIONS 9

Pseudo instructions, as the name implies, are not "real" instructions; they are commands to the Assembler rather than commands to the machine which must be interpreted by the Assembler.

Pseudo instructions may be classed in six general categories according to their capabilities:

Assembler Control

Object Program Linkage

Address and Symbol Definition

Storage Allocation and Constant Definition

Arithmetic Subroutine Calls

Assembly Listing Control

The terms program, subprogram, routine, and subroutine all refer to a set of instructions which are, by themselves, complete. That is, they solve some specific problem or set of problems. The distinction arises through the manner in which these entities relate to the solution of the problem at hand.

With a complex problem, for example, it may be possible to split the problem into separate smaller problems. Each of these may be solved, coded, assembled and tested by a different person or group of persons. Each of these separate problem solutions may be considered a subprogram or subroutine; when combined or linked by pseudo instructions, these form the whole program or routine. Or, there may be a main program/routine which calls various subprograms/subroutines during its execution.

The distinction between these terms is an abstract concept, depending entirely on the way the programmer defines and codes his problem, using pseudo instructions which define program/subprogram boundaries, communication areas and linkage points.

9.1 ASSEMBLER CONTROL

These instructions essentially define a set of instructions as a separate entity, a program. They also provide information to the Assembler about the program being assembled: whether it is absolute or relocatable, for example.

The label field of this class of pseudo instruction is ignored by the Assembler in all cases.

9.1.1 NAM

The NAM pseudo instruction defines a relocatable program.

Op Code	Operand
NAM	[name]
name	One to five alphanumeric characters; the first must be alphabetic or a period. This name is printed on the output listing. If omitted, remarks must be omitted also, or they will be interpreted as the name.

If a program is to be assembled in relocatable form, the NAM statement must immediately follow the ASMB control statement (see Section 11.1). Only statements consisting entirely of comments (* in column 1) and/or an HED pseudo instruction may intervene.

When a NAM instruction is encountered, the program location counter is set to zero. The first instruction requiring memory space following the NAM is assigned relative location zero; the second, relative location one, and so forth.

For example:

```
Label   Operation      Operand                      Comments
1     5      10      15      20   25   30   35   40   45   50
ASMB...
      NAM ERRTN     THE PROGRAM LOCATION COUNTER IS
      LDA EMU       SET TO RELATIVE LOCATION ZERO
      XOR MASKI     FOR THE LDA EMU INSRUCTION I FOR
      STA ABLE      THE XOR MASKI INSTRUCTION, ETC.
       .              . .
       .
       .
```

9.1.2 ORG

ORG defines the origin address of an absolute program, or the address at which portions of absolute or relocatable programs are to begin.

Op Code	Operand
ORG	m

m — When ORG is used to define the beginning of an absolute program, m is a decimal or octal integer specifying the initial setting of the program location counter. When ORG is used to define a beginning address of a portion of a relocatable program, m must be a program relocatable expression; for a portion of an absolute program, any expression. Any symbols used in an expression must be defined in the coding previous to the ORG.

All instructions requiring memory space following an ORG are assigned consecutive addresses starting with the value of the operand. If used to define the origin address of an absolute program, ORG must immediately follow the ASMB control statement (see Section 11.1). Only statements consisting entirely of remarks (* in column 1) and/or an HED pseudo instruction may intervene.

For example:

```
Label      Operation  Operand          Comments
ASMB..     .
           ORG   2000B          THE PROGRAM LOCATION COUNTER IS
           LDA   SAM            SET TO 2000 (OCTAL), IMPLYING
           ADA   CAT            THAT AT THE LDA SAM INSTRUCTION
           .                    IS ASSIGNED ABSOLUTE LOCATION
           .                    2000, ADA CAT TO 2001, ETC.
           .
ASMB..     .
           NAM   PROG1          THE FIRST ADA PLC INSTRUCTION IS
           LDA   Q1             ASSIGNED RELATIVE LOCATION 1
FIRST      ADA   PLC            (OCTAL)--THE LDA AT FOLLOWING
           STA   Q2             THE ORG PSEUDO IS THEN ASSIGNED
           ORG   FIRST+77       RELATIVE LOCATION 100 (OCTAL).
           LDA   A7
```

9.1.3 ORR

ORR resets the program location counter to the value existing when an ORG or ORB instruction was encountered.

Op Code	Operand
ORR	(not used)

More than one ORG or ORB statement may occur before an ORR is used. If so, the program location counter is reset to the value it contained when the first ORG or ORB of the string occurred.

For example:

```
Label  Operation  Operand        Comments
ASMB,...
       NAM   RTN              THE START LDA TOM INSTRUCTION IS
START  LDA   TOM              ASSIGNED RELATIVE LOCATION ZERO.
       ALF                    THE STAR STA QPLC INSTRUCTION IS
       XOR   MASK             ASSIGNED RELATIVE LOCATION 20
       ORG   START+20B        (OCTAL). THE ORR INSTRUCTION
STAR   STA   QPLC             RESETS THE PROGRAM LOCATION
       ADA   RPLC             COUNTER TO THE VALUE IT HAD
       STA   QPLC+1           BEFORE THE ORG--HENCE, LDB FAN
       ORR                    IS ASSIGNED RELATIVE LOCATION 3.
       LDB   FAN
       ADB   DOG
        .
        .
        .
ASMB,...
       NAM   RTN2             THE START LDA RALPH INSTRUCTION
START  LDA   RALPH            IS ASSIGNED RELATIVE LOCATION
STOR   ADA   TOM              ZERO. REL LDB STU IS ASSIGNED
       ORG   STOR+15B         RELATIVE LOCATION 16 (OCTAL),
REL    LDB   STU              AND LDB SAM, RELATIVE LOCATION
       ADB   DAN              33 (OCTAL). THE ORR INSTRUCTION
       ORG   REL+15B          RESETS THE PROGRAM LOCATION
       LDA   SAM              COUNTER TO THE VALUE IT HAD
       ADA   COUNT            BEFORE THE FIRST ORG OCCURRED--
       ORR                    HENCE, STA TOM+1 IS ASSIGNED
       STA   TOM+1            RELATIVE LOCATION 2.
       JMP   REL
        .
        .
        .
```

9.1.4 ORB

ORB permits the assignment of a portion of a relocatable program to the base page.

Op Code	Operand
ORB	(not used)

The ORB statement requires no operand; the assignment of base page locations is made by the Assembler. All statements that follow the ORB statement are assigned contiguous locations in the base page; this assignment terminates when an ORG, ORR, or END statement is encountered.

For example:

```
Label     Operation     Operand                    Comments
1    5    10        15  20        25        30        35        40        45        50
ASMB.,.                 LDA STAR IS ASSIGNED RELATIVE
          NAM  RTN3     LOCATION ZERO. THE ORB CAUSES
          LDA  STAR     THE 1ØØ STORAGE LOCATIONS AND
          ADA  REL      THE DECIMAL 1Ø GENERATED BY THE
          ORB           INPUT BSS 1ØØ AND TEN DEC 1Ø
INPUT     BSS  1ØØ      INSTRUCTIONS TO BE ASSIGNED CON-
TEN       DEC  1Ø       TIGUOUS LOCATIONS IN THE BASE
          ORR           PAGE. WHEN THE ORR IS ENCOUNTER-
          STA  REL+1    ED, THE PROGRAM LOCATION COUNTER
          ALF,ALF       IS SET TO THE VALUE IT HAD BE-
          AND  MASK     FORE THE ORB, AND STA REL+1 IS
          STA  REL+2    ASSIGNED RELATIVE LOCATION 2.
          ORB           THE NEXT ORB CAUSES THE OCTAL 77
MASK      OCT  77       GENERATED BY MASK OCT 77 TO BE
          ORR           STORED AFTER THE DECIMAL 1Ø IN
          LDA  FINE     THE BASE PAGE. THE ORR CAUSES
                        LDA FINE TO BE ASSIGNED RELATIVE
                        LOCATION 6.
```

9.1.5 END

END signifies the end of source language coding; the Assembler terminates each translation pass for a program upon encountering this instruction.

Op Code	Operand
END	[m]

m — name appearing as a statement label in current program. If specified, it identifies the location to which the BCS loader transfers

control after a relocatable program is loaded. A NOP should be stored in this location as control is transferred to this location by the loader with a JSB instruction.

If the operand field is blank, the remarks field must be blank also; otherwise, the Assembler attempts to interpret the first five characters of the remarks as the transfer address symbol.

For example:

```
ASMB...
      NAM  PROG
VALUE DEC  35
STORE BSS  50
       .
       .
       .
MASK  OCT  177777    LOCATION BEGIN (IDENTIFIED BY
BEGIN NOP            END INSTRUCTION) IS THE POSITION
      LDA  VALUE     AT WHICH MACHINE-EXECUTABLE COD-
      ADA  INP       ING BEGINS. THE PROGRAMMER WOULD
       .             NOT WISH TO TRANSFER CONTROL TO
       .             VALUE, SINCE THE DECIMAL 35 IN
       .             THAT LOCATION IS NOT EXECUTABLE.
      STB  STAR
      JMP  BEGIN,I   THE JMP BEGIN,I INSTRUCTION
      END  BEGIN     RETURNS CONTROL TO THE LOADER.
```

9.1.6 REP

The REP pseudo instruction causes the instruction immediately following the REP to be repeated a specified number of times. REP may be used only when the source program is translated by the Assembler provided for 8K or larger machines (8,192-word memory or larger).

Label	Op Code	Operand	Comments
	REP	n	

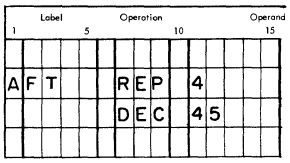
n — any absolute expression, specifying the number of times the instruction following the REP is to be repeated. If symbolic terms are used, they must be defined in the source program previous to the REP.

A label, if used, is assigned to the first repetition of the instruction following the REP. A label should not be specified in the instruction being repeated, since it would not then be unique.

An REP pseudo instruction followed by another REP pseudo instruction is an error; the Assembler issues a diagnostic message and no repetitions occur.

REP may not be used to repeat comment lines.

Example:



would be translated as:

```
AFT   DEC 45
      DEC 45
      DEC 45
      DEC 45
```

9.1.7 IFN/IFZ

The IFN and IFZ pseudo instructions cause the inclusion of instructions in a program provided that either an "N" or "Z", respectively, is specified as a parameter for the ASMB control statement. The IFN or IFZ instruction precedes the set of statements that are to be included. The pseudo instruction XIF serves as a terminator. If XIF is omitted, END acts as a terminator. IFN and IFZ may be used only when the source program is translated by the assembler provided for 8K or larger machines.

Label	Op Code	Operand	Comments
	IFN	All source language statements appearing between the IFN and the XIF pseudo instructions are included in the program if the character "N" is specified on the ASMB control statement.	
	.		
	.		
	.		
	XIF		

IFZ . . . XIF	All source language statements appearing between the IFN and the XIF pseudo instructions are included in the program if the character "Z" is specified on the ASMB control statement.

When the particular letter is not included on the control statement, the related set of statements appears only on the Assembler output listing.

Any number of IFN-XIF and IFZ-XIF sets of coding may appear in a program, however, they may not overlap. An IFZ intervening between an IFN and XIF (or vice versa) results in a diagnostic being issued during compilation; the second pseudo instruction is ignored. When both pseudo instructions are used in the program and both characters are entered on the control statement, the character that appears last determines the set of coding that is to be included in the program; both sets may not be selected in the same assembly.

9.2 OBJECT PROGRAM LINKAGE

These pseudo instructions establish "links", or means of communication, between a main program and its subprograms or between several subprograms which are to be run as a single program.

Labels may be used, but are ignored by the Assembler. The operand field is usually divided into many subfields, separated by commas. The first space not preceded by a comma or left parenthesis terminates the entire field.

9.2.1 COM

The COM pseudo instruction reverses a block of storage locations which may be used by several relocatable subprograms.

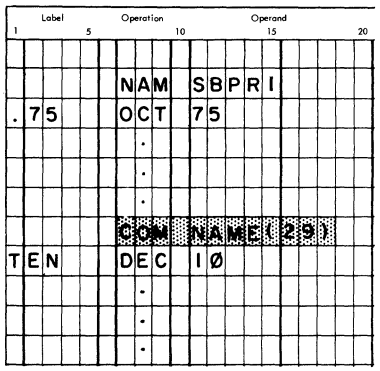
Op Code	Operand
COM	$name_1[(size_1)]$ [, $name_2[(size_2)]$, ..., $name_n[(size_n)]$]

Op Code	Operand
$name_i$	Each name identifies a segment of the block of common storage for the program in which the COM appears. This name may be used in the operand field of the DEF, ABS, EQU pseudo instructions, or any Memory Reference instruction. When used, it refers to the first word of the segment.
$size_i$	A decimal or octal integer specifying the size (in words) of the related name portion of the block. If size is omitted for a name, one word is allocated.

To refer to the common block, other subprograms must also include a COM statement. The segment names and sizes may be the same or they may differ. Regardless of the names and sizes specified in the separate subprograms, there is only one common block for the combined set.

As a simple example, suppose that two subprograms are to use the same data which is read into the computer from an external device. The data consists of names of employees at a company.

These names are read one at a time into a common area. One subprogram refers to the whole name, including the last name, first name, and middle initial. Another subprogram refers to these separately. This could be coded as follows:



```
                    NAM   SBPR2
                    COM   LAST(14),FIRST(14),MIDN
TWEN                DEC   20
MASK                OCT   7777
                      .
                      .
                      .
```

Any number of COM statements may appear in a subprogram. Storage locations are assigned contiguously; the length of the common block is equal to the length of all segments named in all COM statements in the subprogram.

Example:

```
                    NAM   PROG1
                    COM   ADDR1(5),ADDR2(5),ADDR3(5)
                    COM   ADDR4(5)
                      .
                      .
                      .
                    LDA   ADDR2+1       LOADS A WITH 2ND WORD OF SEGMENT
                      .                 ADDR2, 7TH WORD OF COMMON BLOCK.
                      .
                      .
                    END
```

```
                    NAM   PROG2
                    COM   AAA(2),AAB(2),AAD(15)
                      .
                      .
                      .
                    LDA   AAD+2         LOADS A WITH 2ND WORD OF SEGMENT
                      .                 AAD, 7TH WORD OF COMMON BLOCK.
                      .
                      .
                    END
```

Organization of the common block:

PROG1 Segment Name	PROG2 Segment Name	Common Block Location	
ADDR1	AAA	Relative Location	0
			1
	AAB		2
			3
	AAD		4
ADDR2			5
			6
			7
			8
			9
ADDR3			10
			11
			12
			13
			14
ADDR4			15
			16
			17
			18
			19

The first common length declaration processed by the BCS loader establishes the total common storage allocation. Subsequent programs must contain common length declarations which are less than or equal to the length of the first declaration.

The loader also establishes the origin address (common relocation base) of the common block; the origin cannot be set by the ORG or ORB pseudo instruction. All references to the common area are relocatable.

9.2.2 ENT

ENT defines entry points to the program or subprogram.

Op Code	Operand
ENT	$name_1[, name_2, \ldots, name_n]$
name	Each name is a symbol assigned as a label for some instruction in the program.

Entry points allow another subprogram to refer to that specified point in the subprogram. A maximum of 14 entry points may be specified for a subprogram. Symbols appearing in an ENT statement may not also appear in EXT or COM statements in the same subprogram.

9.2.3 EXT

EXT defines external points, labels in other subprograms referenced in this subprogram.

Op Code	Operand
EXT	$name_1$[, $name_2$, . . . , $name_n$]
name	Each name must be defined as an entry point in some other subprogram.

The names defined in the EXT statement may be used in Memory Reference instructions and the EQU and DEF pseudo instructions. An external symbol must appear alone in a Memory Reference instruction; it may not be in a multiple term expression or be specified as indirect. References to external locations are processed as indirect addresses linked through the base page in a manner similar to that described in Section 5.1.2.

Example:

```
Label     Operation  Operand              Comments
       NAM  PROGA
MASK   BSS  1
        .
        .
        .
       ENT  CNST,MASK
CNST   BSS  17
       EXT  JAMAL,START
BEGIN  NOP
       LDA  JAMAL
       STA  MASK      JAMAL AND START ARE REFERRED TO
        .             IN PROGA, BUT ARE ACTUALLY LOCA-
        .             TIONS IN PROGB. HENCE, THEY ARE
        .             DEFINED AS EXTERNALS IN PROGA
       JSB  START     AND ENTRY POINTS IN PROGB.
       LDA  JAMAL
        .
        .
        .
       END  BEGIN
```

```
Label     Operation  Operand                Comments
1    5    10   15   20   25   30   35   40   45   50
          NAM  PROGB
START     NOP
          LDA  CNST
          ADA  MASK
           .
           .                    CNST AND MASK ARE REFERRED TO
           .                    IN PROGB BUT ARE ACTUALLY
          JMP  START,I          LOCATIONS IN PROGA. HENCE
          ENT  START,JAMAL      THEY ARE DEFINED AS EXTERNALS
          EXT  CNST,MASK        IN PROGB AND ENTRY POINTS IN
JAMAL     DEF  START            PROGA.
          END
```

9.3 ADDRESS AND SYMBOL DEFINITION

The pseudo operations in this group assign a value or a word location to a symbol used as an operand elsewhere in the program.

9.3.1 DEF

The address definition (DEF) pseudo instruction provides the means to define a direct or indirect address.

Label	Op Code	Operand
lname	DEF	m[,I]

lname — symbol used as an operand of a Memory Reference instruction using indirect addressing.

m — any address expression valid for type of program being assembled (absolute or relocatable).

I — indirect addressing indicator. Signifies that the address specified by m is used as an indirect address. (For multiple level indirect addressing).

The Assembler generates a 15-bit address pointing to the location specified by m. This address may be referred to in other instructions by lname.

For example:

```
      NAM  PROG
        .
        .
        .
MARGE DEC  1Ø
SALLY DEF  MARGE
        .
        .
        .
      LDA  SALLY,I      THE A-REGISTER IS LOADED WITH
        .               THE DECIMAL 1Ø GENERATED BY THE
        .               MARGE DEC 1Ø INSTRUCTION
        .
      END
```

The m parameter in the JSB statement may be a symbol which appears as an operand in EXT or COM statements in the same program.

For example:

```
      NAM  RTN1
      EXT  SUBR
        .
        .
        .
JMPAD DEF  SUBR
        .
        .
        .
      JSB  JMPAD,I      THE JSB TRANSFERS CONTROL TO
        .               THE SUBR ROUTINE
        .
        .
      END
```

The I option in the DEF statement may be used for multi-level indirect addressing. For example:

```
                    NAM   BOVIN
                     .
                     .
                     .
XSO                  DEF   XSOR,I
XSOR                 DEF   SORT
                     .
                     .
                     .
                     JSB   XSO,I      THE JSB TRANSFERS CONTROL TO
                     .                LOCATION SORT, DEFINED AS AN
                     .                EXTERNAL.
                     .
                     EXT   SORT
                     END
```

The DEF statement allows address modification in relocatable programs. Relocatable programs should not modify memory reference instructions directly, as the example below illustrates.

Incorrect Example:

```
Absolute
Location        Mnemonic Instruction
   .
   .
   .
   77
  100           LINK    DEF     TBL
  101                     .
   .                      .
   .                      .
   .            TBL     BSS     1ØØ
                          .
 3777                     .
---------------------- page boundry ----------------------
 4000           LDTBL   LDA     TBL     LINK, I
 4001                    .
                         .
                         .
                        ISZ     LDTBL
                        JMP     LDTBL
```

(Provided by the BCS Relocatable Loader)

The "LDTBL LDA TBL" and "TBL BSS 1ØØ" instructions are in different pages; therefore, the BCS Relocating Loader provides a 15-bit link address in the base page and modifies the address of the LDTBL instruction to refer to this link address (see arrows). The ISZ instruction, then, erroneously increments the reference to the link address, so that the next time the LDTBL instruction is executed, the A-register is loaded with the contents of the location whose address is contained in absolute location 101.

The following assures correct address modification during program execution:

```
Label     Operation  Operand                         Comments
1    5     10       15     20     25     30     35     40     45     50
      NAM  EXAMP
       .
       .
       .
ITBL  DEF  TBL
TBL   BSS  100          THE ISZ NOW CORRECTLY MODIFIES
       .                THE REFERENCE TO TBL BY
       .                INCREMENTING THE 15-BIT
       .                ADDRESS AT LOCATION ITBL
LDTBL LDA  ITBL,I
       .
       .
       .
      ISZ  ITBL
       .
       .
       .
      END
```

9.3.2 EQU

EQU assigns to a symbol an address value other than the one normally assigned by the program location counter.

Label	Op Code	Operand
lname	EQU	m

lname — symbol which may be used to refer to the value represented by m.

m — any expression, relocatable or absolute; cannot be negative. Must be previously defined in the source program.

EQU may be used to equate two address symbols, such that both symbols refer to the same location, or it may be used to give an absolute address value to a symbol.

For example:

```
Label  Operation  Operand        Comments
       NAM  REMI
        .
        .
        .
TABLE  BSS  1Ø           (DEFINES A 1Ø WORD STORAGE AREA--
        .                TABLE IS ASSOCIATED WITH FIRST
        .                WORD OF THIS STORAGE AREA)
        .
TBL2   EQU  TABLE+5
        .
        .
        .
       LDA  TBL2+3         LOADS A-REGISTER WITH 9TH WORD
        .                  OF TABLE AREA. LDA TABLE+8 WOULD
        .                  PERFORM THE SAME OPERATION
        .
       END
```

```
Label  Operation  Operand        Comments
       NAM  REGIS
        .
        .
        .
A      EQU  Ø           THE SYMBOL A IS EQUATED TO ABSOLUTE
B      EQU  1           LOCATION Ø (LOCATION REFERRING TO
        .               A-REGISTER. SYMBOL B IS EQUATED TO
        .               ABSOLUTE LOCATION 1 (REFERRING TO
        .               THE B-REGISTER).
       LDA  B           THE A-REGISTER IS LOADED WITH THE
        .               CONTENTS OF THE B-REGISTER.
        .
        .
       END
```

9.3.3 ABS

ABS defines a 16-bit absolute value.

Label	Op Code	Operand
lname	ABS	m

lname — A label symbol, if used, refers to the value represented by m.

Label	Op Code	Operand
	m	any absolute expression; if a single symbol is used, it must be defined as absolute elsewhere in the program.

For example:

```
Label     Operation  Operand                    Comments
1    5    10         15        20        25        30        35        40        45        50
          NAM   DAISY
                .
                .
                .
AB        EQU   35              DEFINES SYMBOL AB TO REFER TO
                .               ABSOLUTE LOCATION 35
                .
                .
M35       ABS   -AB             LOCATION M35 CONTAINS -35.
P7Ø       ABS   AB+AB           LOCATION P7Ø CONTAINS 7Ø.
P3Ø       ABS   AB-5            LOCATION P3Ø CONTAINS 3Ø.
                .
                .
                .
          END
```

9.4 STORAGE ALLOCATION AND CONSTANT DEFINITION

These pseudo instructions define blocks of storage locations and constants.

9.4.1 BSS

BSS reserves a block of consecutive memory locations for data storage or for a work area.

Label	Op Code	Operand
lname	BSS	m

lname — A label symbol, if used, refers to the first word of the defined storage area.

m — a positive integer or any expression which results in a positive integer. If an expression is used, the symbols must be previously defined in the program.

The program or base page location counter advances according to the value of the operand. The initial content of the area reserved by the statement is unaltered.

For example:

```
Label     Operation   Operand             Comments
1    5    10    15    20    25    30    35    40    45    50
      NAM EXAM
       .
       .
       .
TAB   BSS 1ØØ       A 1ØØ-WORD STORAGE AREA IS SET
       .            ASIDE. THE FIRST WORD OF THE
       .            STORAGE AREA MAY BE REFERRED TO
       .            BY TAB, THE SECOND BY TAB+1, ETC
      END
```

9.4.2 ASC

ASC generates the binary representation of a string of ASCII (American Standards Code for Information Interchange) characters into consecutive computer words.

Label	Op Code	Operand
lname	ASC	n, ⟨2n characters⟩

lname — A label symbol, if used, refers to the first word of characters generated by the operand.

n — any expression resulting in an unsigned decimal value in the range 1 through 28. Any symbol used must be defined in the coding previous to the ASC.

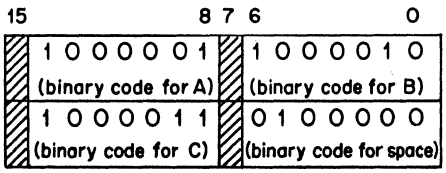
⟨2n characters⟩ — ASCII characters to be generated. Since the binary representation of two ASCII characters may be stored in one computer word, 2×(number of words requested by n) is the number of characters generated. If less than 2n characters are detected before the end-of-statement mark, spaces are filled in the remaining spaces. If more than 2n characters are specified, the excess characters are treated as remarks.

Each character generates seven binary bits; these bits are right-justified in each half of the computer word.

For example:

```
        TTYP    ASC    2,ABC
```

generates the following:



The label TTYP refers to the first word.

The code for the ASCII symbols (CR) (carriage return) and (LF) (line feed) cannot be generated by ASC. The OCT pseudo instruction (Section 8.4.4) must be used.

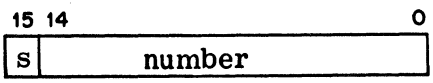
9.4.3 DEC

DEC generates a string of decimal constants into consecutive binary words.

Label	Op Code	Operand
lname	DEC	$d_1[,d_2,\ldots,d_n]$
	lname	A label symbol, if used, refers to the first value generated by the operand.
	d_i	a decimal integer value or floating point expression

INTEGER CONSTANTS

If d_i is a decimal integer, it may be positive, negative, or zero, in the range of 0 to $2^{15}-1$, or $32,767_{10}$. The integer constant is converted into one binary word and appears as follows:



sign bit; 1 implies negative number
(in 2's complement form)

0 implies positive number

Examples:

Instruction	Results
DEC 7,-17,32767	0 000 000 000 000 111 1 111 111 111 101 111 0 111 111 111 111 111
DEC -32767,8	1 000 000 000 000 001 0 000 000 000 001 000 sign bit

FLOATING POINT CONSTANTS

The floating point capability expands the set of numbers which can be expressed from whole numbers in the range $-32767 \leq \text{int} \leq 32767$ to any real number in the approximate range $10^{-38} \leq \text{real} \leq 10^{38}$.

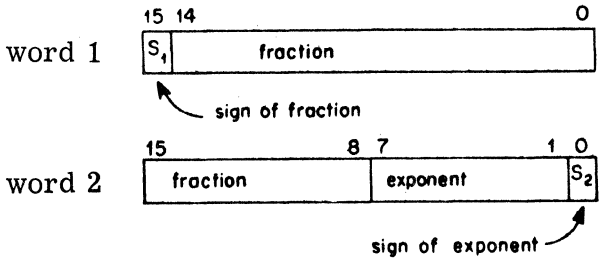
This is accomplished through the conversion of these real numbers to binary floating point format.

These floating point numbers are expressed in the operand field of the DEC pseudo instruction in any of the following forms, where n is any whole number and e is the power of 10 to which the n portion is multiplied.

Form	Examples
±n.n	6.7, -32.691, +91.75, 98.6
±n.	6., -713., +321764., 200.
±.n	.75, +.000001, -.3
±n.nE±e	3.2E2, 517.9E-4, -21.53E-1 (3.2E2 expresses 3.2×10^{2}, or 320; 517.9E-4 expresses 517.9×10^{-4}, or .05179; -21.53E-1 expresses -21.53×10^{-1}, or -2.153)
±.nE±e	.21E3, .100975E-2, -.9E-5 (.21E3 expresses $.21 \times 10^{3}$, or 210; .100975E-2 expresses $.100975 \times 10^{-2}$, or .00100975; -.9E-5 expresses $-.9 \times 10^{-5}$, or -.000009)

Form	Examples
±n.E±e	-3.E-5, 700.E3, 121766.E-4
	(-3.E-5 expresses $-3 \times 10^{-5} = -.00003$; 700.E3 expresses 700×10^{3}, or 700,000; 121766.E-4 expresses $121,766 \times 10^{-4}$, or 12.1766)
±nE±e	-321E5, 79E-2, 769E-7
	(-321E5 expresses -321×10^{5}, or 32,100,000; 79E-2 expresses 79×10^{-2}, or .79; 769E-7 expresses 769×10^{-7}, or .0000769)

Any number expressed in one of the above formats is converted by the Assembler to floating point format; expressed as a 23-bit binary fraction and a 7-bit binary exponent. The binary point of the fractional portion is assumed to the immediate left of bit 14 in word 1. Both the fraction and the exponent carry a sign bit indicating positive (0) or negative (1); thus a floating point number occupies 32 bits, or two computer words:



As illustrated by the expressions 2.5, 250E-2, .25E1, there are many ways of expressing the same value. These expressions are all converted to the same floating point format through a convention called normalizing.

Normalizing consists of placing the point directly to the left of the most significant digit and adjusting the exponent such that the normalized number and the number specified have the same value. For positive binary numbers, the most significant digit is the left-most 1-bit. For negative binary numbers (in 2's complement form) the most significant digit is the left-most zero-bit. For example, to convert the expression 45E-1 to normalized binary:

$$45E\text{-}1 = 4.5_{10} = 4.4_{8} = 100.1_{2} = .1001_{2} \times 2^{3}$$

The expressions 4.5, 4500E-3, .00045E4 all result in the same normalized binary number $.1001_2 \times 2^3$.

To convert a decimal number to floating point format, this procedure is followed:

(1) Convert the decimal number to binary

Examples: (a) $2.5_{10} = 2.4_8 = 10.1_2$

(b) $-435E-2 = -4.35_{10} = -4.2546314_8 =$

$-100.010101100110011001100_2 =$

$011.101010011001100110100_2$

(in 2's complement)

(2) Normalize

(a) $10.1_2 = .101 \times 2^2$

(b) $011.101010011001100110100_2 =$

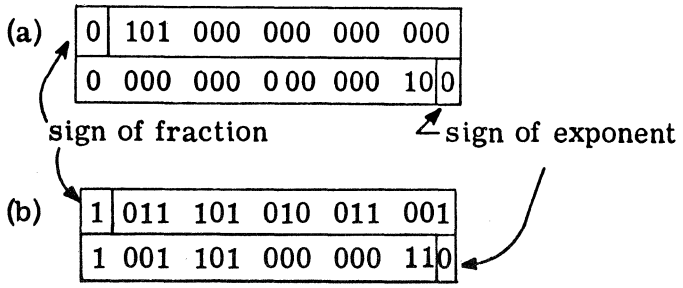
$.011101010011001100110100_2 \times 2^3$

(3) Convert the exponent to binary (if negative, convert to 2's complement)

(a) $2_{10} = 10_2$

(b) $3_{10} = 11_2$

(4) Express in 2-word floating point format



Examples of DEC:

Instruction		Generated floating point values
DEC -.695,400E-4	word 1	1010011100001010
	2	0011110100000000
	3	0101000111101011
	4	1000010111111001
DEC 2.5,-1.0	word 1	0101000000000000
	2	0000000000000100
	3	1000000000000000
	4	0000000000000000

9.4.4 OCT

OCT generates one or more octal constants in consecutive words.

Label	Op Code	Operand
lname	OCT	$o_1[,o_2,\ldots,o_n]$

lname — A label symbol, if used, refers to the first octal constant generated.

o_i — octal constant, one to six digits: $b_1b_2b_3b_4b_5b_6$, where b_1 may be 0 or 1, b_2-b_6 may be 0 - 7. Constants less than 6 characters are right-justified in the computer word. If no sign is given, the constant is assumed positive. The letter B must not be used after the constants in the operand field; it is used when defining an octal term in any instruction other than OCT.

Examples:

Instruction		Generated Words
OCT	77	15 0 0000000000111111
OCT	107642,-177,10101	1000111110100010 1111111110000001 0001000001000001
OCT	1976	(Illegal; octal constants only include digits 0 through 7)
OCT	-177777	15 0 1000000000000001
OCT	177B	(Illegal; B is not used to indicate an octal number in the OCT pseudo instruction.)

9.5 ARITHMETIC SUBROUTINE CALLS

These pseudo instructions provide calls to arithmetic subroutines which perform often-used functions not available with any one machine instruction.† This group of pseudo instructions may only be used in relocatable programs; the operand field may contain any relocatable expression or an absolute expression resulting in a value less than or equal to 77_8.‡

9.5.1 MPY

This pseudo instruction calls a subroutine which multiplies the contents of the A-register by the contents of a memory location or a literal and stores the product in registers B and A.

Label	Op Code	Operand
	MPY	{ m[,I] / lit }

m — absolute or relative address. If absolute, must result in value less than or equal to 77_8.

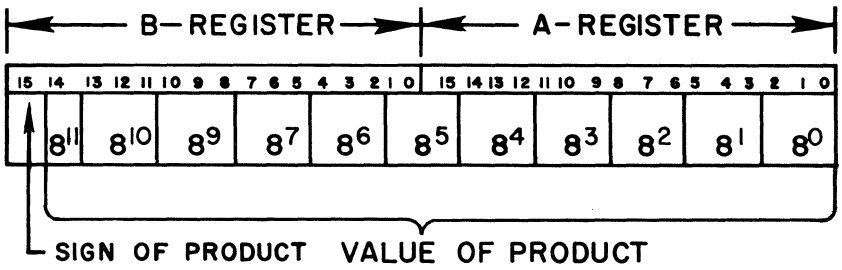
† Each call generates two words of code:

```
JSB     .<mnemonic>
DEF     m[,I]
```

‡ If the configuration includes the Extended Arithmetic Unit option, the mnemonics MPY, DIV, DLD, and DST result in machine instructions; they may be used in absolute as well as relocatable programs.

Label	Op Code	Operand
	I	indirect addressing indicator
	lit	literal value

The result is stored right-justified in the combined B- and A-registers:



The lower blocks (8^i) indicate the octal place value of the bit positions. Negative numbers are in two's complement form.

For example:

	Before Execution	After Execution
MPY VALUE	(A) = 000173_8	(B) = 000000
	(VALUE) = 000034_8	(A) = 006564_8
	(B) = any quantity	(VALUE) = 000034_8
MPY DANTE	(A) = 101325_8	(B) = 147761_8
	(DANTE) = 061111_8	(A) = 154275_8
	(B) = any value	(DANTE) = 061111_8
MPY D20	(A) = 000075_8	(B) = 000000
		(A) = 002304

Note that in the second example, the negative answer (in eight's complement form) is really 1774354275. Split into the two 16-bit registers and right-justified, it is represented as shown above.

9.5.2 DIV

DIV divides the contents of B and A by the contents of a memory location or a literal; the quotient is stored in A and the remainder in B.

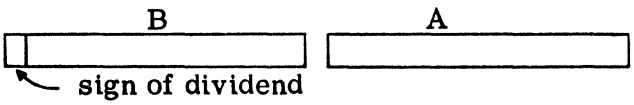
Label	Op Code	Operand
	DIV	{m[,I] / lit}

m — absolute or relocatable address. If absolute, must result in value less than or equal to 77_8.

I — indirect addressing indicator

lit — literal value

Initially, the dividend is stored right-justified in the combined B- and A-registers:



An attempt to divide by zero causes the overflow bit to be set.

For example:

	Before Execution	After Execution
DIV ALAN	(B) = 000000	(B) = 000000
	(A) = 054147_8	(A) = 000563_8
	(ALAN) = 000075_8	(ALAN) = 000075_8
DIV = B73	(B) = 000000	(B) = 000002_8
	(A) = 000075_8	(A) = 000001_8

9.5.3 FMP

This pseudo instruction multiplies the floating point quantity in registers A and B by a two-word floating point quantity in memory or a literal and stores the result in the A and B registers in floating point format.

Label	Op Code	Operand
	FMP	$\left\{\begin{matrix} m[,I] \\ lit \end{matrix}\right\}$

m — location of first word of two-word floating point quantity

I — indirect addressing indicator

lit — literal value

For example:

	Before Execution	After Execution
FMP SOCK	(A) = 050000_8 (B) = 000004_8 Quantity in A and B registers represents 2.5_{10} in floating point format. (SOCK) = 100000_8 (SOCK+1) = 000000 Quantity in two memory locations represents -1.0_{10} in floating point format.	(A) = 130000_8 (B) = 000004_8 Quantity in A and B registers represents -2.5_{10} in floating point format. (SOCK) = 100000_8 (SOCK+1) = 000000
FMP = F10.0	(A) = 074000_8 (B) = 000004_8 Quantity in A and B registers represents 3.75_{10} in floating point format.	(A) = 045400_8 (B) = 000014_8 Quantity in A and B registers represents 37.5_{10} in floating point format.

9.5.4 FDV

FDV divides the floating point quantity in registers A and B by a two-word floating point quantity in memory or a literal and stores the result in the A- and B-registers in floating point format.

Label	Op Code	Operand
	FDV	{ m[,I] / lit }

m — location of first word of two-word floating point quantity

I — indirect addressing indicator

lit — literal value

For example:

	Before Execution	After Execution
FDV SOCK	(A) = 050000_8	(A) = 130000_8
	(B) = 000004_8	(B) = 000004_8
	Quantity in A and B registers represents 2.5_{10} in floating point format.	Quantity in A and B registers represents -2.5_{10} in floating point format.
	(SOCK) = 100000_8	(SOCK) = 100000_8
	(SOCK+1) = 000000	(SOCK+1) = 000000
	Quantity in two memory locations represents -1.0 in floating point format.	
FDV = F2.0	(A) = 074000_8	(A) = 074000_8
	(B) = 000004_8	(B) = 000002_8
	Quantity in A and B registers represents 3.75_{10} in floating point format.	Quantity in A and B registers represents 1.875_{10} in floating point format.

9.5.5 FAD

This pseudo operation adds the floating point quantity in registers A and B to a two-word floating point quantity in memory or a literal and stores the result in the A- and B-registers in floating point format.

Label	Op Code	Operand
	FAD	$\left\{\begin{matrix} m[,I] \\ lit \end{matrix}\right\}$

m — location of first word of two-word floating point quantity

I — indirect addressing indicator

lit — literal value

For example:

	Before Execution	After Execution
FAD SOCK	(A) = 050000_8	(A) = 060000_8
	(B) = 000004_8	(B) = 000002_8
	Quantity in A and B registers represents -2.5_{10} in floating point format.	Quantity in A and B registers represents 1.5_{10} in floating point format.
	(SOCK) = 100000_8	(SOCK) = 100000_8
	(SOCK+1) = 000000	(SOCK+1) = 000000
	Quantity in two memory locations represents -1.0 in floating point format.	
FAD = F1Ø.25	(A) = 074000_8	(A) = 070000_8
	(B) = 000004_8	(B) = 000010_8
	Quantity in A and B registers represents 3.75 in floating point format.	Quantity in A and B registers represents 14_{10} in floating point format.

9.5.6

FSB

FSB subtracts a two-word floating point quantity in memory or a literal from a floating quantity in registers A and B and stores the result in the A- and B-registers in floating point format.

Label	Op Code	Operand
	FSB	$\left\{\begin{matrix} m\,[,I] \\ lit \end{matrix}\right\}$

m — location of first word of two-word floating point quantity

I — indirect addressing indicator

lit — literal

For example:

	Before Execution	After Execution
FSB SOCK	(A) = 050000_8	(A) = 070000_8
	(B) = 000004_8	(B) = 000004_8
	Quantity in A and B registers represents 2.5_{10} in floating point format.	Quantity in A and B registers represents 3.5_{10} in floating point format.
	(SOCK) = 100000_8	(SOCK) = 100000_8
	(SOCK+1) = 000000	(SOCK+1) = 000000
	Quantity in two memory locations represents -1.0 in floating point format.	
FSB = F3.5	(A) = 074000_8	(A) = 040000_8
	(B) = 000004_8	(B) = 000377_8
	Quantity in A and B registers represents 3.75 in floating point format.	Quantity in A and B registers represents $.25_{10}$ in floating point format.

9.5.7 DLD

DLD loads the A and B registers with the contents of two consecutive words in memory.

Label	Op Code	Operand
	DLD	m[,I]

m — location of first word -- the contents of this location is loaded into the A-register. Location m+1 is loaded into B-register.

I — indirect addressing indicator

For example:

	Before Execution	After Execution
DLD FLPT	(A) = any quantity	(A) = 017777_8
	(B) = any quantity	(B) = 177400_8
	(FLPT) = 017777_8	(FLPT) = 017777_8
	(FLPT+1) = 177400_8	(FLPT+1) = 177400_8
DLD IND,I	(A) = any quantity	(A) = 035467_8
	(B) = any quantity	(B) = 054100_8
	(IND) = 002177_8	(IND) = 002177_8
	(2177_8) = 035467_8	(2177_8) = 035467_8
	(2200_8) = 054100_8	(2200_8) = 054100_8

9.5.8 DST

DST stores the contents of the A and B registers into two consecutive memory locations.

Label	Op Code	Operand
	DST	m[,I]

m — location of first word -- the contents of the A-register is stored in this location. The contents of the B-register is stored in location m+1.

I — indirect addressing indicator

For example:

	Before Execution	After Execution
DST TROUT	(A) = 000042_8	(A) = 000042_8
	(B) = 177401_8	(B) = 177401_8
	(TROUT) = any quantity	(TROUT) = 000042_8
	(TROUT+1) = any quantity	(TROUT+1) = 177401_8
DST IVAN, I	(A) = 017532_8	(A) = 000000
	(B) = 152525_8 (Note 1 in column 15)	(B) = 017000_8
	(IVAN) = 102027_8	(IVAN) = 102027_8
	(2027_8) = 002777_8	(2027_8) = 002777_8
	(2777_8) = 000000	(2777_8) = 000000
	(3000_8) = 017000_8	(3000_8) = 017000_8

9.5.9 SWP

This instruction exchanges the contents of the A and B registers. The contents of the A register is shifted into the B register and the contents of the B register, into the A register. The SWP instruction may be used only in a configuration which includes the Extended Arithmetic Unit option.

Label	Op Code	Operand
	SWP	

The instruction has no operand.

9.6 ASSEMBLY LISTING CONTROL

Assembly listing control pseudo instructions allow the user to control the Assembly listing output during pass 2 or 3 of the assembly process. These pseudo instructions may be used only when the source program is translated by the Assembler provided for 8K or larger machines (8, 192-word memory or larger).

9.6.1 UNL

UNL allows suppression of selected portions of the source program from the assembly listing.

Label	Op Code	Operand
	UNL	

All listable output following the UNL pseudo instruction is suppressed until either an LST or END pseudo instruction is encountered. The UNL is also suppressed from the listing. The source statement sequence numbers, printed in character positions 1-4 of the listing, are incremented to allow for the instructions encountered between a UNL and an LST or END. Diagnostic messages for errors encountered in the suppressed instructions will always be printed. The binary object program is not affected.

9.6.2 LST

LST re-initiates the listing of the source program which was suppressed by a previous UNL psuedo instruction.

Label	Op Code	Operand
	LST	

A UNL instruction followed by another UNL instruction, an LST followed by an LST, or an LST not preceded by a UNL are not considered errors by the Assembler.

Example:

The Assembler listing shown below was generated from the following source program segment:

```
        LDA  KNITI
        STA  PURL2
        UNL
        ISZ  COUNT
        JMP  RED
        LST
```

```
0014  00012 062001R         LDA KNITI
0015  00013 072004R         STA PURL2
0019                        LST
```

Note that the UNL, ISZ, and JMP instructions are not listed, but the source statement sequence number is increased from ØØ15 to ØØ19.

9.6.3 SKP

SKP causes the Assembly listing to be skipped to the bottom of the current page.

Label	Op Code	Operand
	SKP	

The SKP instruction is not printed on the listing; however, the source statement sequence number is incremented to allow for the SKP. Listing continues with the instruction following the SKP at the top of the next page.

9.6.4 SPC

SPC causes the Assembly listing to be skipped a specified number of lines on the list output, or to the bottom of the page (whichever occurs first) before printing the next instruction.

Label	Op Code	Operand
	SPC	n

n any absolute expression; specifies the number of lines to be skipped.

The SPC instruction is not printed on the listing; however, the source statement sequence number is incremented to allow for the SPC. Listing continues with the instruction following the SPC.

For example:

The Assembler listing shown below was generated from the following source program segment:

```
        LDA  FREEN
        STA  FISBY
        SPC  3
        LDA  COOL
        STA  POOL
```

```
0021  00017 062006R        LDA FREEN
0022  00020 072007R        STA FISBY


0024  00021 062002R        LDA COOL
0025  00022 072003R        STA POOL
```

Note that the SPC instruction is not listed, but the source statement sequence number is incremented from ØØ22 to ØØ24.

9.6.5 SUP

SUP suppresses the listing of all but the first code line generated by the following pseudo instructions:

ASC	DIV	FAD	FSB
OCT	DLD	FDV	MPY
DEC	DST	FMP	

SUP also suppresses the listing of literal values generated by the Assembler, if specified immediately before the END statement in the source program.

Label	Op Code	Operand
	SUP	

9.6.6 UNS

UNS re-initiates the listing of extended code lines which were suppressed by an SUP instruction.

Label	Op Code	Operand
	UNS	

An SUP instruction preceded by an SUP, UNS preceded by UNS, or UNS not preceded by SUP is not considered an error by the Assembler.

For example:

The listing segment shown below was generated from the following source program segment:

```
        LDA  LINE1
        STA  BABLE
        SUP
LINE1   ASC  15,BLESSINGS ON THEE, LITTLE MAN
        UNS
LINE2   ASC  15,BAREFOOT BOY WITH CHEEK OF TAN
```

```
0027  00023 062025R        LDA LINE1
0028  00024 072010R        STA BABLE
0029                       SUP
0030  00025 041114  LINE1  ASC 15,BLESSINGS ON THEE, LITTLE MAN
0031                       UNS
0032  00044 041101  LINE2  ASC 15,BAREFOOT BOY WITH CHEEK OF TAN
      00045 051105
      00046 043117
      00047 047524
      00050 020102
      00051 047531
      00052 020127
      00053 044524
      00054 044040
      00055 041510
      00056 042505
      00057 045440
      00060 047506
      00061 020124
      00062 040516
```

9.6.7 HED

This pseudo instruction causes a specified heading to be printed at the top of a page.

Label	Op Code	Operand
	HED	m

m — a string of up to 56 ASCII characters to be printed as a heading

If HED is specified before the NAM or ORG pseudo instruction at the beginning of a program, the heading m will be printed at the top of the first page of the pass 2 list output and at the top of every following page until another HED instruction or the end of the listing occurs.

If HED is specified elsewhere within the program, the Assembler skips to the top of the next page, prints the heading, and continues listing with the instruction following the HED.

The source statement containing the HED pseudo instruction is not printed on the listing, but the source statement sequence number is incremented to allow for the instruction.

Example:

The listing segment shown below was generated from the following source program segment:

```
Label     Operation      Operand                                Comments
1    5    10        15        20        25        30        35        40        45        50        55
ASMB,R,L,T
      HED      THIS IS A RELEVANT AND REVEALING EXAMPLE
      NAM  TEST
COUNT BSS  1
KNIT1 BSS  1
COOL  BSS  1
POOL  BSS  1
PURL2 BSS  1
```

```
PAGE ØØØ2 #Ø1  THIS IS A RELEVANT AND REVEALING EXAMPLE

ØØØ1                         ASMB,R,L,T
ØØØ3  ØØØØØ                         NAM  TEST
ØØØ4  ØØØØØ  ØØØØØØ  COUNT  BSS  1
ØØØ5  ØØØØ1  ØØØØØØ  KNIT1  BSS  1
ØØØ6  ØØØØ2  ØØØØØØ  COOL   BSS  1
ØØØ7  ØØØØ3  ØØØØØØ  POOL   BSS  1
ØØØ8  ØØØØ4  ØØØØØØ  PURL2  BSS  1
```

Note that the HED pseudo instruction is not listed, but the source statement sequence number is incremented from ØØØ1 to ØØØ3.

1. Which of the following are invalid?

   (a) NAM PROG

   (b) NAMPROG

   (c) NAM .123

   (d) NAM E.RASE

   (e) NAM

   (f) NAM 2016

2. When is the ORG pseudo not valid in a relocatable assembly?

3. What is the significance of the Operand field in the END statement of a relocatable program?

4. Which of the following COM statements should be loaded first? Why?

   (a) COM A(5), B(8), C(15)

   (b) COM A(4), B(3), C(10), D(20)

   (c) COM E(6), A(7)

5. If the symbol CAT is used in PROGA to refer to a label in PROGB, what must be specified to provide the necessary linkage between the two programs?

6. What pseudo is used to generate an indirect address?

7. How many characters may be generated by ASC?

8. Write a routine which will find the 2's complement of the 25 values placed in consecutive locations beginning at POS and store the complements in the 25 locations beginning at NEG. Assume the POS area as the first 25 locations in a common area.

9. Given 50 quantities stored in locations TAB to TAB+49. Write a routine to store in locations TAG to TAG+49 in reverse order. Assume the TAB area as the first 50 locations in a common area.

BCS INPUT/OUTPUT REQUESTS 10

The Input/Output Control routine (.IOC.), a part of the Basic Control System, provides a simplified method of performing I/O operations. The user provides the information necessary, and .IOC. interprets the call, initiates the operation, and returns control to the user's program.

Several operations may be performed: (1) transferring data between the computer and an I/O device, (2) positioning of a reel of magnetic tape, (3) terminating a previously issued I/O request before all data is transferred, and (4) determining the status of an operation or a device.

Input/output operations are accomplished through a set of subroutines called drivers. The I/O request provides information as to which device is to be used, whether data is to be transferred into or out of the computer, and the format of the data (binary or ASCII). The .IOC. routine then checks an internal equipment table, determining the channel to which the device is connected, and gives control to the related driver. The driver routine reads or writes the specified amount of data, processing all interrupts that occur during the transfer.

Input/Output requests are specified as a series of Assembly-language instructions. A JSB instruction to .IOC. is specified first; thus .IOC. must be declared as an external point in the program with the EXT pseudo instruction. The JSB is followed by other instructions which form the call. .IOC. always returns control to the instruction following the last instruction of the I/O request.

10.1 DATA TRANSFER REQUEST

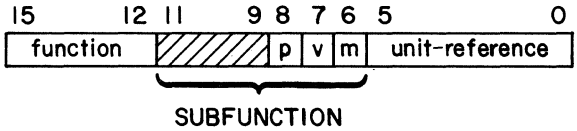
The general form of the data transfer request is:

```
JSB     .IOC.
OCT     <function> <subfunction> <unit-reference>
JSB  }
JMP  }  reject address
DEF     buffer address
```

```
DEC  }
OCT  } buffer length
  .
  .
  .
EXT    .IOC.
```

10.1.1 FUNCTION, SUBFUNCTION, AND UNIT-REFERENCE

The second instruction of the data transfer request defines the function to be performed and the unit of equipment for which the action is to be taken. This information is supplied in the form of an octal constant. .IOC. interprets the bit combination as follows:



FUNCTION

Bits 15-12 define the function to be performed; 01_8 defines a read operation, 02_8 defines a write operation.

SUBFUNCTION

The subfunction (bits 11-6) defines the options for certain input/output operations:

p = 1 — Print input; the ASCII data read from the 2752A Teleprinter is to be printed as it is received.

v = 1 — Variable length binary input: the value in bits 15-8 of the first word on an input paper tape indicates the length of the record (including the first word). If the value exceeds the length of the buffer (defined by the fifth word of the Input/Output request), only the number of words specified as the buffer length are read. If v = 0, the buffer length always determines the length of the record to be transmitted. If the device does not read paper tape, the parameter is ignored.

m = 1 Mode: the data is transmitted in binary form exactly as it appears in memory or on the external device. If m = 0, the data is transmitted in ASCII format. (See Record Formats, Appendix E.)

Allowable combinations of function and subfunction codes are as follows:

Operation	Octal Value of Bits 15-6
Read ASCII or BCD record	0100
Read ASCII record and print	0104
Read binary record	0101
Read variable length binary record	0103
Write ASCII or BCD record	0200
Write binary record	0201

Combinations considered illegal by .IOC. are rejected.

UNIT-REFERENCE

The value specified for the unit-reference field indicates the unit of equipment on which the operation is to be performed. The number may represent a standard unit assignment or an installation unit assignment. Standard unit numbers are as follows:

Number	Name	Usual Equipment Type
1	Keyboard Input	Teleprinter
2	Teleprinter Output	Teleprinter
3	Program Library	Punched Tape Reader
4	Punch Output	Tape Punch
5	Input	Punched Tape Reader
6	List Output	Teleprinter

Installation unit numbers may be in the range 7_8-74_8 with the largest value determined by the number of units of equipment available at the installation. The installation unit number specified in an I/O request is related to a specific device through a BCS equipment table (EQT), defined at the time the computer and related software is installed. This table defines the type of equipment (Teleprinter, magnetic tape, and so forth), the channel on which each unit is connected, and other related details. The first unit described in the table is referred to by

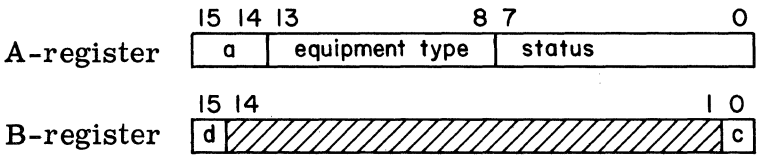
the number 7_8; the second, 10_8; the third, 11_8; and so forth. The entries for one possible equipment table might establish the following relationships:

Installation Unit Number	Device	I/O Channel
7	Teleprinter	12 and 13
10	Punched Tape Reader	10
11	Tape Punch	11

The standard unit numbers are associated with physical equipment via a standard equipment table (SQT) and the EQT. The SQT is a list of references to the EQT. SQT is also created at the time the computer and related software is installed. Each standard unit may be a separate device, or a single device accessed by several standard unit numbers as well as an installation unit number.

10.1.2 REJECT ADDRESS

.IOC. transfers control to the third instruction of the I/O request if the input/output operation cannot be performed. On transfer, status information is provided in the A- and B-registers which may be checked by the user's program. The third word usually contains a reject address which is the starting location of a user subroutine designed to check the cause of the reject and take appropriate action.



The contents of the A-register indicate the physical status of the equipment (see Status Request, Section 10.4).

The contents of the B-register indicate the cause of the reject:

- d = 1 — The device or driver subroutine is busy and therefore unavailable, or, for Kennedy 1406 Tape Unit, a broken tape condition encountered.
- c = 1 — A Direct Memory Access channel is not available to operate the device.

d = c = 0	The function or subfunction selected is not legal for the device.
	For HP 2020A/B Magnetic Tape unit, device or driver is busy, or device is in local status.

10.1.3 BUFFER ADDRESS

The buffer address specified in the fourth instruction is the location of the first word of data to be written on an output device or the first word of a block reserved for storage of data read from an input device. The block must have been reserved in the program by a BSS or COM instruction.

10.1.4 BUFFER LENGTH

The octal or decimal integer specified in the fifth instruction is the number of words or 8-bit characters to be input or output. If the length is given as words, the specification is a positive integer; if characters, a negative integer. For example, either DEC 10 or DEC -20 would specify the same amount of data to be transferred.

Characters may be specified only if the device is capable of 8-bit character transmission. The buffer length for data that may be printed on the Teleprinter should be no more than 72 characters (36 words). The buffer length for data transmitted via a Direct Memory Access channel may be up to 16K words; character transmission is applicable, but only an even number of characters will be transmitted.

Examples:

```
Label  Operation  Operand    Comments
1    5    10    15    20   25   30   35   40   45   50
       NAM PROG
        .
        .
        .
       EXT .IOC.     DECLARE .IOC. AS EXTERNAL.
LINE   BSS 36        RESERVE STORAGE AREAS--36 WORDS
       COM BKB(1ØØ)  FOR LINE AND 1ØØ WORDS (IN THE
       DEF BKB       COMMON BLOCK) FOR BKB.
        .
        .
READ1  JSB .IOC.     READ 72 ASCII CHARACTERS FROM
       OCT 1ØØØ5     THE STANDARD INPUT UNIT AND
       JMP REJAD     STORE AT LINE. IF REQUEST IS
       DEF LINE      REJECTED, TRANSFER CONTROL TO
       DEC -72       REJAD.
        .
        .
        .
WRIT1  JSB .IOC.     WRITE 1ØØ BINARY WORDS ON UNIT
       OCT 2Ø111     11, THE THIRD DEVICE DESCRIBED
       JMP REJAB     IN THE EQT. DATA IS CURRENTLY
       DEF BUF       STORED IN THE COMMON BLOCK
       DEC 1ØØ       STARTING AT LOCATION BUF.
        .
        .
        .
       END
```

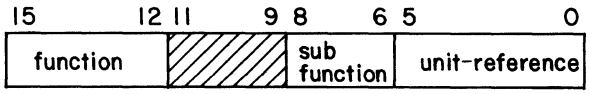
10.2 MAGNETIC TAPE CONTROL REQUESTS

This request controls the positioning of a reel on a magnetic tape device. The calling sequence is similar to the data transmission request, but consists of only three words:

```
EXT         .IOC.
 .
 .
 .
JSB         .IOC.
OCT          <function> <subfunction> <unit-reference>
{JSB}
{JMP}       reject address
```

10.2.1 FUNCTION, SUBFUNCTION, AND UNIT-REFERENCE

The second instruction of this request defines the function to be performed and the unit of equipment for which the action is to be taken. This information is supplied in the form of an octal constant. .IOC. interprets the bit combination as follows:



FUNCTION

A function code of 03_8 in bits 15-12 defines the calling sequence as a tape positioning request.

SUBFUNCTION

The subfunction defines the type of positioning:

Octal Code	Operation
1	Write end-of-file
2	Backspace one record
3	Forward space one record
4	Rewind
5	Unload

Write End-of-File

A standard EOF character (17_8) is written on tape, Control returns to the normal location. A three-inch gap is written before the EOF mark. A status request will show the EOF bit set in the status field.

Backspace one record

The tape is positioned at the beginning of the previous record.

Forward space one record

The tape is positioned at the beginning of the next record.

Rewind

This command initiates a rewind operation and then returns control to the normal return location.

Rewind and Standby

This causes the tape to be positioned at load point and switches the device to local status. Control returns to the normal return location after the operation is initiated.

UNIT-REFERENCE

The unit reference field is defined in the same manner as for the data transmission request.

10.2.2 REJECT ADDRESS

The reject address, which is usually specified in the third instruction, is the starting location of a user subroutine designed to check the cause of the reject and take appropriate action. Status information is provided in the A- and B-registers as for the data transmission request.

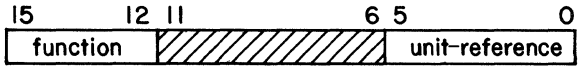
10.3 CLEAR REQUEST

The clear request terminates a previously issued input or output operation before all data is transmitted.† The calling sequence is as follows:

```
EXT     .IOC.
 .
 .
 .
JSB     .IOC.
OCT     <function> <unit-reference>
```

10.3.1 FUNCTION AND UNIT-REFERENCE

The second instruction of the clear request defines the function to be performed and the unit of equipment for which the action is to be taken. This information is supplied in the form of an octal constant. .IOC. interprets the bit combination as follows:



FUNCTION

A function code of 00_8 in bits 15-12 defines the calling sequence as a clear request.

UNIT-REFERENCE

The unit-reference field is defined in the same manner as for the data transmission request.

If the unit-reference number is specified as 00 (i.e., the second word of the calling sequence is OCT Ø), all previously requested input and output operations are terminated. This request, the system clear request, makes all devices available for the initiation of a new operation. On return from a system clear request, the contents of the A- and B-registers are meaningless.

† The devices are not ready immediately; the driver, however, is available on return.

Examples:

```
Label   Operation   Operand                  Comments
NAM TIME
       .
       .
       .
       EXT  .IOC.          DECLARE .IOC. AS EXTERNAL
MSG    BSS  36             RESERVE 36 WORDS FOR MSG.
       .
       .
       .
READM  JSB  .IOC.          READ AND PRINT A MESSAGE OF ONE
       OCT  1Ø4Ø1          LINE FROM THE TELEPRINTER. WHEN
       JMP  REJ            CONTROL RETURNS AFTER INITIATING
       DEF  MSG            THE REQUEST, THE JSB TRANSFERS
       DEC  36             TO A SUBROUTINE (TIMER) WHICH
       JSB  TIMER          CHECKS THE TIME ALLOWED FOR A
       .                   MESSAGE TO BE COMPLETED.
       .
       .
CLRRD  JSB  .IOC.          IF THE MESSAGE IS NOT FURNISHED
       OCT  1              WITHIN A SPECIFIC TIME LIMIT,
       .                   THE REQUEST IS CLEARED.
       .
       .
       END
```

```
Label   Operation   Operand                  Comments
       NAM  URGENT
       .
       .
       .
       EXT  .IOC.          DECLARE .IOC. AS EXTERNAL
RDARA  BSS  5Ø             RESERVE 5Ø WORDS FOR INPUT AREA
       .
       .
       .
       JSB  .IOC.          CLEAR ALL PREVIOUSLY ISSUED
       OCT  Ø              I/O REQUESTS
       JSB  .IOC.          READ A VARIABLE LENGTH BINARY
       OCT  Ø1Ø31Ø         RECORD FROM UNIT 1Ø INTO
       JMP  RJCT           RDARA.
       DEF  RDARA
       DEC  5Ø
       .
       .
       .
       END
```

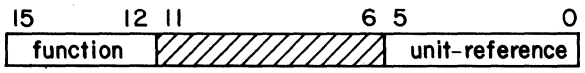
10.4 STATUS REQUEST

This request may be directed to .IOC. to determine the status of previous input/output requests or to determine the physical status of one or all units of equipment. The general form of the request is as follows:

```
EXT     .IOC.
 .
 .
 .
JSB     .IOC.
OCT     <function> <unit-reference>
```

10.4.1 FUNCTION AND UNIT-REFERENCE

The second instruction of the status request defines the function to be performed and the unit of equipment for which the action is to be taken. This information is supplied in the form of an octal constant. .IOC. interprets the bit combination as follows:



FUNCTION

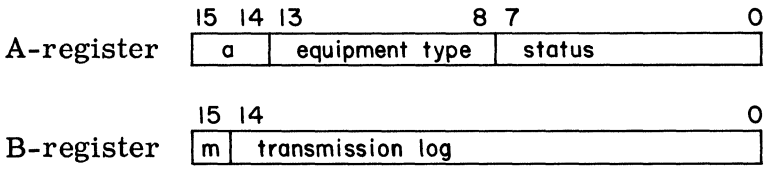
A function code of 04_8 in bits 15-12 define the function as a status operation.

UNIT-REFERENCE

The unit-reference field is defined in the same manner as for the data transmission request.

If the unit-reference number is specified as 00 (i.e., the second word of the calling sequence is OCT 4ØØØØ), the request is interpreted as a system request.

If information is requested for a single unit, .IOC. returns to the location immediately following the request with the status information in the A- and B-registers:



a — availability of device;

- 0 The device is available; the previous operation is complete.
- 1 The device is not available; the previous operation is complete but a transmission error has been detected or the device (tape) is in local status.
- 2 The device is not available; the operation is in progress.

equipment type — This field contains a 6-bit code identifying the device referred to:

- 00-07 Paper tape devices
  - 00 2752A Teleprinter
  - 01 2737A Punched Tape Reader
  - 02 2753A Tape Punch
- 10-17 Unit Record devices
- 20-37 Magnetic Tape and Mass Storage devices
  - 20 Kennedy 1406 Incremental Tape Transport
  - 21 HP 2020 Magnetic Tape Unit
- 40-77 Instrumentation devices
  - 40 Data Source Interface
  - 41 Integrating Digital Voltmeter
  - 42 Guarded Crossbar Scanner
  - 43 Time Base Generator
  - 77 HP 2401C/HP 2911 Scanning Driver (HP 2018 System)

status

The status field indicates the actual status of the device when the data transmission is complete. The contents depend on the type of device referred to:

Teleprinter reader or Punched Tape reader:

Bits 7-0	Condition
xx1xxxxx	end-of-tape (10 feed frames)

Tape punch:

Bits 7-0	Condition
xx1xxxxx	tape supply low

Kennedy 1406 Incremental Tape Transport:

Bits 7-0	Condition
xx1xxxxx	end-of-tape mark sensed
xxxx1xxx	broken tape; no tape on write head
xxxxxxx1	device busy

HP 2020 Magnetic Tape Unit

Bits 7-0†	Condition
1xxxxxxx	end-of-file record (17_8) encountered while reading, forward spacing, or backward spacing
x1xxxxxx	start-of-tape marker sensed
xx1xxxxx	end-of-tape marker sensed
xxx1xxxx	timing error on read/write
xxxx1xxx	I/O request rejected: a. tape motion required but controller busy b. backward tape motion required but tape at load point c. write request given but reel does not have write enable ring

† Hardware status bit 8 is not included in status field (similar information in bit 0).

Bits 7-0	Condition
xxxxx1xx	Reel does not have write enable ring or tape unit is rewinding
xxxxxx1x	Parity error on read/write
xxxxxxx1	Tape in motion or unit in local status

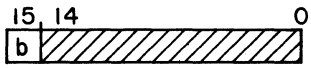
m — This bit defines the mode of the data transmission.

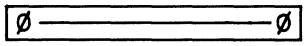
0 ASCII or BCD

1 binary

transmission log — This field is a log of the number of characters or words transmitted. The value is given as a positive integer and indicates characters or words as specified in the calling sequence. The value is stored in this field only when the request is completed; that is, when all data is transmitted or when a transmission error is detected.

If a system status request is made, the information in the A and B registers is as follows:

A-register = 

B-register = 

b System Status

0 no device busy

1 at least one device is busy

HP 2020A/B
Status
Information

If errors (timing or parity) are detected during input/output operations, the HP 2020A/B subroutine will attempt to repeat the operation four more times; a total of five Read or Write operations will be initiated.

For an output operation, the sequence of instructions involves a write, a backspace, writing a three inch gap, and then the next write attempt. If the error persists after the five attempts, control returns to the user program at the normal return location. If a Status operation is performed at this point, the word in the A-Register would contain a 1 in the "a" field and either the timing bit (4) or the parity error bit (1) set in the status field. For a Write operation, the record produced by the last attempt will be on tape. For a Read, the buffer will contain the record read on the last attempt.

If the End-of-Tape marker is sensed on a Write operation, the EOT bit is set in the status field and control returns to the normal return location. If another Write is then attempted, the "a" field is set to 1 indicating a transmission error and control returns to the normal return location; no data will be written. If the End-of-Tape marker is sensed on a Read operation, the EOT bit is set in the status field and control returns to the normal return location. Another Read operation may be attempted if it is known that another record exists on the tape; if there is no record, reading continues through the physical end of the tape. This could also occur if the last record on the tape is placed before the EOT marker. (Forward motion is terminated by an end-of-record gap.)

Timing Errors

All operations are performed with the interrupt system active; data transfer is accomplished on interrupt command. Consequently, the priority of the device and state of the interrupt system are significant. When establishing the hardware configuration, the tape device should be given the highest priority channels. If not, the Library subroutine ENDIO should be called before every Read and Write command. During the execution of any input/output operation, the interrupt system may not be inhibited for more than 5 machine cycles; otherwise, a timing error may occur on high density (556 bpi) tapes.

Examples:

```
      NAM  STCHK
        .
        .
        .
      EXT  .IOC.          DECLARE .IOC. AS EXTERNAL.
INARA BSS  1Ø             RESERVE STORAGE AREA.
        .
        .
        .
READM JSB  .IOC.          READ AN ASCII RECORD FROM UNIT-
      OCT  1ØØ15          REFERENCE NUMBER 15 AND STORE AT
      JMP  RJCT           LOCATION INARA
      DEF  INARA
      DEC  1Ø
        .
        .
        .
STATM JSB  .IOC.          CHECK STATUS OF READ REQUEST. IF
      OCT  4ØØ15          BIT 15 IS SET, UNIT 15 IS BUSY--
      SSA                 LOOP ON STATUS REQUEST UNTIL OP-
      JMP  STATM          ERATION IS COMPLETE. ROTATE AND
      RAL                 CHECK BIT 14--IF SET, TRANSFER
      SSA,RSS             CONTROL TO END-OF-TAPE CHECK
      JMP  PROCS          ROUTINE. ROTATE AND CHECK BIT 5
EOT   ALF,ALF             --IF SET, TRANSFER CONTROL TO
      RAL                 ENDPR ROUTINE (PERFORMS ENDING
      SSA                 PROCESS).IF NOT SET, TRANSFER
      JMP  ENDPR          CONTROL TO ABORT ROUTINE (PER-
      JMP  ABORT          FORMS TERMINATION PROCEDURE). IF
        .                 REQUEST IS COMPLETED, CONTINUE
        .                 PROCESSING AT LOCATION PROCS.
        .
        .                 DETERMINE CAUSE OF REJECT. IF
RJCT  SSB                 DEVICE OR DRIVER BUSY, LOOP ON
      JMP  READM          REQUEST UNTIL AVAILABLE. OTHER-
      JMP  ABORT          WISE, TERMINATE PROGRAM AT ABORT
```

```
      NAM  SYSST
        .
        .
        .
AGAIN JSB  .IOC.          TEST ALL DEVICES TO SEE IF ANY
      OCT  4ØØØØ          ARE STILL BUSY-- IF AT LEAST
      SSA                 ONE DEVICE IS STILL BUSY, KEEP
      JMP  AGAIN          TESTING SYSTEM STATUS. IF NONE
      JMP  BEGIN          ARE BUSY, JUMP TO BEGIN.
      END
```

REVIEW

1. What is the name of the BCS routine which provides simplified I/O?

2. The above name must be specified in an _____ pseudo when any BCS I/O calling sequence is specified in a routine.

3. Input/output operations are accomplished through a set of subroutines called _____.

4. Installation unit numbers are related to specific devices through the _____ _____.

5. A _____ request may be specified to terminate a previously issued I/O operation.

6. Specify an I/O request to read a 10-word ASCII record from the punched tape reader with installation unit assignment 10. Transfer control to location ERROR if the operation cannot be performed.

7. Code the ERROR routine in the above question to check for the cause of the error, print either ILL FN ON TR (in the case of illegal function or subfunction)

   or

   TR DV BSY (in the case of the device or driver busy) on the standard teletype device, and halt.

ASSEMBLER INPUT AND OUTPUT 11

The Assembler accepts as input a paper tape containing a control statement and a source language program. A relocatable source language program may be divided into several subprograms or into a main program and several subroutines; the designation of these elements is optional. The output produced by the Assembler may include a punched paper tape containing the object program, an object program listing, and diagnostic messages.

11.1 CONTROL STATEMENT

The control statement must be the first statement of the source program; it directs the Assembler.

ASMB, $p_1, p_2, \ldots, p_n$

ASMB indicates the control statement; it must begin in character position one. Following the comma are two or more parameters, in any order, which define the output to be produced. No spaces may be specified within the control statement. The control statement must be terminated by an end-of-statement mark, (CR) (LF)

The parameters may be any legal combination of the following starting in character position 6:

A Absolute: The addresses generated by the Assembler are to be interpreted as absolute locations in memory. The object program may be loaded by the Basic Binary Loader.

R Relocatable: The object program may be loaded by the BCS Relocating Loader.

B Binary output: A program is to be punched according to one of the above parameters.

N All coding segments starting with IFN are to be assembled into program. (Void if Z follows.)

Z All coding segments starting with IFZ are to be assembled into program. (Void if N follows.)

L List output: A program listing is to be produced either during pass two or pass three according to one of the above parameters.

T Table print: List the symbol table at the end of the first pass.

Either A or R must be specified with any combination of B, L or T.

11.2 SOURCE PROGRAM

The source program follows the control statement. Each statement is followed by an end-of-statement mark. The first statement of the program must be a NAM statement for a relocatable program or an ORG statement indicating the origin of an absolute program. The HED pseudo instruction and statements consisting entirely of comments (indicated by the asterisk in position one), however, may appear between the ASMB statement and the NAM or ORG statement. The last statement must be an END statement and usually contains the transfer address for the start of a relocatable program.

11.3 BINARY OUTPUT

The punch output includes the instructions translated from the source program. It does not include system subroutines referenced within the source program (arithmetic subroutine calls, input/output requests to BCS, etc.). These routines must be loaded into memory at object program execution time.

11.4 LIST OUTPUT

List output as requested by the L and T parameters on the ASMB statement has the following format:

11.4.1 ASSEMBLY LISTING

The Assembler provides a listing of the Assembled source program if requested by the L parameter on the ASMB statement for the program. Each page of the listing is preceded by the page number (PAGE xxxx).

If the source program is assembled by the Assembler provided for 8K and larger machines, the number of the source tape cur-

rently being processed by the Assembler is printed following the page number (#xx). Headings requested by the HED pseudo instruction are printed as specified.

The body of the listing has the following format:

Columns	Content
1-4	Source statement sequence number generated by the Assembler
5-6	Blank
7-11	Location (octal)
12	Blank
13-18	Object code word in octal
19	Relocation or external symbol indicator
20	Blank
21-72	First 52 characters of source statement

Lines consisting entirely of remarks are printed as follows:

Columns	Content
1-4	Source statement sequence number
5-72	Up to 68 characters of remarks

11.4.2 SYMBOL TABLE LISTING

The Assembler produces a listing of the symbol table during pass 1, if requested by the T parameter on the ASMB statement for the program. Each page of the listing is preceded by a page number (PAGE xxxx).

A Symbol Table listing has the following format:

Columns	Content
1-5	Symbol
6	Blank
7	Relocation or external symbol indicator
8	Blank
9-14	Value of the symbol

The characters that designate an external symbol or type of relocation for the Operand field or the symbol are as follows:

Blank	Absolute
R	Program relocatable
B	Base page relocatable
C	Common relocatable
X	External symbol

At the end of each pass, the following is printed:

** NO ERRORS *

or

** nnnn ERRORS *

The value nnnn indicates the number of errors.

11.5 ERROR MESSAGES

The Assembler recognizes certain coding errors in the source program and produces a 1- or 2-letter mnemonic followed by the sequence number and the first 62 characters of the statement in error. The messages are printed on the Teleprinter during the passes indicated:

Error Code	Pass	Description
CS	1	Control statement error: a) The control statement contained a parameter other than the legal set. b) Neither A nor R, or both A and R were specified. c) There was no output parameter (B, T or L).
DD	1	Doubly defined symbol: A name defined in the symbol table appears more than once as: a) A label of a machine instruction. b) A label of one of the pseudo operations: BSS EQU ASC ABS DEC OCT DEF Arithmetic subroutine call

Error Code	Pass	Description
		c) A name in the operand field in a COM or EXT statement. . An arithmetic subroutine mnemonic appears in a program both as a pseudo instruction and as a label.
EN	1	An entry point has been defined in the operand field of an EXT or COM statement or has been equated to an absolute value.
EN 0000 <symbol>	2	An entry point specified in an ENT statement does not appear in the label field of a machine or BSS instruction.
IF	1	An IFZ follows an IFN (or vice-versa) without an intervening XIF. The second pseudo instruction is ignored.
IL	1	Illegal instruction: a) Instruction mnemonic cannot be used with type of assembly requested in control statement. The following are illegal in an absolute assembly: NAM EXT ENT COM ORB Arithmetic subroutine calls b) The ASMB statement has an R parameter, but NAM is not detected as the first op code. Illegal character: A literal has been specified with an illegal character for its type (e. g. , A-Z, 8 or 9 in an = B literal).
IL	2 or 3	Illegal character: A numeric term used in the operand field contains an illegal character (e. g. , an octal constant contains A-Z, 8 or 9). Illegal instruction: ORB in absolute assembly.

Error Code	Pass	Description
M	1, 2, or 3	Illegal operand: a) An operand is missing for an op code requiring one. b) Operands are optional and omitted but remarks are included for: SOC SOS HLT c) An absolute expression in one of the following instructions from a relocatable program is greater than 77B: Memory Reference DEF Arithmetic subroutine calls d) A negative operand is used with an op code field other than ABS, DEC, and OCT. e) A character other than I follows a comma in one of the following statements: ISZ ADA AND DEF JMP ADB XOR Arithmetic JSB LDA IOR subroutine LDB CPA calls STA CPB STB f) A character other than C follows a comma in one of the following statements: STC LIB OTA CLC MIA OTB LIA MIB HLT g) A relocatable expression in an ABS or REP statement. h) An illegal operator appears in an operand field (e.g. + or - as the last character).

Error Code	Pass	Description
		i) An ORG statement appearing in a relocatable program includes an expression that is base page or common relocatable or absolute.
		j) A relocatable expression contains an illegal mixture of program, base page, and common relocatable terms.
		k) An external symbol appears in an operand expression or is followed by a comma and the letter I.
		l) The literal or type of literal is illegal for the operation code used.
		m) Operand of EAU shift-rotate instruction = Ø or > 16.
NO	1	No origin definition: The first statement in the assembly containing a valid op code following the ASMB control statement and remarks, if any, is neither an ORG nor NAM statement. If the A parameter was given on the ASMB statement, the program is assembled starting at 2000; if an R parameter was given, the program is assembled starting at zero.
OP	1	Illegal op code following control statement. A valid op code has not yet been encountered and the statement being processed does not contain an asterisk in position one. The statement is assumed to contain an illegal op code; it is treated as a remarks statement.

Error Code	Pass	Description
OP	1, 2, or 3	Illegal op code: A mnemonic appears in the op code field which is not one of the accepted machine or pseudo codes. A word is generated in the object program.
OV	1, 2, or 3	Numeric operand overflow: The numeric value of a term or expression has overflowed its limit: 2^6-1 Input/Output, Overflow, Halt $2^{10}-1$ Memory Reference $2^{15}-1$ DEF and ABS operands; data generated by DEC; expressions concerned with program location counter. $2^{16}-1$ OCT
R?	Before 1	An attempt is being made to assemble a relocatable program following the assembly of an absolute program. The Assembler must be reloaded.
SO	1	There are more symbols defined in the program than the symbol table can handle.
SY	1	Illegal Symbol: A label field contains an illegal character or is greater than 5 characters. A label with illegal characters may result in an erroneous assembly if not corrected. A long label is truncated on the right to 5 characters. Too many control statements: A control statement has been input on the teleprinter and the source tape. The Assembler assumes that the source tape control statement is a label, since it begins in column 1. Thus the commas are considered as illegal characters and the "label" is too long. The binary object tape is not affected by this error,

Error Code	Pass	Description
		and the control statement entered via the teleprinter is the one used by the Assembler.
SY	2 or 3	Illegal Symbol: A symbolic term in the operand field is greater than five characters; the symbol is truncated on the right to 5 characters.
		Too many control statements: see above.
TP	1, 2, or 3	An error has occured while reading or writing magnetic tape. if the T-Register contains 102040, an irrecoverable error has occured; restart the assembly. Otherwise, correct condition and resume.
UN	1, 2, or 3	Undefined Symbol: a) A symbolic term in an operand field is not defined in the Label field of an instruction or is not defined in the operand field of a COM or EXT statement. b) A symbol appearing in the operand field of one of the following pseudo operations was not defined previously in the source program: BSS ORG ASC END EQU

Examples:

The program shown below generates the error messages shown in the listing following:

```
                Label           Operation           Operand
1               5               10                  15                  20
ASMB,R,L,T
                                ENT                 WHERE
IAM                             OCT                 9999
NOT                             OOP                 32
TOO                             DEC                 32768
BRITE                           LDA                 ?
BRITE                           STA                 FORGET
                                JMP                 BRITE,DUH
                                END
```

```
 PAGE 0001


0001                    ASMB,R,L,T

NO  0002          ENT WHERE

OP  0004  NOT     OCP 32

DD  0007  BRITE STA FORGET
IAM     R 000000
NOT     R 000001
TOO     R 000002
BRITE R 000003
**0003 ERRORS*



 PAGE 0002


EN  0000  WHERE
**0001 ERRORS*



 PAGE 0003 #01


0001                    ASMB,R,L,T

NO  0002          ENT WHERE
0002                          ENT WHERE

IL  0003  IAM     OCT 9999
0003  00000 000000  IAM     OCT 9999

OP  0004  NOT     OCP 32
0004  00001 000000  NOT     OCP 32

OV  0005  TOO     DEC 32768
0005  00002 000000  TOO     DEC 32768

UN  0006  BRITE LDA ?
0006  00003 062002  BRITE LDA ?

SY  0007  BRITE STA FORGET

UN  0007  BRITE STA FORGET
0007  00004 072002  BRITE STA FORGET

M   0008          JMP BRITE,DUH
0008  00005 026003R         JMP BRITE,DUH
0009                          END
**0008 ERRORS*
```

SAMPLE EXERCISES 12

1. Code a routine to check the answers to the examples for the MPY, DIV, FMP, FDV, FAD, and FSB pseudo-instructions, given in Section 9.5. That is, determine whether an MPY instruction multiplying the two values 173_8 and 34_8 would result in the A-register containing 006564_8 and the B-register containing 000000, and so forth.

2. Code a routine to generate 15 fixed point integers, sort the integers according to positive or negative, and print them in octal on the teleprinter. Negative numbers are to be complemented and preceded by a minus sign, and appropriate headings provided:

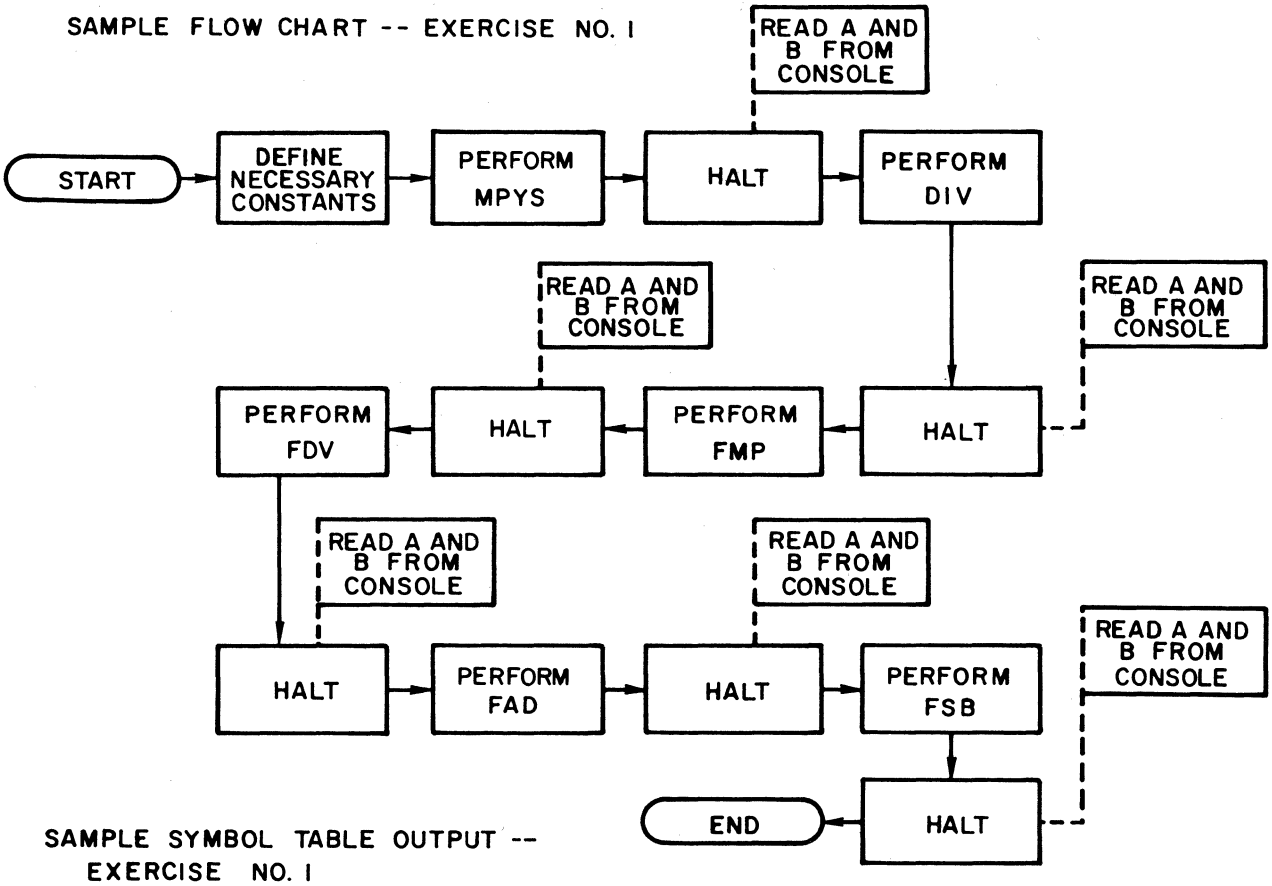
```
POSITIVE VALUES

      xxxxx
      xxxxx
      xxxxx
        .          (Indent positive values six spaces)
        .
        .
      xxxxx

NEGATIVE VALUES

     -xxxxx
     -xxxxx
        .          (Indent negative values five spaces
        .          and precede with minus sign)
        .
     -xxxxx
```

SAMPLE FLOW CHART -- EXERCISE NO. 1



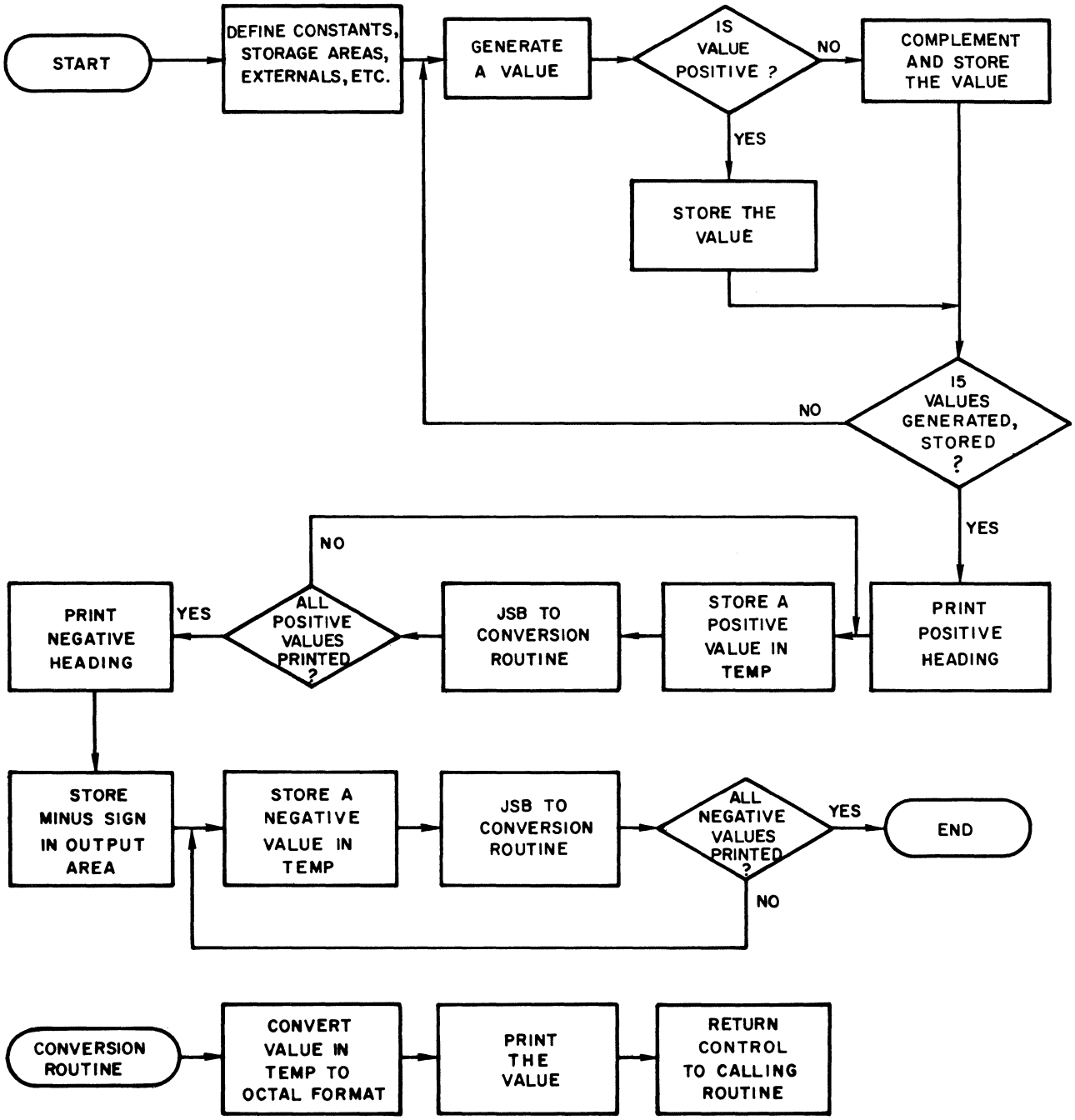
SAMPLE SYMBOL TABLE OUTPUT -- EXERCISE NO. 1

```
PAGE 0001


0001                    ASMB,R,B,L,T
AQUAN  R 000000
VALUE  R 000001
AQUAT  R 000002
DANTE  R 000003
.75    R 000004
.0     R 000005
.VAL   R 000006
.2.5   R 000007
SOCK   R 000011
TEST   R 000013
BEGIN  R 000015
.MPY   X 000001
.DIV   X 000002
.DLD   X 000003
.FMP   X 000004
.FDV   X 000005
.FAD   X 000006
.FSB   X 000007
**  NO ERRORS*
```

```
 PAGE 0002 #01
0001                        ASMB,R,B,L,T
0001  00000                       NAM CHECK
0002  00000 000173  AQUAN   OCT 173
0003  00001 000034  VALUE  OCT 34
0004  00002 101325  AQUAT  OCT 101325
0005  00003 061111  DANTE  OCT 61111
0006  00004 000075  .75    OCT 75
0007  00005 000000  .0     OCT 0
0008  00006 054147  .VAL   OCT 54147
0009  00007 050000  .2.5   DEC 2.5
      00010 000004
0010  00011 100000  SOCK   DEC -1.0
      00012 000000
0011  00013 074000  TEST   DEC 3.75
      00014 000004
0012  00015 000000  BEGIN  NOP
0013  00016 062000R       LDA AQUAN
0014  00017 016001X       MPY VALUE
      00020 000001R
0015  00021 102000        HLT
0016  00022 062002R       LDA AQUAT
0017  00023 016001X       MPY DANTE
      00024 000003R
0018  00025 102000        HLT
0019  00026 062004R       LDA .75
0020  00027 016001X       MPY =D20
      00030 000115R
0021  00031 102000        HLT
0022  00032 062006R       LDA .VAL
0023  00033 066005R       LDB .0
0024  00034 016002X       DIV .75
      00035 000004R
0025  00036 102000        HLT
0026  00037 062004R       LDA .75
0027  00040 066005R       LDB .0
0028  00041 016002X       DIV =B75
      00042 000116R
0029  00043 102000        HLT
0030  00044 016003X       DLD .2.5
      00045 000007R
0031  00046 016004X       FMP SOCK
      00047 000011R
0032  00050 102000        HLT
0033  00051 016003X       DLD TEST
      00052 000013R
0034  00053 016004X       FMP =F10.0
      00054 000117R
0035  00055 102000        HLT
0036  00056 016003X       DLD .2.5
      00057 000007R
0037  00060 016005X       FDV SOCK
      00061 000011R
0038  00062 102000        HLT
0039  00063 016003X       DLD TEST
      00064 000013R
0040  00065 016005X       FDV =F2.0
      00066 000121R
0041  00067 102000        HLT
```

```
0042  00070 016003X        DLD .2.5
      00071 000007R
0043  00072 016006X        FAD SOCK
      00073 000011R
0044  00074 102000         HLT
0045  00075 016003X        DLD TEST
      00076 000013R
0046  00077 016006X        FAD =F10.25
      00100 000123R
0047  00101 102000         HLT
0048  00102 016003X        DLD .2.5
      00103 000007R
0049  00104 016007X        FSB SOCK
      00105 000011R
0050  00106 102000         HLT
0051  00107 016003X        DLD TEST
      00110 000013R
0052  00111 016007X        FSB =F3.5
      00112 000125R
0053  00113 102000         HLT
0054  00114 126015R        JMP BEGIN,I
      00115 000024
      00116 000075
      00117 050000
      00120 000010
      00121 040000
      00122 000004
      00123 051000
      00124 000010
      00125 070000
      00126 000004
0055                       END
**  NO ERRORS*
```


START
DEFINE CONSTANTS, STORAGE AREAS, EXTERNALS, ETC.
GENERATE A VALUE
IS VALUE POSITIVE ?
NO
COMPLEMENT AND STORE THE VALUE
YES
STORE THE VALUE
15 VALUES GENERATED, STORED ?
NO
YES
PRINT POSITIVE HEADING
STORE A POSITIVE VALUE IN TEMP
JSB TO CONVERSION ROUTINE
ALL POSITIVE VALUES PRINTED ?
NO
YES
PRINT NEGATIVE HEADING
STORE MINUS SIGN IN OUTPUT AREA
STORE A NEGATIVE VALUE IN TEMP
JSB TO CONVERSION ROUTINE
ALL NEGATIVE VALUES PRINTED ?
YES
END
NO
CONVERSION ROUTINE
CONVERT VALUE IN TEMP TO OCTAL FORMAT
PRINT THE VALUE
RETURN CONTROL TO CALLING ROUTINE

```
0001                    ASMB,R,B,L,T
VALUE  R 000000
COUNR  R 000001
NEGPL  R 000002
POSPL  R 000021
NEGMD  R 000040
NEGAD  R 000041
POSMD  R 000042
POSAD  R 000043
PCOUN  R 000044
NCOUN  R 000045
TEMP   R 000046
.IOC.  X 000001
OUT    R 000047
B        000001
NEG5   R 000050
NEG2   R 000051
HEAD1  R 000052
HEAD2  R 000062
MASK   R 000072
CONST  R 000073
MINUS  R 000074
OUTPT  R 000075
BEGIN  R 000103
LOOP   R 000104
CHECK  R 000114
CAT    R 000117
WRITE  R 000122
NEXTP  R 000133
NEXTN  R 000155
FIN    R 000164
CONVT  R 000165
AGAIN  R 000167
IOCHK  R 000214
**  NO ERRORS*
```

```
 PAGE 0002 #01
0001                    ASMB,R,B,L,T
0001  00000                     NAM .8CNV
0002  00000 154321  VALUE OCT 154321
0003  00001 177761  COUNR DEC -15
0004  00002 000000  NEGPL BSS 15
0005  00021 000000  POSPL BSS 15
0006  00040 000002R NEGMD DEF NEGPL
0007  00041 000002R NEGAD DEF NEGPL
0008  00042 000021R POSMD DEF POSPL
0009  00043 000021R POSAD DEF POSPL
0010  00044 000000  PCOUN BSS 1
0011  00045 000000  NCOUN BSS 1
0012  00046 000000  TEMP  BSS 1
0013                    EXT .IOC.
0014  00047 000100R OUT   DEF OUTPT+3
0015  00001         B     EQU 1
0016  00050 177773  NEG5  DEC -5
0017  00051 177776  NEG2  DEC -2
0018  00052 050117  HEAD1 ASC 8,POSITIVE VALUES
      00053 051511
      00054 052111
      00055 053105
      00056 020126
      00057 040514
      00060 052505
      00061 051440
0019  00062 047105  HEAD2 ASC 8,NEGATIVE VALUES
      00063 043501
      00064 052111
      00065 053105
      00066 020126
      00067 040514
      00070 052505
      00071 051440
0020  00072 000007  MASK  OCT 7
0021  00073 000060  CONST OCT 60
0022  00074 020055  MINUS ASC 1, -
0023  00075 020040  OUTPT ASC 6,
      00076 020040
      00077 020040
      00100 020040
      00101 020040
      00102 020040
0024  00103 000000  BEGIN NOP
0025  00104 062000R LOOP  LDA VALUE   LOAD A WITH VALUE--ROTATE TO
0026  00105 001300        RAR         GENERATE NEW VALUE
0027  00106 072000R       STA VALUE   STORE IN VALUE
0028  00107 002021        SSA,RSS     IS THE VALUE NEGATIVE?
0029  00110 026117R       JMP CAT     NO--JUMP TO CAT
0030  00111 003004        CMA,INA     YES--CONVERT TO TWOS COMPLEMENT
0031  00112 172040R       STA NEGMD,I STORE IN NEGPL AREA
0032  00113 036040R       ISZ NEGMD   INCREMENT INDIRECT ADDRESS
0033  00114 036001R CHECK ISZ COUNR   ALL 15 VALUES GENERATED,STORED?
0034  00115 026104R       JMP LOOP    NO--GO BACK FOR NEXT VALUE
0035  00116 026122R       JMP WRITE   YES--JUMP TO WRITE
0036  00117 172042R CAT   STA POSMD,I STORE POSITIVE VALUE IN POSPL AREA
0037  00120 036042R       ISZ POSMD   INCREMENT INDIRECT ADDRESS
0038  00121 026114R       JMP CHECK   JUMP TO LOCATION CHECK
```

```
0039  00122 016001X WRITE JSB .IOC.    *CALL .IOC.
0040  00123 020006       OCT 20006    *DEFINE OUTPUT, DEVICE, FORMAT
0041  00124 026122R       JMP WRITE    *IF BUSY, KEEP TRYING
0042  00125 000052R       DEF HEAD1    *START OF OUTPUT AREA
0043  00126 000010       DEC 8        *LENGTH OF OUTPUT AREA
0044  00127 062042R       LDA POSMD    LOAD ADDRESS OF LAST POS. VALUE+1
0045  00130 003004       CMA,INA      CONVERT TO TWOS COMPLEMENT
0046  00131 042043R       ADA POSAD    ADD ADDRESS OF FIRST POS. VALUE
0047  00132 072044R       STA PCOUN    STORE -(NO. OF POS. VALS) IN PCOUN
0048  00133 162043R NEXTP LDA POSAD,I LOAD A WITH A POSITIVE VALUE
0049  00134 001200       RAL          POSITION FOR CONVT ROUTINE
0050  00135 072046R       STA TEMP     STORE IN TEMPORARY LOCATION
0051  00136 016165R       JSB CONVT    JUMP TO CONVERT-WRITE ROUTINE
0052  00137 036043R       ISZ POSAD    INCREMENT INDIRECT ADDRESS
0053  00140 036044R       ISZ PCOUN    ALL POSITIVE VALS. PRINTED?
0054  00141 026133R       JMP NEXTP    NO--GO BACK FOR NEXT ONE
0055  00142 016001X       JSB .IOC.    *YES--PRINT NEGATIVE HEADING
0056  00143 020006       OCT 20006    *DEFINE OUTPUT, DEVICE, FORMAT
0057  00144 026142R       JMP *-2      *IF BUSY, KEEP TRYING
0058  00145 000062R       DEF HEAD2    *START OF OUTPUT AREA
0059  00146 000010       DEC 8        *LENGTH OF OUTPUT AREA
0060  00147 062040R       LDA NEGMD    LOAD ADDRESS OF LAST NEG. VALUE+1
0061  00150 003004       CMA,INA      CONVERT TO TWOS COMPLEMENT
0062  00151 042041R       ADA NEGAD    ADD ADDRESS OF FIRST NEG. VALUE
0063  00152 072045R       STA NCOUN    STORE -(NO. OF NEG. VALS) IN NCOUN
0064  00153 062074R       LDA MINUS    LOAD A WITH MINUS SIGN
0065  00154 072077R       STA OUTPT+2  STORE IN OUTPT AREA
0066  00155 162041R NEXTN LDA NEGAD,I LOAD A WITH NEGATIVE VALUE
0067  00156 001200       RAL          POSITION FOR CONVT ROUTINE
0068  00157 072046R       STA TEMP     STORE IN TEMP
0069  00160 016165R       JSB CONVT    JUMP TO CONVERT-WRITE ROUTINE
0070  00161 036041R       ISZ NEGAD    INCREMENT INDIRECT ADDRESS
0071  00162 036045R       ISZ NCOUN    ALL NEGATIVE VALS. PRINTED?
0072  00163 026155R       JMP NEXTN    NO--GO BACK FOR NEXT ONE
0073  00164 126103R FIN   JMP BEGIN    YES--
0074  00165 000000  CONVT  NOP
0075  00166 066050R       LDB NEG5     LOAD B WITH CHARACTER-COUNTER
0076  00167 062046R AGAIN LDA TEMP     LOAD A WITH OCTAL QUANTITY
0077  00170 001723       ALF,RAR      POSITION (NEXT) DIGIT
0078  00171 072046R       STA TEMP     RESTORE IN TEMP
0079  00172 012072R       AND MASK     MASK OUT ALL BUT ONE DIGIT
0080  00173 032073R       IOR CONST    MAKE ASCII CHARACTER
0081  00174 004010       SLB          IS THIS 1ST, 3RD, OR 5TH DIGIT?
0082  00175 001727       ALF,ALF      YES--LEFT JUSTIFY
0083  00176 132047R       IOR OUT,I    MERGE WITH 2ND/4TH DIGIT OR BLANK
0084  00177 172047R       STA OUT,I    STORE IN OUTPT AREA
0085  00200 006011       SLB,RSS      IS THIS 1ST, 3RD, OR 5TH DIGIT?
0086  00201 036047R       ISZ OUT      NO--INCREMENT INDIRECT ADDRESS
0087  00202 034001       ISZ B        ALL OCTAL CHARACTERS PROCESSED?
0088  00203 026167R       JMP AGAIN    NO--GO BACK FOR NEXT ONE
0089  00204 016001X       JSB .IOC.    YES--CALL .IOC. TO WRITE
0090  00205 020006       OCT 20006    *DEFINE OUTPUT, DEVICE, FORMAT
0091  00206 026204R       JMP *-2      *IF BUSY, KEEP TRYING
0092  00207 000075R       DEF OUTPT    *START OF OUTPUT AREA
0093  00210 177765       DEC -11      *LENGTH OF OUTPUT AREA
0094  00211 062047R       LDA OUT      RE-INITIALIZE THE
0095  00212 042051R       ADA NEG2          INDIRECT
0096  00213 072047R       STA OUT           ADDRESS
```

```
PAGE 0004 #02

0097  00214 016001X IOCHK JSB .IOC.    *CHECK THE STATUS
0098  00215 040006        OCT 40006    *      OF THE TELEPRINTER
0099  00216 002020        SSA          BUSY?
0100  00217 026214R       JMP IOCHK    YES--KEEP TESTING
0101  00220 062075R       LDA OUTPT    NO--RE-INITIALIZE
0102  00221 072100R       STA OUTPT+3       THE OUTPT
0103  00222 072101R       STA OUTPT+4       AREA TO
0104  00223 072102R       STA OUTPT+5       ALL BLANKS
0105  00224 126165R       JMP CONVT,I RETURN CONTROL TO CALLING ROUTINE
0106                      END
**  NO ERRORS*
```

```
POSITIVE VALUES
      73064
      35432
      16615
      43543
      64354
      32166
      15073
      61507
NEGATIVE VALUES
     -11630
     -70472
     -56117
     -27050
     -71343
     -34562
     -47135
```

REVIEW ANSWERS A

Introduction and Chapter 1

1. Bits
2. Assemblers and compilers
3. Radix, or base, and modulus
4. (a) 2 (b) 8 (c) 10
5. "Complement" may be defined as a method of representing a negative number in the computer.
6. (a) 55_{10}; (b) 100111000_2; (c) 16752_8; (d) 13823_{10};
   (e) 1000000000_2; (f) 476_{10}; (g) 616.6_8; (h) 191.010439_{10}
7. (a) 11110010_2; (b) 33345_8; (c) 2123_8; (d) 101011_2;
   (e) 1110011_2; (f) 367_8
8. (a) 10_2; (b) 401_8; (c) 2346_{10}; (d) 471_{10};
   (e) 10101011_2; (f) 676767_8

Chapter 2

1. Words
2. Addresses
3. Instructions
4. Memory A-register
5. "Overflow" may be defined as the condition arising when an operation produces a result larger than can be contained in a computer register or word.
6. Program
7. 128

Chapter 3

1. Memory reference instructions, register reference instructions, input/output instructions.
2. Pages
3. The division of memory into pages is based upon the 10-bit address field of the memory reference instructions. $2^{10} = 1,024$.
4. Zero (base) page or the current page.
5. The D/I bit = 1.
6. 32,768
7. Central processor (computer) and the external I/O devices.
8. Control bit, flag bit, and channel buffer.
9. The slot in which the interface card for the device is placed.

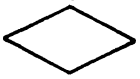
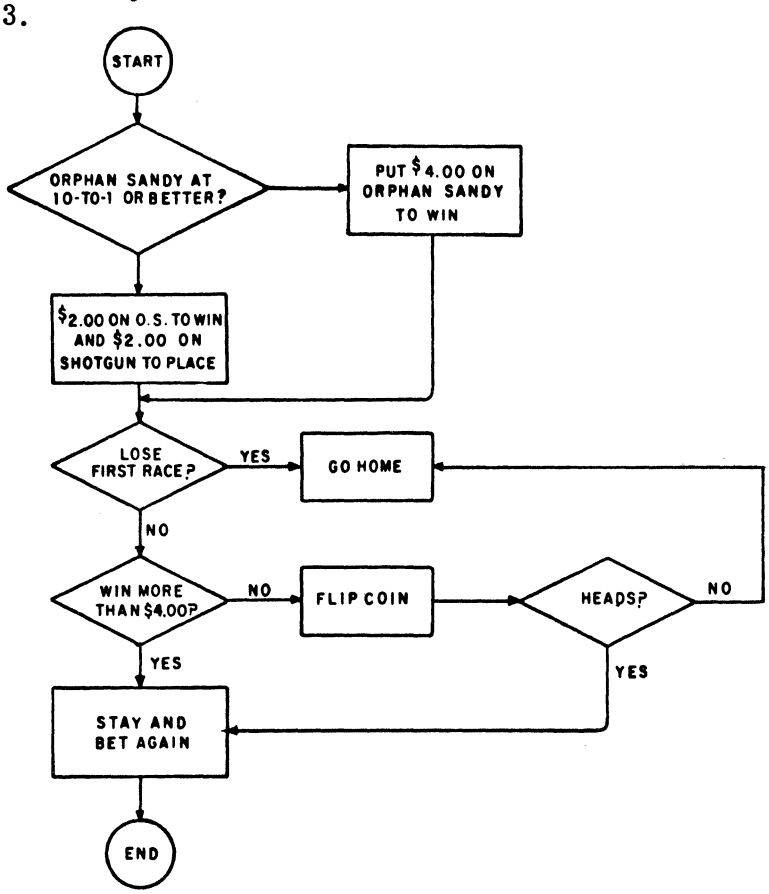
Chapter 4

1. Source, object
2. Machine instructions and pseudo instructions.
3. Symbol table
4. An absolute program is one whose addresses are translated permanently and are not modified as a result of loading at object program execution time.
5. A relocatable program is assigned relative addresses at assembly time which are modified as a result of loading at object program execution time.
6. A "pass" is defined as one Assembler examination of the source code.
7. Two or three passes are required to complete an assembly, depending on the assembly output selected and the number of devices available for the output.

Chapter 5

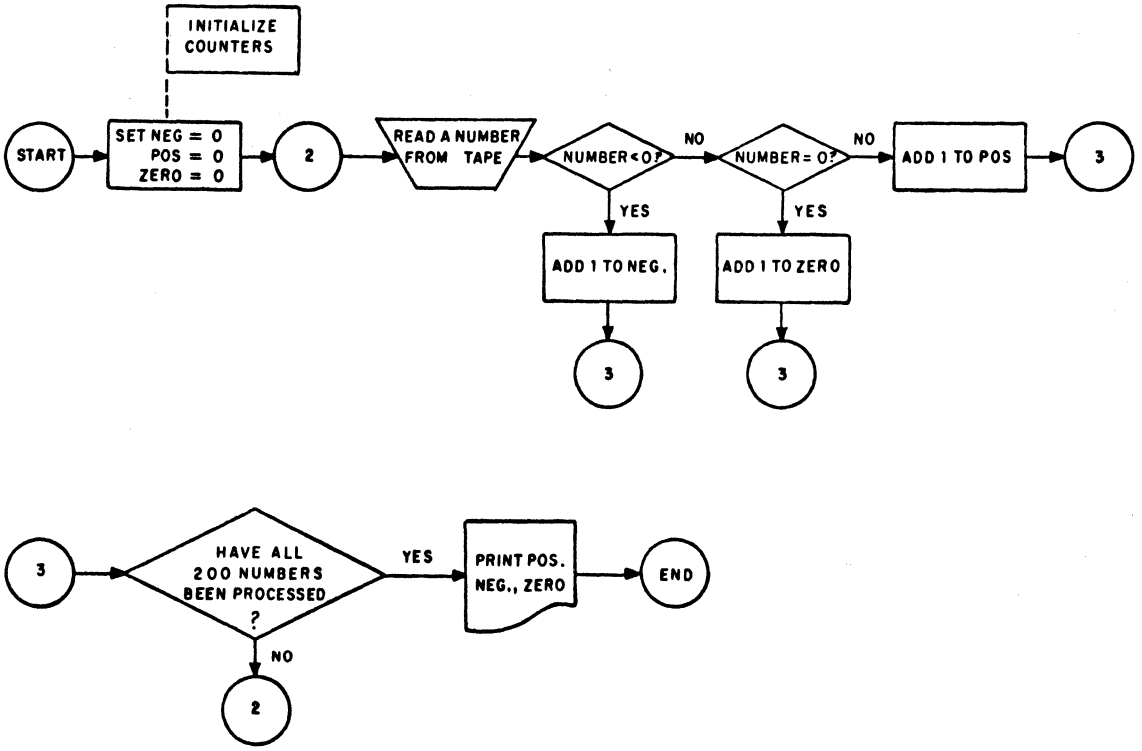
1. Method of loading programs, simplified input/output, debugging aids.
2. The Basic Binary Loader, which loads absolute programs, and the Relocating Loader, which loads relocatable programs.
3. The Relocating Loader.
4. Data which is loaded is intended to be executed; data which is read or written is to be acted upon.
5. "Debugging" is a term used to mean program error detection and correction.

Chapter 6

1. Analysis
2.  represents a decision
3. 

Chapter 6 (Cont'd)

4.



Chapter 7

1. Label, Op Code, Operand, Comments
2. 5 characters
3. 1 - 9, A - Z, and the period (.).
4. Leave character position one blank.
5. Asterisk in character position 1 followed by remarks.
6. (c), (e), (f)
7. CR LF carriage return and line feed.
8. (a) $32,767_{10}$ or 177777_8; (b) 1024_{10} or 1777_8; (c) 64_{10} or 77_8
9. B
10. Permits reference to the value of the program location counter at the time the statement is encountered.
11. (a), (c), (e), (h)

Chapter 8

1. When the contents of A and the contents of the specified address are not equal.
2. When the contents of the specified address plus one is equal to zero.
3. 130_8
4. 40_8
5. TAG+49
6. Four shift-rotate instructions may be combined; eight alter-skip instructions may be combined.
7. (a) Instructions for A- and B-register are combined.
   (b) Shift-rotate and alter-skip instructions are combined.
   (c) Instructions are out of order.
   (d) Comma separating instructions omitted.
8. NOP
9. STF 0
10. The executing program is interrupted and control is transferred to the interrupt location for the channel causing the interrupt.
11. (c) a character
12. STC, CLC
13. 003770_8

The answers to the coding problems, shown below, represent one possible solution for a problem. There may be other solutions, equally valid.

14.
```
        LDA  CAT
        ADA  DOG
        STA  SUM
```

15.
```
        LDA  Y
        ADA  Z
        CPA  Q
        JMP  UNEQ
        ADA  W
        STA  R1
        JMP  STOP
UNEQ    ADA  W
        ADA  Q
        STA  R2
        JMP  STOP
```

Chapter 8 (Cont'd)

16. MASK = 0001050_8

```
    LDA  TEST
    AND  MASK
    CPA  MASK
    JMP  ON
    JUM  OFF
```

17.
```
    LDA  Y                    ODD  NOP
    CMA, INA                       SSB, RSS
    LDB  X                         JMP  ODD, I
    ADB  ØØØØØ                     CMB, INB
    SLB, RSS                       JMP  ODD, I
    JSB  ODD
```

Chapter 9

1. (b), (d), (e), and (f) are invalid
2. If used as the first statement in the program or if the operand field contains an absolute expression.
3. It provides the loader with the starting address of the object program to which the loader transfers control.
4. (b) should be loaded first because the maximum size of the common area is determined by the COM statement which is loaded first.
5. In PROGA, the pseudo EXT CAT must be specified.
   In PROGB, the pseudo ENT CAT must be specified.
6. DEF
7. 56 characters

The following answers represent one possible solution to the problem. There may be other equally valid solutions.

Chapter 9 (Cont'd)

8.
```
POSNG   NOP
LOOP    LDA     POSA,I
        XOR     MASK
        STA     NEGA,I
        ISZ     NEGA,I
        ISZ     POSA
        ISZ     NEGA
        ISZ     COUNT
        JMP     LOOP
        JMP     POSNG,I
POSA    DEF     POS
NEGA    DEF     NEG
COUNT   DEC     -25
MASK    OCT     177777
        COM     POS (25)
NEG     BSS     25
```

9.
```
REV     NOP
LOOP    LDA     TABA,I
        STA     TAGA,I
        ISZ     TABA
        LDA     TAGA
        ADA     DECRM
        STA     TAGA
        ISZ     COUNT
        JMP     LOOP
        JMP     REV,I
TABA    DEF     TAB
TAGA    DEF     TAG+49
DECRM   DEC     -1
        COM     TAB (50)
TAG     BSS     50
```

Chapter 10

1. .IOC
2. EXT
3. drivers
4. equipment table
5. Clear

6.
```
        JSB   .IOC.
        OCT   010010
        JMP   ERROR
        DEF   BUFA
        OCT   12
              :
              :
        EXT   .IOC.
BUFA    BSS   10
```

7.
```
ERROR   SSB
        JMP   DVBUS
        JSB   .IOC.
        OCT   020002
        JMP   HALT
        DEF   D1
        DEC   6
        JMP   HALT
DVBUS   JSB   .IOC.
        OCT   020002
        JMP   HALT
        DEF   D2
        DEC   5
HALT    JSB   .IOC.
        OCT   040002
        SSA
        JMP   *-3
        HLT
D1      ASC   6,ILL FN ON TR
D2      ASC   5,TR DV BSY
        EXT   .IOC.
        ENT   ERROR
```

ASCII CHARACTER FORMAT B

b_7				0	0	0	0	1	1	1	1
b_6				0	0	1	1	0	0	1	1
b_5				0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1								
0	0	0	0	NULL	DC_0	b	0	@	P		
0	0	0	1	SOM	DC_1	!	1	A	Q		
0	0	1	0	EOA	DC_2	"	2	B	R		
0	0	1	1	EOM	DC_3	#	3	C	S		
0	1	0	0	EOT	DC_4 (STOP)	$	4	D	T		
0	1	0	1	WRU	ERR	%	5	E	U		
0	1	1	0	RU	SYNC	&	6	F	V		
0	1	1	1	BELL	LEM	(APOS)	7	G	W		
1	0	0	0	FE_0	S_0	(	8	H	X		
1	0	0	1	HT/SK	S_1	)	9	I	Y		
1	0	1	0	LF	S_2	*	:	J	Z		
1	0	1	1	V_{TAB}	S_3	+	;	K	[		
1	1	0	0	FF	S_4	(COMMA)	<	L	\		ACK
1	1	0	1	CR	S_5	−	=	M	]		①
1	1	1	0	SO	S_6	.	>	N	↑		ESC
1	1	1	1	SI	S_7	/	?	O	←		DEL

UNASSIGNED — UNASSIGNED

Standard 7-bit set code positional order and notation are shown below with b_7 the high-order and b_1 the low-order, bit position.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1
EXAMPLE: The code for "R" is:	1	0	1	0	0	1	0

LEGEND

NULL	Null/Idle
SOM	Start of message
EOA	End of address
EOM	End of message
EOT	End of transmission
WRU	"Who are you?"
RU	"Are you...?"
BELL	Audible signal
FE_0	Format effector
HT	Horizontal tabulation
SK	Skip (punched card)
LF	Line feed
V_{TAB}	Vertical tabulation
FF	Form feed
CR	Carriage return
SO	Shift out
SI	Shift in
DC_0	Device control reserved for data link escape
DC_1-DC_3	Device Control
DC_4(Stop)	Device control (stop)
ERR	Error
SYNC	Synchronous idle
LEM	Logical end of media
S_0-S_7	Separator (information)
b	Word separator (space, normally non-printing)
<	Less than
>	Greater than
↑	Up arrow (Exponentiation)
←	Left arrow (Implies/Replaced by)
\	Reverse slant
ACK	Acknowledge
①	Unassigned control
ESC	Escape
DEL	Delete/Idle

BINARY CODED DECIMAL FORMAT C

Kennedy 1406/1506 ASCII-BCD Conversion

Symbol	BCD (octal code)	ASCII Equivalent (octal code)	Symbol	BCD (octal code)	ASCII Equivalent (octal code)
(Space)	2Ø	Ø4Ø	A	61	1Ø1
!	52	Ø41	B	62	1Ø2
#	13	Ø43	C	63	1Ø3
$	53	Ø44	D	64	1Ø4
%	34	Ø45	E	65	1Ø5
&	6Ø	Ø46	F	66	1Ø6
'	14	Ø47	G	67	1Ø7
(	34	Ø50	H	7Ø	11Ø
)	74	Ø51	I	71	111
*	54	Ø52	J	41	112
+	6Ø	Ø53	K	42	113
,	33	Ø54	L	43	114
-	4Ø	Ø55	M	44	115
.	73	Ø56	N	45	116
/	21	Ø57	O	46	117
			P	47	12Ø
Ø	12	Ø6Ø	Q	50	121
1	Ø1	Ø61	R	51	122
2	Ø2	Ø62	S	22	123
3	Ø3	Ø63	T	23	124
4	Ø4	Ø64	U	24	125
5	Ø5	Ø65	V	25	126
6	Ø6	Ø66	W	26	127
7	Ø7	Ø67	X	27	13Ø
8	1Ø	Ø7Ø	Y	30	131
9	11	Ø71	Z	31	132
:	15	Ø72	[	75	133
;	56	Ø73	\	36	134
<	76	Ø74	]	55	135
=	13	Ø75			
>	16	Ø76			
?	72	Ø77			
@	14	1ØØ			

Other symbols which may be represented in ASCII are converted to spaces in BCD (20)

HP 2020A/B ASCII - BCD Conversion

Symbol	ASCII (Octal code)	BCD (Octal code)	Symbol	ASCII (Octal code)	BCD (Octal code)
(Space)	4Ø	2Ø	A	1Ø1	61
!	41	52	B	1Ø2	62
"	42	37	C	1Ø3	63
#	43	13	D	1Ø4	64
$	44	53	E	1Ø5	65
%	45	34	F	1Ø6	66
&	46	60 †	G	1Ø7	67
'	47	36	H	11Ø	70
(	5Ø	75	I	111	71
)	51	55	J	112	41
*	52	54	K	113	42
+	53	6Ø	L	114	43
,	54	33	M	115	44
-	55	4Ø	N	116	45
.	56	73	O	117	46
/	57	21	P	12Ø	47
			Q	121	50
Ø	6Ø	12	R	122	51
1	61	Ø1	S	123	22
2	62	Ø2	T	124	23
3	63	Ø3	U	125	24
4	64	Ø4	V	126	25
5	65	Ø5	W	127	26
6	66	Ø6	X	13Ø	27
7	67	Ø7	Y	131	30
8	7Ø	1Ø	Z	132	31
9	71	11			
			[	133	75 ‡
:	72	15	]	135	55 ‡
;	73	56	↑	136	77
<	74	76	←	137	32
=	75	35			
>	76	16			
?	77	72			
@	1ØØ	14			

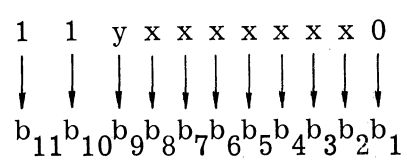
† BCD code of 60 always converted to ASCII code 53 (+).

‡ BCD code of 75 always converted to ASCII code 50 (() and BCD code of 55 always converted to ASCII code 51 ()).

The following list contains HP devices which may be connected to the HP 2116 , the size and format of the element transferred, and sample input/output subroutines which transfer one or more elements of data, assuming interrupt system disabled.

HP 2752A TELEPRINTER

The teleprinter transfers an 11-bit element; bits are transmitted in serial fashion, one bit about every 9.1 msec. Each bit is transferred into bit 7 of the A- or B-register, and out from bit 0 of the A- or B-register. The flag bit is set after transmission of each bit. The format of an element is as follows:



where the 1's and 0 are control characters meaningful to the device and x's are the binary representation of an ASCII character. The bits are transferred in order from b_1 to b_{11}.

Sample coding, input:

```
READ    NOP
        LDA     K1          Set counter to -11.
        STA     CTR
        CLA                 Clear the A-register.
        STC     TELIN,C     Enables element transfer (sets
                            control bit)†.
A       RAR                 Rotate A right one bit.
        SFS     13B         Is transfer of one bit completed?
        JMP     *-1         No--keep testing.
```

† The HP 2752A will actually begin to input a character when (1) a key is punched (2) tape is placed in HP 2752A punched tape reader and switch is moved to START position. The flag will not be set until either of the former actions takes place and the transfer of one element is completed.

```
         MIA     TELIN,C   Yes--place bit in A-register,
                           clear flag so that test for com-
                           pletion of transfer of next bit is
                           valid.
         ISZ     CTR       Have all 11 bits been trans-
                           ferred?
         JMP     A         No--return for next bit.
         RAL,RAL           Yes--rotate A left two bits.
         AND     B         Mask out control bits and 8th
                           character bit.
         STA     CHAR      Store the element.
         CLC     TELIN     Turn off the device.
         JMP     READ,I    Exit from the routine.
CTR      BSS     1
K1       OCT     177765
B        OCT     000177
         COM     CHAR
TELIN    EQU     13B
```

Sample coding, output:

```
TYPE     NOP
         LDA     K1        Set counter to -11.
         STA     CTR
         LDA     CHAR      Load A-register with ASCII
                           character to be output.
         ALS               Shift left one bit (get 0 control
                           bit).
         IOR     C         Add two 1-bits as control bits.
         STC     TLOUT†    Enables data output (sets control
                           bit).
D        SFS     TLOUT     Is transfer of one bit completed?
         JMP     D         No--keep testing.
         OTA     TLOUT,C   Yes--output next bit and clear
                           flag so that test for completion
                           of transfer of next bit from buf-
                           fer to device is valid.
```

† Note that, C is not specified for the STC. The flag bit will be set at the time this instruction is executed--the flag bit is set automatically when the device is turned on-line, and the routine leaves the flag set when it exits. Thus, the first time the loop from D to the ISZ is executed, control will pass to the OTA. Otherwise, if C were specified and the flag cleared, the two instructions beginning at D would be executed indefinitely.

```
        RAR             Position next bit for transfer.
        ISZ     CTR     Have all 11 bits been trans-
                        ferred?
        JMP     D       No--transfer next bit.
        CLC     TLOUT   Yes--turn off device.
        JMP     TYPE,I  Exit from subroutine.

C       OCT     003000
        COM     CHAR
TLOUT   EQU     12B
```

HP 2754A TELEPRINTER

Same as HP 2752A Teleprinter, above.

HP 2737A PUNCHED TAPE READER

The punched tape reader transfers an 8-bit element to bits 7-0 of the A- or B-register. The format is as follows:

Paper tape track 1 ⟶ bit 0
Paper tape track 2 ⟶ bit 1
.
.
.
Paper tape track 8 ⟶ bit 7

The binary representation of an ASCII character is transferred to bits 6-0; the eighth track, going to bit 7, is always zero.

Sample coding (assume select code 10_8):

```
READ    NOP
        STC     PTRD,C  Set control bit, clear flag.
        SFS     PTRD    Transfer of element complete?
        JMP     *-1     No--keep testing.
        LIA     PTRD    Yes--place the element in the
                        A-register.
        STA     CHAR    Store the element.
        CLC     PTRD    Turn off the device.
        JMP     READ,I  Exit from the routine.
        COM     CHAR
```

HP 2737B PUNCHED TAPE READER-SPOOLER

I/O is the same as for the HP 2737A Punched Tape Reader, above.

HP 2753A TAPE PUNCH

The tape punch transfers an 8-bit element from bits 7-0 of the A- or B-register. The format is as follows:

bit 0 ⟶ Paper tape track 1
bit 1 ⟶ Paper tape track 2
.
.
.
bit 7 ⟶ Paper tape track 8

The binary representation of an ASCII character is transferred from bits 6-0; bit 7, going to the eighth track, is always zero.

Sample coding:

```
WRITE   NOP
        LDA     CHAR      Load A with element to be output.
        OTA     TPNCH     Transfer element to channel buf-
                          fer and clear flag.
        STC     TPNCH     Output element to device.
        SFS     TPNCH     Is transfer from buffer to de-
                          vice complete?
        JMP     *-1       No--keep testing.
        CLC     TPNCH     Yes--turn off device.
        JMP     WRITE,I   Exit from routine.
        COM     CHAR
TPNCH   EQU     11B
```

HP 2401C AND HP 3460A DIGITAL VOLTMETERS

The HP 2401C and 3460A Integrating Digital Voltmeters provide data through the Digital Voltmeter Data Interface. Data is requested by the user's program through the Digital Voltmeter Programmer and the Crossbar Scanner Programmer.

Digital Voltmeter Programmer

Data is output to the Digital Voltmeter Programmer as an 8-bit element from bits 7-0 of the A- or B-register.

The element, in effect, tells the voltmeter the sample period, the type of reading to be taken, and the range. The format is as follows:

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	Provides
					0	0	0	Autorange
					0	0	1	+10 Gain, 2411A
					0	1	0	0.1V Range
					0	1	1	1V Range
					1	0	0	10V Range
					1	0	1	100V Range
					1	1	0	1000V Range
					1	1	1	10 Megohm Range
		0	0	0				AC Normal
		0	0	1				AC Fast
		0	1	0				Frequency
		0	1	1				Period
		1	0	0				DC Volts
		1	0	1				Ohms
0	0							1 Sec. Sample Period
0	1							0.1 Sec. Sample Period
1	0							0.01 Sec. Sample Period

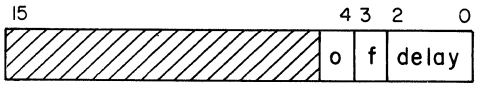
Crossbar Scanner Programmer

An STC for the Crossbar Scanner Programmer initiates a reading on the voltmeter.† Information may be transferred to the Crossbar Scanner Programmer with OTA/OTB, giving information as to the type of reading to be taken, without providing an STC.

A sixteen-bit element is output to the Crossbar Scanner Programmer; however, not all the bits are significant. When the

† An STC causes the channel to be selected which in turn sends an encode after the channel is reached.

flag bit is set to zero, the element supplies a delay time and type of measurement:



o = f = o Signifies volts measurement

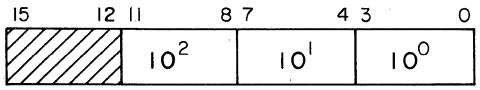
o = 1 Signifies ohms measurement

f = 1 Signifies frequency measurement

delay Signifies delay before measurement

value	delay
000	15 msec.
001	17.5 msec.
010	22 msec.
011	27 msec.
100	42 msec.
101	62 msec.
110	145 msec.
111	500 msec.

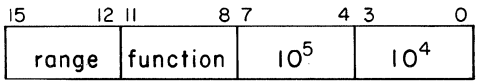
When the flag bit is set to 1, the element supplies 3 digit BCD identification of which channel is to be read from the 2911A (Crossbar Scanner). The channel is automatically incremented by each successive STC instruction for the Crossbar Scanner Programmer after the first STC.



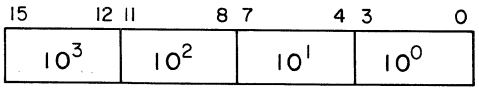
Digital Voltmeter Data Interface

The Digital Voltmeter Data Interface provides the reading from the HP 2401C or 3460A Integrating Digital Voltmeter in the form of a 32-bit element:

First word:



Second word:



range — decimal 10^{-n} multiplier

function — the type of reading:

Function	8-4-2-1 Code
Period	0 0 0 0
+Vdc	0 0 0 1
-Vdc	0 0 1 0
kHz	0 0 1 1
k	0 1 0 0
m	0 1 0 1
Overload	1 0 0 1
Vac	1 0 1 1

10^5-10^0 — A six BCD digit value

Sample coding--reading 200 inputs on the HP 2401C:

```
RDVLT   NOP
        LDA     RDNG        Load A with data for Digital
                            Voltmeter Programmer.
        OTA     DVMPR       Set up voltmeter to take reading
                            of DC volts.
        LDA     DELAY       Load A with ohms/frequency/
                            delay indicator; a delay of 27
                            msec, volts to be measured.
        LDB     INP         Load B with channel identifica-
                            tion for first measurement.
        CLF     SCANR       Clear flag and output ohms/fre-
        OTA     SCANR       quency/delay indicator.
        STF     SCANR       Set flag to enable output of chan-
                            nel identification.
        OTB     SCANR       Output the channel identification.
LOOP    STC     DVMDI,C     Ready the Data Interface, clear
                            flag so it will indicate when read-
                            ing has been taken.
        STC     SCANR       Set control bit; initiate the mea-
                            surement. The channel identi-
                            fication is automatically incre-
                            mented at each successive STC.
        SFS     DVMDI       Has reading been taken?
        JMP     *-1         No--keep testing.
        LIB     DVMDI       Yes--load B with first word.
        LIA     DVMDI       Load A with second word.
        DST     VPLC,I      Store the reading.
        ISZ     VPLC        Modify storage address.
        ISZ     VPLC
        ISZ     CNTR        All 200 readings been taken?
        JMP     LOOP        No--return for next input.
        CLC     DVMDI       Yes--turn off data interface de-
                            vice.
        JMP     RDVLT,I     Exit from routine.
```

```
CNTR    DEC     -200
DVMPR   EQU     17B
SCANR   EQU     20B
DVMDI   EQU     21B
RDNG    OCT     144
DELAY   OCT     3
INP     OCT     1
VLPC    DEF VSTOR
        COM VSTOR (400)
```

KENNEDY 1406 AND 1506 INCREMENTAL TAPE TRANSPORTS

The Kennedy 1406 and 1506 Incremental Tape Transports record BCD data at 200 bpi at a recording speed of 0 to 400 characters per second.

The following commands are available:

Octal Value Bits 15-14	Command
0	Write (step)
1	Write file gap
2	Write record gap
3	Write file gap

A data character is transferred from bits 5-0 of the A- or B-register.

Status bits may be transferred from the buffer to bits 6, 5, 3 and 1 of the A- or B-register. They are as follows:

b_0 busy	This bit is one when the unit is busy. When zero, the unit is ready to accept a command.
b_3 broken tape	There is no tape on the write head. This bit is zero when the tape is rethreaded.
b_5 end-of-tape	This bit is set to one when the end-of-tape reflective marker is sensed. It remains set until the tape is rewound (manual control).

b_6 load point	The start-of-tape marker has been sensed. This bit is set only when this marker is opposite the photosensor.

Sample coding:

The following are samples for writing two characters on tape, writing a record gap and a file gap on tape, and for testing status. The unit is on channel 21.

a. Write (step):

```
WRIT1   LDA   CHAR      Load character with zero com-
        OTA   INCTP     mand bits in A-register, output
        STC   INCTP,C   A to buffer. Set control bit,
        SFS   INCTP     clear flag bit. Test if charac-
        JMP   *-1       ter written; then write second
WRIT2   LDA   CHAR+1    character.
        OTA   INCTP
        STC   INCTP,C
          .
          .
          .
INCTP   EQU   21B       BCD characters: HP. Bits 15
CHAR    OCT   7Ø        and 16 of each are zeros (write
        OCT   47        command).
```

b. Write record gap:

```
WRTRG   CLA             Clear A-register
        CCE             Set E-register to one
        ERA             Rotate 1-bit into bit 15 of A
        OTA   INCTP     (creating write record gap com-
        STC   INCTP,C   mand). Output command to buf-
          .             fer, set control bit, and clear
          .             flag.
          .
INCTP   EQU   21B
```

c. Write file gap:

```
WRTFG   CLA             Clear A-register
        CCE             Set E-register to one
        ERA,ERA         Rotate 1-bits into bits 15 and 14
        OTA   INCTP     of A (creating write file gap
        STC   INCTP,C   command). Output command to
          .             buffer, set control bit, and clear
          .             flag.
          .
INCTP   EQU   21B
```

d. Test status:

```
CKST    LIA   INCTP     Load status bits into A-register.
        SZA,RSS         Any bits set?
        JMP   (cont.)
        SLA,RSS         Bit 0 = 1 ?
        JMP   (cont.)
        RAR,RAR
        RAR             Bit 3 = 1 ?
        SLA,RSS
        JMP   (cont.)
        RAR,RAR         Bit 5 = 1 ?
        SLA,RSS
        JMP   (cont.)
          .
          .
          .
```

HP 2020A/B MAGNETIC TAPE UNIT

The HP 2020A/B Magnetic Tape Unit is operated through two channels, a command channel anda data channel. Requests and status information are relayed through the command channel; data is transferred through the data channel.

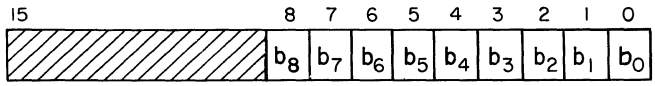
Command Channel

A request element is transferred from bits 7-0 of the A- or B-register:

Octal Value Bits 7-0	Command
071	Write record, odd parity (Binary)
031	Write record, even parity (BCD)
015	Write end-of-file gap
063	Read record, odd parity (Binary)
023	Read record, even parity (BCD)
003	Forward space record
101	Backspace record
201	Rewind
241	Rewind and unload
300	Clear

The flag bit is set on the command channel when any tape motion operation is completed.

A status element is transferred from the command channel to bits 7-0 of the A- or B-register:



b_0 busy	This bit is one when the tape is in motion or the transport is in local status. When zero, the tape unit is ready to accept a command.
b_1 parity error	This bit is set to one if a vertical or longitudinal parity error occurs during a read or write operation. Parity is not checked on forward space record and backspace record operations.
b_2 write not enabled	This bit is one when either the tape reel does not have a write enable ring or the tape unit is rewinding.
b_3 reject	A command will be rejected (ignored) and this bit set if: (1) Tape motion is required and the unit is busy. (2) Backward tape motion is required and the tape is at load point. If a rewind and unload command is given while the tape is at load point, the command will be ignored but the reject bit will not be set. (3) A write command is given and the tape reel does not have a write enable ring.
b_4 timing	This bit is set if the data channel flag has not been cleared or the interrupt request not acknowledged between data interrupt requests while reading or writing.

b_5	end-of-tape	This bit is set when the end-of-tape reflective marker is sensed while the tape is moving forward. It remains set until a rewind command is given.
b_6	start-of-tape	This bit is one when the start of tape marker is under the photo sense head.
b_7	end-of-file	This bit is set to one when a one-character tape mark (17_8) record is detected while reading, forward spacing or backspacing.
b_8	local	The device is in local status.

The parity error, reject, timing, and end-of-file bits are reset when a command resulting in tape motion is accepted. The busy and start-of-tape bits are reset when the condition is no longer true.

Data Channel

Data is transferred between the data channel and bits 5-0 of the A- or B-register as a 6-bit element. Parity is generated (output) or checked (input) as requested by the command channel element.

The flag bit is set on the data channel when one character has been transferred between the data channel buffer and the tape device. Clearing the control bit on a write even or odd parity operation causes an end-of-record gap to be generated.

Sample coding, input one record:

```
RDTAP   NOP
        LIA     CC      Load status element.
        SLA             Test busy bit--unit busy?
        JMP     *-2     Yes--keep testing.
        CLF     DC      Clear flag on data channel so
                        test for completion of transfer
                        of element is valid.
        LDA     RRE     Set command--read even parity.
NEXT    OTA     CC,C    Output command, clear flag so
        SFC     CC      test for completion of transfer
        JMP     X       of record is valid.
        SFS     DC      One element transferred?
        JMP     *-3     No--keep testing.
```

```
        LIA     DC,C    Yes, load character, clear flag
                        so test for completion of transfer
                        of the record is valid.
        STA     BUF,I   Store element.
        ISZ     BUF     Modify storage address.
                        Has complete record been read?
        JMP     NEXT    No--go back for next element.
X       JSB     SCHEK   Yes--transfer control to rou-
                        tine to check status word for
                        parity errors, etc.
        JMP     RDTAP,I Exit from routine.
CC      EQU     15B
DC      EQU     16B
RRE     OCT     23
BUF     DEF     BUFR
        COM     BUFR(50) The programmer should be
                        aware of the size of the records
                        being read so that an adequate
                        input buffer area can be pro-
                        vided.
```

I/O RECORD FORMATS E

ASCII RECORDS (PAPER TAPE)

An ASCII record is a group of characters terminated by an end-of-record mark, consisting of a carriage return, (CR) , and a line feed, (LF) .

For an input operation, the length of the record transmitted to the buffer is the number of characters or words designated in the request, or less if an end-of-record mark is encountered before the character or word count is exhausted. The codes for (CR) and (LF) are not transmitted to the buffer. An end-of-record mark preceding the first data character is ignored.

For an output operation, the length of the record is determined by the number of characters or words designated in the request. An end-of-record mark is supplied at the end of each output record by the input/output system.

If the last character of an output record is←, however, the end-of-record mark is omitted. This allows control of Teleprinter line spacing. The user may write a message (the←is not printed) and expect the reply to be typed on the same line. The reply must be terminated with the (CR) (LF) .

If, a (RUB OUT) code† followed by a (CR) (LF) is encountered on input from the Teleprinter or Punched Tape Reader, the current record is ignored (deleted) and the next record transmitted.

If less than ten feed frames (all zeros) are encountered before the first data character from the Punched Tape Reader, they are ignored. Ten feed frames are interpreted as an end-of-tape condition.

† (RUB OUT) which appears on the Teleprinter keyboard is synonymous with the ASCII symbol, (DEL) .

BINARY RECORDS (PAPER TAPE)

A binary record is transmitted exactly as it appears in memory or on an 8-level paper tape. The record length is determined by the number of characters or words in the buffer, as designated in the request.

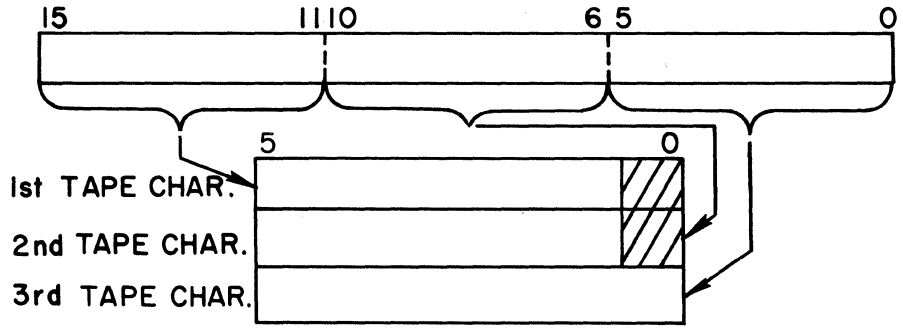
Binary input records may be specified as variable in length. The first word of the record contains a number in bits 15-8 specifying the length of the record in words, including the first word. The entire record, including the word count, is transmitted to the buffer. If the actual length exceeds the size of the buffer, only the number of words equivalent to the buffer length is transmitted.

On input operations, less than ten feed frames preceding the first data character are ignored. Ten feed frames are interpreted as an end-of-tape condition. On output, the system writes four feed frames to serve as a physical record separator.

BINARY RECORDS (MAGNETIC TAPE)

A binary record on magnetic tape is a group of 6-level tape "characters" recorded in odd parity and terminated by a record gap.† The record length is determined by the number of characters or words in the buffer as designated in the request.

Each computer word is translated into three tape "characters" (and vice versa) as follows:



† Odd parity: a seventh bit is recorded on tape if the total of the bits in the six levels is an even number.
Even parity: a seventh bit is recorded on tape if the total of the bits in the six levels is an odd number.

For output operations on the HP 2020, the minimum buffer length is three computer words. If less are specified, zeros are supplied to fill a three word record.

BINARY CODED DECIMAL RECORDS

A BCD record on magnetic tape is a group of BCD characters recorded in even parity and terminated by a record gap. (See Appendix C for BCD character set.) A request to write a BCD record results in the translation of each 7-level ASCII character in the buffer area into a 6-level BCD character on magnetic tape. The translation process does not alter the original contents of the buffer. A request to read a BCD record results in the translation of each BCD character into an ASCII character after the block has been read.

The length of the record is determined by the number of characters or words designated in the request. A record gap is supplied at the end of each record by the input/output system. For an Incremental Magnetic Tape operations, the record gap is omitted if the last character in the buffer is a ← ; the ← is not written on tape.

A WRITE request for the Incremental Magnetic Tape specifying a buffer length of zero causes a record gap only to be written.

For the HP 2020 Magnetic Tape Unit, the maximum record length is 120 tape characters or 120 ASCII characters. If a buffer is specified greater then 120 characters, the first 120 are transmitted and the remaining characters are skipped. For output operations, the minimum buffer length is 7 characters. If less are specified, spaces are supplied to fill to 7 characters.

INPUT/OUTPUT FORMATS FOR INSTRUMENT REQUESTS

Use of the Data Source Interface driver subroutine requires the specification of a "dummy" buffer for an binary output (removal of "hold-off") operation and either a two-or-eight-word buffer for an input operation. If a Read Binary operation is requested, the 32 bits (8 BCD digits) of information are read directly into the two-word buffer.† If a Read ASCII operation is performed, the 8 BCD digits are converted into 16 ASCII characters in the following format:

r f d_5 d_4 d_3 d_2 d_1 d_0 E - s s ^ ^ g g

r	range - a negative power of 10
f	function †
d_5-d_0	six digit data value
E-ss	range expressed as an exponent of two digits
^ ^	two blanks
gg	function expressed as a two-digit number

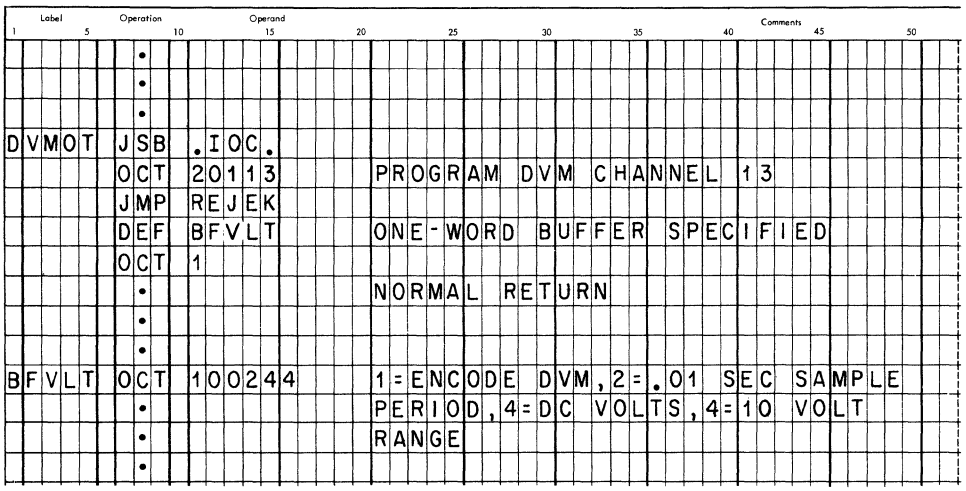
Example:

```
Label     Operation  Operand     Comments
1    5    10         15     20        25        30        35        40        45        50
          .          .
          .          .
          .          .
DSIOT     JSB        .IOC.
          OCT        20115       REMOVE DVM "HOLD-OFF", DVM ON
          JMP        REJEK       CHANNEL 15
          OCT        0           DUMMY
          OCT        0           BUFFER
          .                      NORMAL RETURN
          .          .
          .          .
DSIIN     JSB        .IOC.
          OCT        10015       READ AND CONVERT TO ASCII
          JMP        *-2         LOOP UNTIL DATA READY
          DEF        BFDSI       SIXTEEN CHARACTER
          DEC        -16         BUFFER
          JMP        ...         NORMAL RETURN
BFDSI     BSS        8
          .          .
          .          .
          .          .
```

† See Appendix D

When a Write request is made for Digital Voltmeter Programmer, a one-word buffer must be specified. This word contains the voltmeter program: sample period (bits 7-6), function (bits 5-3), and range (bits 2-0).† If bit 15 contains a 1, an encode command is sent to the Voltmeter (always 0 if the configuration includes a Scanner).

Example:



† See Appendix D

When a Scanner Programmer output operation is performed, the system requires a two-word buffer. The first word contains the scanner program: the function (bits 4-3) and the delay (bits 2-0).† The second word contains the channel number for the start of the scan. The driver subroutine converts the binary channel number value produced by the Assembler to the BCD format required by the device.

Example:

```
Label   Operation  Operand              Comments
            .
            .
            .
SCNOT   JSB   .IOC.
        OCT   20114           SEND PROGRAM AND CHANNEL TO
        JMP   REJEK           SCANNER ON CHANNEL 14
        DEF   BFSCN
        DEC   2
        JMP   ...
BFSCN   OCT   03              0=VOLTS,3=27MSEC DELAY
        DEC   100             START CHANNEL 100
            .
            .
            .
```

† See Appendix D

CONSOLIDATED CODING SHEET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	AND	001		0	Z/C	← Memory Address →									
D/I	XOR	010		0	Z/C										
D/I	IOR	011		0	Z/C										
D/I	JSB	001		1	Z/C										
D/I	JMP	010		1	Z/C										
D/I	ISZ	011		1	Z/C										
D/I	AD*	100		A/B	Z/C										
D/I	CP*	101		A/B	Z/C										
D/I	LD*	110		A/B	Z/C										
D/I	ST*	111		A/B	Z/C										

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	SRG	000		A/B	0	D/E	*LS		000	CLE	D/E	SL*	*LS		000
							*RS		001				*RS		001
							R*L		010				R*L		010
							R*R		011				R*R		011
							*LR		100				*LR		100
							ER*		101				ER*		101
							EL*		110				EL*		110
							*LF		111				*LF		111
				NOP	000				000		000				000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	ASG	000		A/B	1	CL*	01	CLE	01	SEZ	SS*	SL*	IN*	SZ*	RSS
						CM*	10	CME	10						
						CC*	11	CCE	11						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	IOG	000		A/B	1	H/C	HLT		000	← Select Code →					
					1	0	STF		001						
					1	1	CLF		001						
					1	0	SFC		010						
					1	0	SFS		011						
					1	H/C	MI*		100						
					1	H/C	LI*		101						
					1	H/C	OT*		110						
				0	1	H/C	STC		111						
				1	1	H/C	CLC		111						
					1	0	STO		001		000			001	
					1	1	CLO		001		000			001	
					1	H/C	SOC		010		000			001	
					1	H/C	SOS		011		000			001	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	EAU	000		MPY**		000		010			000			000	
				DIV**		000		100			000			000	
				DLD**		100		010			000			000	
				DST**		100		100			000			000	
				ASR		001		000		0	1	← number of bits →			
				ASL		000		000		0	1				
				LSR		001		000		1	0				
				LSL		000		000		1	0				
				RRR		001		001		0	0				
				RRL		000		001		0	0				

Notes: * = A or B.
D/I, A/B, Z/C, D/E, H/C coded: 0/1.
**Second word is Memory Address.

INDEX

CUT ALONG LINE

HEWLETT hp PACKARD

READER COMMENT SHEET

ASSEMBLER/BCS TRAINING MANUAL

HP 02116-9073 April, 1970

Hewlett-Packard welcomes your evaluation of this text. Any errors, suggested additions, deletions, or general comments may be made below. Use extra pages if you like.

FROM

NAME: _____

ADDRESS: _____

_____

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AS SHOWN ON OTHER SIDE AND TAPE

FOLD FOLD

FIRST CLASS
PERMIT NO. 141
CUPERTINO
CALIFORNIA

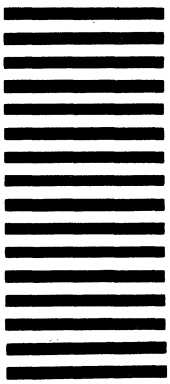
BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States Postage will be paid by

SUPERVISOR, SOFTWARE PUBLICATIONS

HEWLETT - PACKARD

CUPERTINO DIVISION
11000 Wolfe Road
Cupertino, California
95014



FOLD FOLD