Programming the IBM 7090:

A Self-Instructional Programmed Manual

JAMES A. SAXON

Saxon Research Corporation

PRENTICE-HALL, INC., ENGLEWOOD CLIFFS, N. J. 1963

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA, PTY., LTD., *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL FRANCE, S.A.R.L., *Paris*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL DE MEXICO, S.A., *Mexico City*

Library of Congress Catalog Card Number 63-10543

Printed in the United States of America

73033–C

ACKNOWLEDGEMENT

The technical assistance and constructive criticism given by Dr. George Forsythe and Mr. James Watt, both of Stanford University, and Mr. Ted Medin of General Dynamics, Astronautics, is greatly appreciated by the author.

INTRODUCTORY NOTE

This Self-Instructional Text Book is designed to perform the function of teaching you to program for the IBM 7090 computer.

There will be no formal test at any time throughout the course. You will go through it as fast or as slowly as you desire. It is recommended that study periods should not extend beyond two hours and that no more than two such (two hour) periods be utilized during any one day.

There are large numbers of problems and exercises scattered throughout the book. In every case, the correct answer is given on the back of the page. You are to work each problem in the space allotted to it in the book and then check your answer with the correct answer given. If your answer was incorrect, go back to the previous page for an additional review.

There is nothing to keep you from cheating by looking at the correct answer before you have attempted to work the problem except the realization that you will not learn to program if you do so. The fact that you have this book in front of you indicates that you want to learn to program. If this is true, then please follow all instructions to the letter. Thank you for your cooperation.

Computer manufacturers are constantly making advances and some of the limitations listed in this text will be exceeded, but as long as the 7090 or similar computers are used, the general information and programming methodology will be applicable.

TABLE OF CONTENTS

CONTENTS continued

GENERAL INFORMATION

Before getting into the mechanics of programming for the 7090, a certain amount of general information relating to the characteristics and operation of the machine, should be discussed.

The 7090 is a scientific computer. Although it can, and does, do other work, its major function is that of solving complex mathematical problems. Despite complex formulas, every problem can be broken down to the four basic arithmetic operations of addition, subtraction, multiplication and division. This is the method the computer uses in solving its problems. It may have to multiply a set of numbers a thousand times (or a million times), but this poses no problem as each operation is executed in a tiny fraction of a second. The computer is controlled and told what to do by human beings through the use of <u>programs</u>, which are interpreted and executed by the machine.

A <u>program</u>, is a sequence of instructions, stored internally in the machine, which tell the computer exactly what to do with the data to be processed. It must take into account every eventuality and all possibilities. Nothing must be left to chance because the machine has no capacity for thinking. It can only do what it has been told to do by the <u>program</u>. For example, if an overflow occurs during an arithmetic operation and the programmer has not provided for this possibility in his program, the machine will not be able to handle it.

There are three phases in computer processing: INPUT, COMPUTATION and OUTPUT. The <u>input</u> phase consists of placing the instructions and data to be processed into the computer. Input may be punched cards or magnetic tape although magnetic tape is more commonly used as it is a much faster method.

The <u>computation</u> phase carries out the instructions. It has two functions, that of <u>arithmetic</u> and <u>control</u>. <u>Arithmetic</u> simply carries out those instructions that are concerned with arithmetic operations and <u>control</u> carries out the instructions in a specified order. Normally, the computer carries out instructions sequentially (one after the other), but the programmer may use certain <u>control</u> instructions which may instruct the computer to proceed to any instruction in the program.

The output phase consists of reporting the results of the computer action. This may be in printed form, on punched cards or on tape. It is most economical to produce the output on tape, then if one of the other products is desired, it may be accomplished off-line (detached from the computer), saving considerable machine operating time.

Tape, card and printer units are connected to the DATA CHANNEL (DC), which is connected to the Central Processing Unit of the computer. The DC allows input and output of information at the same time that computation is taking place. Channels A through H are available.

Each Channel may have up to ten tape units. A printer, card reader and punch may be attached to each Channel. All Channels may operate at the same time, but only one input/output unit per Channel may be in operation at any one time.

The printer writes at the rate of 150 lines per minute. The card reader reads cards at the rate of 250 cards per minute. The Punch can punch cards at the rate of 100 cards per minute. These are all extremely high speeds, but they can not be compared to the speed attained by magnetic tape. For this reason, tape is the most commonly used input/output device on the 7090.

Tape may be operated on either high or low density mode. In low density, 200 characters are packed to each inch of tape. In high density, 556 characters to an inch. Tape may be run at high or low speed. Using tape drive, model 729-II, tape passes at the rate of 75 inches per second and using tape drive, model 729-IV, it passes at the rate of 112½ inches per second. A normal tape is about 2400 feet long. In low density mode, about 900,000 machine words may be put on a reel of tape. In high density mode, about 2½ million words will fit on a single reel. This should effectively demonstrate the fantastic speeds attained in the input or output of information utilizing tapes.

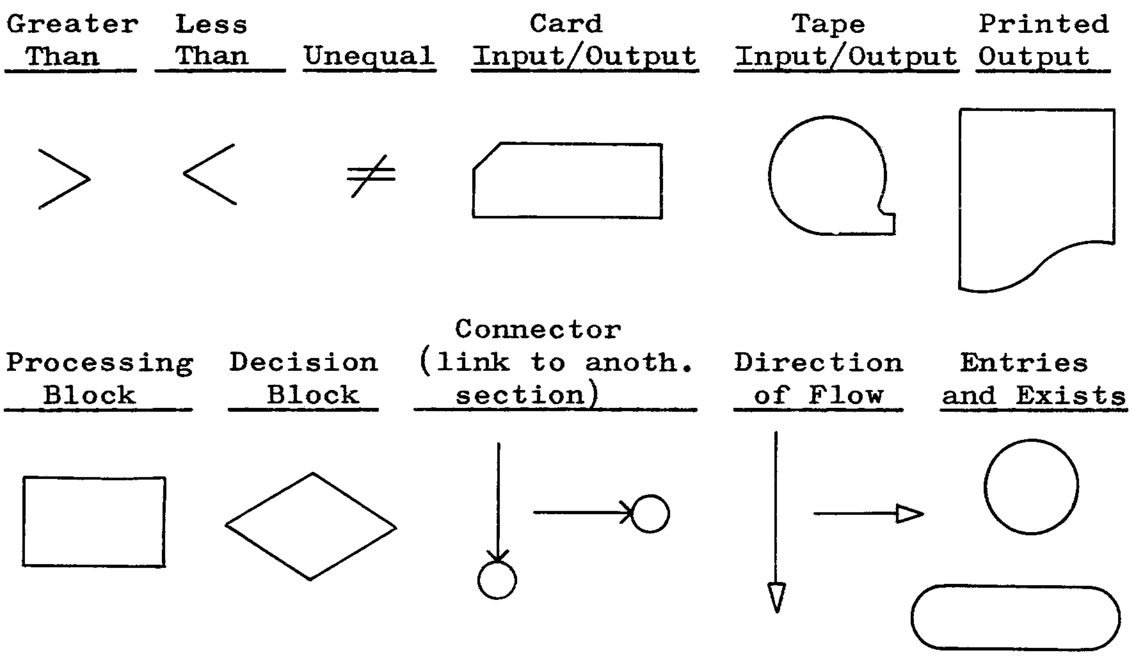
The following paragraphs are presented for the benefit of those students who have little, or no, computer background:

PLANNING: After an application to be processed is selected, it must be thoroughly planned. Planning consists of the following steps:

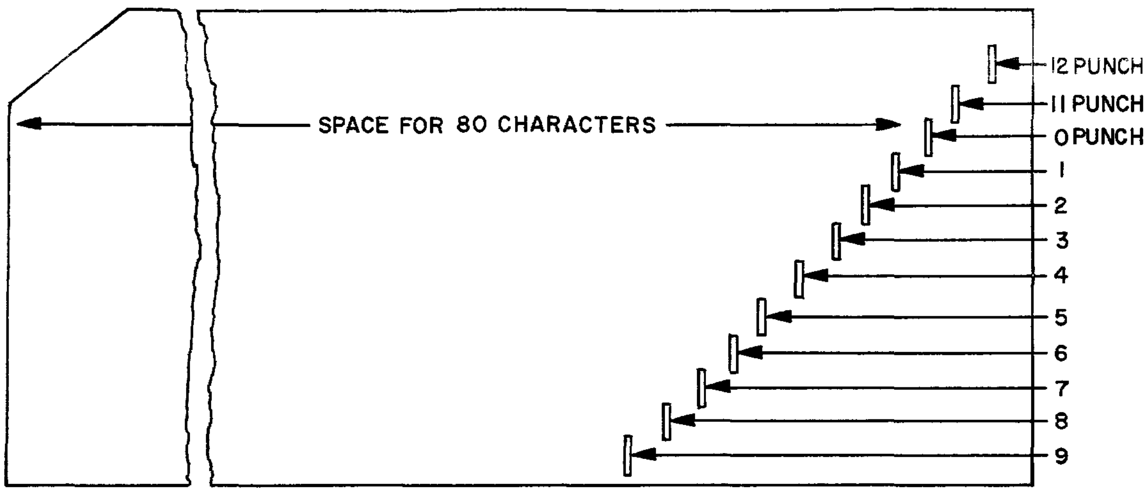
1. Analysis of the application
2. Planning and sequencing steps to be used
3. Writing the instructions
4. Determining which areas of storage will be used for various purposes

<u>FLOW CHARTING</u>: Before writing machine instructions, it is usually advisable to express the necessary steps to be taken in block diagram form. This is called flow charting. A flow chart may be quite general or very detailed, depending on the needs of the programmer. Generally speaking, the larger and more complex the problem, the more detailed the flow chart should become.

A flow chart attempts to cover all aspects of a problem. Every problem contains a multitude of detail which must be analyzed, organized and dealt with each in its own turn, with nothing left out and nothing forgotten. The flow chart is a way of accomplishing this purpose. It is also useful in making modifications and corrections to programs already written. It is advantageous to use a standardized set of symbols so that others may more easily interpret a programmers' flow chart. A few of the more commonly used signs and forms are shown below.



READING A PUNCHED CARD: It is not necessary for a fledgeling programmer to be able to read punches on a card as fluently as he reads English, but it is necessary for him to understand the code used and to be able to decypher the punches if it becomes necessary to do so. A punched card may contain up to 80 characters of information in a horizontal line and it has 12 vertical positions.



The code is as follows:

12 PUNCH - 1 PUNCH together in a column = A, 12-2=B, 12-3=C, 12-4=D, 12-5=E, 12-6=F, 12-7=G, 12-8=H, 12-9=I.

11 PUNCH - 1 PUNCH together in a column = J, 11-2=K, 11-3=L, 11-4=M, 11-5=N, 11-6=Ø, (Slash through 0 indicates it to be alphabetic), 11-7=P, 11-8=Q, 11-9=R.

0 PUNCH - 2 PUNCH together in a column = S, 0-3=T, 0-4=U, 0-5=V, 0-6=W, 0-7=X, 0-8=Y, 0-9=Z.

For numeric 1 through 9, punch only the number, omitting all three of the top columns. Special characters (i.e. comma, period) require special groupings of punches.

COMPUTER-PROGRAMMER INTERACTION: Very briefly, this is how the system works: The programmer is assigned to do a job. He analyzes, flow charts, then programs it on special programming work sheets. These work sheets go to keypunch, where cards are punched from them. This is called the source program. A special program called FAP (Fortran Assembly Program) is loaded into the computer and the source program cards are then fed into the computer. Translation of the cards into language the machine will understand is accomplished automatically by the FAP program.

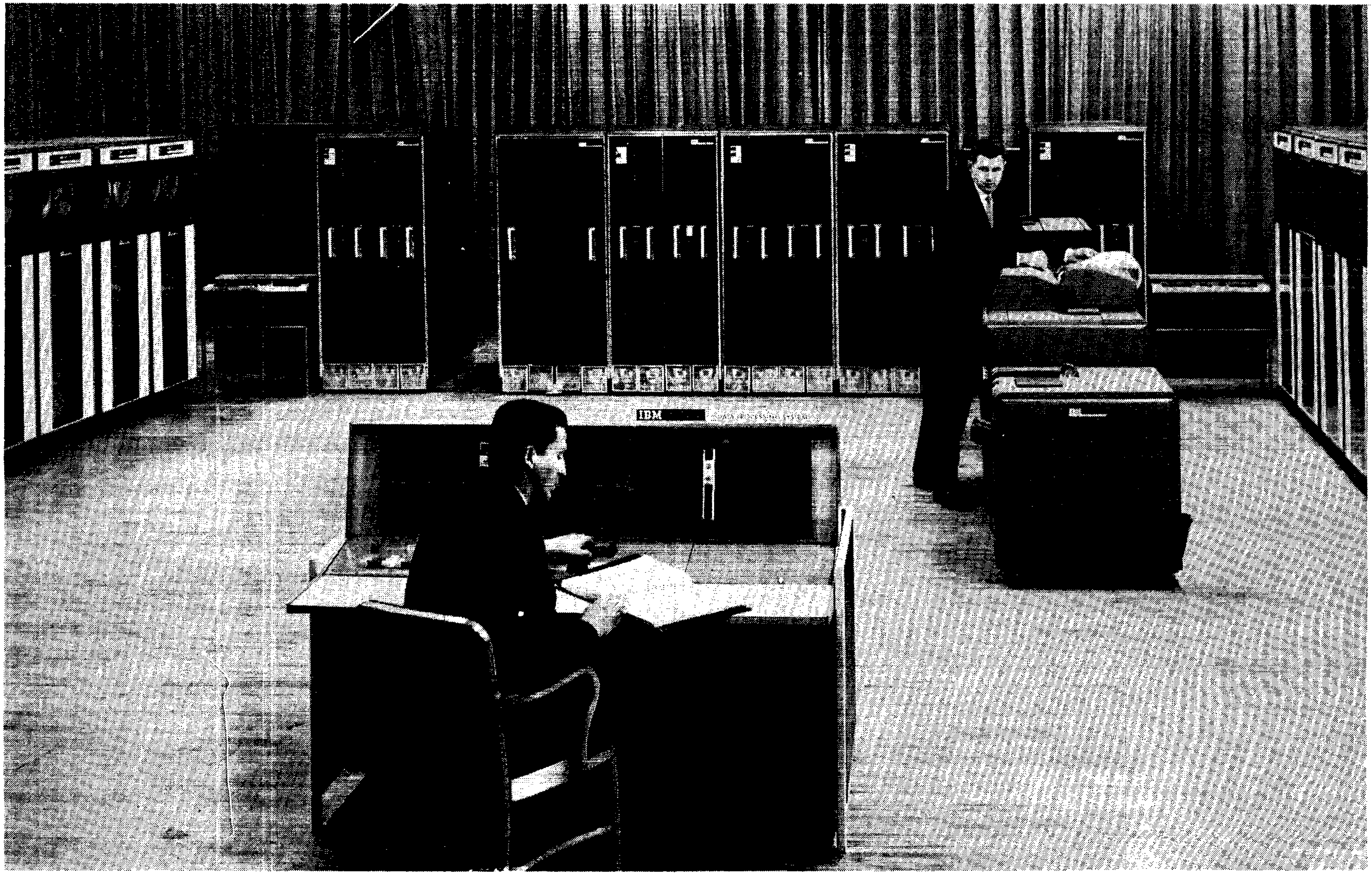
The new program is then ready for operational use and may be left on cards or put on magnetic tape. When operational data is ready for processing, the program is loaded into the computer before the data is allowed to enter. When data does enter, the program takes over and processes according to the specifications of the job.

INSTRUCTIONS: Approximately one hundred instructions will be covered in detail in this course. Many instructions will not be covered since there is a limit to the size of such a course, but the most important, or useful, ones are covered and the others may be picked up from the reference manual prepared by IBM, entitled, "Reference Manual - 7090 Data Processing System."

COURSE FORMAT: Throughout the course, a small amount of information will be imparted, followed by detailed examples and problems covering the area of information just covered. You are to work the problems in the space provided on the problem page and then check your answers with the correct answers given on the following page.

Pages xiii and xiv will give you an example of how this is done. Work the problems on page xiii to see how much you have retained from your reading of pages vii through x. When you have finished, check your answers with the correct answers given on page xiv.

Each time you pick up the book, it is a good policy to review the portion already covered before starting on the new section. It is difficult to retain everything you read from one learning session to the next and this review will help you keep the knowledge already gained.


IBM

WORK AREA

Work the problems in this space, then check your answers with the correct answers given on the next page.

PROBLEMS

A. A sequence of instructions, stored internally by the computer is called a ________________.

B. The three phases of computer processing are__________, ________________ and ____________________.

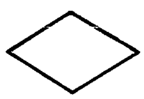
C. How many Channels are available to the 7090?________.

D. In low density, _______ characters are packed to each inch of tape. In high density, __________ characters are packed to an inch.

E. What is the length of a normal tape? ______________.

F. Define the following flow-charting symbols:

1.  ____________________________.

2.  ____________________________.

G. Give the alphabetic representation of the following punches in a card:

1. 12 PUNCH 4	_____	8. 0 PUNCH 8	Y
2. 0 PUNCH 4	_____	9. 11 PUNCH 9	R
3. 11 PUNCH 4	M	10. 12 PUNCH 1	A
4. 0 PUNCH 2	S	11. 11 PUNCH 2	K
5. 12 PUNCH 6	F	12. 0 PUNCH 9	Z
6. 12 PUNCH 9	I	13. 12 PUNCH 2	B
7. 11 PUNCH 1	J	14. 11 PUNCH 8	Q

CORRECT ANSWERS

A. Program (see page vii)

B. Input, Computation and Output (see page vii)

C. 8 (see page viii)

D. 200 556 (see page viii)

E. 2400 feet (see page viii)

F. 1. Processing Block (see page ix)

2. Decision Block (see page ix)

G. (see page ix)

1. D	8. Y
2. U	9. R
3. M	10. A
4. S	11. K
5. F	12. Z
6. I	13. B
7. J	14. Q

If you have answered all of these questions correctly, turn the page and start studying Lesson 1.

LESSON 1

DECIMAL, OCTAL AND BINARY NUMBERING SYSTEMS: The IBM 7090, and nearly all other large scale computers, operate on the BINARY numbering system. We are all familiar with the DECIMAL system, which utilizes 10 digits as its base, but many people are completely unfamiliar with the other two systems mentioned below. To program for the 7090, it is absolutely essential to become familiar with both BINARY and OCTAL systems.

The BINARY system is a base two system, utilizing only two digits, zero and one. This is most convenient for computers because an electrical current may be "on" or "off" and a magnetic field may be "magnetized" or "not magnetized". These are also base two types of actions. Since computers use BINARY circuits, the internal arithmetic of computers is BINARY in nature.

BINARY numbers tend to be extremely long (roughly 3.3 times longer than a DECIMAL number). For this reason, a shorthand method is used, called the OCTAL system. OCTAL, is a base eight numbering system, from zero through seven (0-7). OCTAL numbers are used when working with the 7090, but it must be remembered that the machine itself works in the BINARY system.

The relationship between OCTAL and BINARY is so great that conversion of numbers from one system to the other may be accomplished quite easily. A very complete set of tables has been designed to convert DECIMAL to OCTAL and OCTAL to DECIMAL numbers, but it is not necessary to depend on these tables as it is fairly simple to make the necessary conversion with pencil and paper. When working with the computer and large volumes of numbers, the conversion tables become very useful.

On the following pages, each of these two new numbering systems will be examined in detail including some simple arithmetic problems. For the time being, we will deal with whole numbers (integers) exclusively. Fractions and decimal fractions will not be discussed at this time. Fraction conversion tables are available in the event that need for them should arise.

BINARY NUMBERING SYSTEM: Counting in the BINARY system is as follows:

DECIMAL	BINARY	DECIMAL	BINARY
0	0	5	101
1	1	6	110
2	10	7	111
3	11	8	1000
4	100	9	1001

Since the BINARY system only contains 0 and 1, it is necessary to take the same "move" at 2, that is taken at 10 in the DECIMAL system. This is to place a "1" to the left and start again with "0". Therefore, a DECIMAL 2 is a BINARY 10, 3=11 and then another shift must be made, adding "1" to the left and starting again with "0".

For convenience, BINARY numbers are usually grouped in threes (001 010 100). Consider the BINARY position to the right as the "ones" position, then double the number for each position to the left (twos, fours, eights, etc.). By using this approach, we can determine the DECIMAL equivalent of any BINARY number.

EXAMPLE:

0	0	1	0	1	0	1	0	1
256	128	64	32	16	8	4	2	1

64 + 16 + 4 + 1 = 85

Add together all numbers that have BINARY "ones". Disregard "0".

A DECIMAL "7" is written as BINARY 111 (4 + 2 + 1 = 7)
A DECIMAL "15" is written as BINARY 001 111 (8 + 4 + 2 + 1 = 15)

Rather than referring to the three systems by name, it is more convenient to designate any number with the system being used, as follows:

DECIMAL 11 will be written 11_{10}

OCTAL 11 will be written 11_8

BINARY 11 will be written 011_2, but it is obvious by inspection if a number is written in BINARY, as it usually consists of a long series of zeros and ones.

WORK AREA

Work the problems in this space, then check your answers with the correct answers given on the next page.

PROBLEMS:

1. Convert 17_{10} to BINARY notation. 10 001

2. Convert 18_{10} to BINARY notation. 10 010

3. Convert 26_{10} to BINARY notation. 11 010

4. The following BINARY figures convert to what DECIMAL figures?

   a. 000 001 — 1

   b. 010 101 — 21

   c. 001 011 — 11

   d. 001 010 — 16

   e. 010 100 — 20

   f. 001 001 001 — 73

   g. 001 010 100 — 84

5. Convert 233_{10} to BINARY notation.

   011 101 001

CORRECT ANSWERS

1. 010 001 $(16 + 1 = 17_{10})$
2. 010 010 $(16 + 2 = 18_{10})$
3. 011 010 $(16 + 8 + 2 = 26_{10})$
4. a. 1

   b. $16 + 4 + 1 = 21_{10}$

   c. $8 + 2 + 1 = 11_{10}$

   d. $8 + 2 = 10_{10}$

   e. $16 + 4 = 20_{10}$

   f. $64 + 8 + 1 = 73_{10}$

   g. $64 + 16 + 4 = 84_{10}$
5. 011 101 001 $(128 + 64 + 32 + 8 + 1 = 233_{10})$

As you can see from problem 5, when the number gets fairly large, it becomes quite difficult to convert in this manner. This is one of the reasons why OCTAL is used as an intermediate step between DECIMAL and BINARY.

Lesson 1, (cont'd)

BINARY ARITHMETIC: Only a few rules need to be observed to accomplish simple arithmetic in BINARY form.

ADDITION:

Rule 1:	Zero plus zero equals zero.
Rule 2:	Zero plus one equals one.
Rule 3:	One plus one equals zero with a carry of one to the left.

EXAMPLE: Add $15_{10} + 7_{10}$

(column)	sixteens	eights	fours	twos	ones	
	1	1	1	1		(carry)
	0= 0	1= 1	0= 1	0= 1	1	$(= 15_{10})$
+	0	0	1	1	1	$(= 7_{10})$
	1	0	1	1	0	$(16 + 4 + 2 = 22_{10})$

In the "ones" column, Rule 3 applies. In the "twos" column, Rule 3 applies again, but we must further add the "carry", so the result is 1 with a "carry". The same thing happens in the "fours" column. In the "eights" column, Rule 2 applies, but again we must add the "carry", so now Rule 3 takes over and we end up with zero and a "carry". In the "sixteens" column, Rule 1 applies, then add the "carry", which winds it up with a 1.

SUBTRACTION:

Rule 1:	Zero minus zero equals zero.
Rule 2:	One minus one equals zero.
Rule 3:	One minus zero equals one.
Rule 4:	Zero minus one equals one, with one borrowed from the left.

EXAMPLE: Subtract $15_{10} - 7_{10}$

(column)	sixteens	eights	fours	twos	ones	
						(borrows)
	0	1	1	1	1	$(= 15_{10})$
−	0	0	1	1	1	$(= 7_{10})$
	0	1	0	0	0	$(= 8_{10})$

Applying the rules above, in the "ones" column, Rule 2 applies. Also in the "twos" and "fours" columns. In the "eights" column, Rule 3 applies. In the "sixteens" column, Rule 1 applies.

Similar, but somewhat different rules are used for multiplication and division. They are nothing more than sequences of addition and subtraction, extremely cumbersome with paper and pencil, but very rapidly accomplished with the high speeds attained by modern computers. This page demonstrates the way arithmetic is actually accomplished within the computer.

Lesson 1, (cont'd)

EXAMPLES:

1. Add: $3_{10} + 3_{10}$

fours	twos	ones	
1	1		(carries)
0	1	1	(3)
0	1	1	(3)
1	1	0	(4 + 2 = 6)

In the "ones" column, Rule 3 applies. In the "twos" column, two steps must be taken; first, 1 + 1 = 0 with a carry; second, the 0 (resulting from the first step) + 1 (from the previous carry) = 1. In the "fours" column, two steps must be taken; first, 0 + 0 = 0, second, this 0 + 1 (from the previous carry) = 1. Each time there is a "carry", the second step must be taken.

2. Add: $4_{10} + 3_{10}$

fours	twos	ones	
			(carries)
1	0	0	(4)
0	1	1	(3)
1	1	1	

(4 + 2 + 1 = 7)

3. Add: $7_{10} + 6_{10}$

eights	fours	twos	ones	
1	1			(carries)
	1	1	1	(7)
	1	1	0	(6)
1	1	0	1	

(8 + 4 + 1 = 13)

---

4. Subtract: $12_{10} - 4_{10}$

eights	fours	twos	ones	
				(borrows)
1	1	0	0	(12)
0	1	0	0	(4)
1	0	0	0	(= 8)

5. Subtract: $12_{10} - 7_{10}$

eights	fours	twos	ones	
	1		1	(borrows)
0 1	0 1	1 0	0	(12)
0	1	1	1	(7)
0	1	0	1	(4 + 1 = 5)

In the "ones" column, Rule 4 applies, but since there is no "1" to borrow in the "twos" column, we must get it from the "fours" column, changing the 1 to a 0 in the "fours" and the 0 to a 1 in the "twos". In the "twos" column, Rule 2 applies. In the "fours" column, 0-1 causes a "borrow" from the "eights" column, leaving it a 0, which results in 0 for the final subtraction.

WORK AREA

Work the problems in this space, then check your answers with the correct answers given on the next page.

PROBLEMS

6. Add:

	sixteens	eights	fours	twos	ones	(carries)
	0	1	0	0	1	(9)
+	0	0	1	1	0	(6)
Result:	0	1	1	1	1	

7. Add:

	sixteens	eights	fours	twos	ones	(carries)
	1	1	0	0	1	(25)
+	0	0	1	1	0	(6)
Result:	1					

8. Add:

	sixteens	eights	fours	twos	ones	(carries)
		1	0	1	0	(10)
+		0	1	1	1	(7)
Result:	1	0	0	0	1	

---

9. Subtract:

	sixteens	eights	fours	twos	ones	(borrows)
	0	1	0	0	1	(9)
-	0	0	1	1	0	(6)
Result:				1		

10. Subtract:

	sixteens	eights	fours	twos	ones	(borrows)
	1	1	0	0	1	(25)
-	0	0	1	1	0	(6)
Result:	1	0	0	1	1	

11. Subtract:

	sixteens	eights	fours	twos	ones	(borrows)
		1	0	1	0	(10)
-		0	1	1	1	(7)
Result:		0	0	1	1	

CORRECT ANSWERS

6.

	sixteens	eights	fours	twos	ones	
						(carries)
	0	1	0	0	1	(9)
+	0	0	1	1	0	(6)
Result:	0	1	1	1	1	(8 + 4 + 2 + 1 = 15)

7.

	sixteens	eights	fours	twos	ones	
						(carries)
	1	1	0	0	1	(25)
+	0	0	1	1	0	(6)
Result:	1	1	1	1	1	

(16 + 8 + 4 + 2 + 1 = 31)

8.

	sixteens	eights	fours	twos	ones	
	1	1	1			(carries)
		1= 1	1= 0	1	0	(10)
+		0	1	1	1	(7)
Result:	1	0	0	0	1	(16 + 1 = 17)

9.

	sixteens	eights	fours	twos	ones	
				1		(borrows)
	0	0 1	1 0	0	1	(9)
-	0	0	1	1	0	(6)
Result:	0	0	0	1	1	(2 + 1 = 3)

10.

	sixteens	eights	fours	twos	ones	
				1		(borrows)
	1	0 1	1 0	0	1	(25)
-	0	0	1	1	0	(6)
Result:	1	0	0	1	1	(16 + 2 + 1 = 19)

11.

	sixteens	eights	fours	twos	ones	
				1	1	(borrows)
		0 1	1 0	0 1	0	(10)
-		0	1	1	1	(7)
Result:		0	0	1	1	(2 + 1 = 3)

<u>OCTAL NUMBERING SYSTEM</u>: This is a <u>base</u> <u>8</u> system, using the digits from 0 through 7. Counting in the <u>OCTAL</u> system is as follows (notice that "8" and "9" are never used):

DECIMAL	OCTAL	DECIMAL	OCTAL
0	0	8	10
1	1	9	11
2	2	10	12
3	3	11	13
4	4	12	14
5	5	13	15
6	6	14	16
7	7	15	17

The relationship between <u>OCTAL</u> and <u>BINARY</u> is so simple that conversion may be made instantaneously. Consider every <u>BINARY</u> number in groups of threes (001010101 = 001 010 101). Now, each grouping of three <u>BINARY</u> digits is identified by "ones," "twos," and "fours" positions and these are used to convert to <u>OCTAL</u>, as follows:

$$\underbrace{\overset{\text{fours}}{0}\quad\overset{\text{twos}}{0}\quad\overset{\text{ones}}{1}}_{1}\qquad\underbrace{\overset{\text{fours}}{0}\quad\overset{\text{twos}}{1}\quad\overset{\text{ones}}{0}}_{2}\qquad\underbrace{\overset{\text{fours}}{1}\quad\overset{\text{twos}}{0}\quad\overset{\text{ones}}{1}}_{5}\quad\underline{\text{OCTAL}}$$

<u>EXAMPLES</u>:

1. Binary: 011 011 010 111

   Octal: 3 3 2 7

2. Binary: 10 010

   Octal: 2 2

   If the <u>Binary</u> digits do not come out in groups of "three", add zeros to the left until the final group also contains three digits.

3. Binary: 0 100 010 110

   Octal: 0 4 2 6

CONVERTING FROM OCTAL TO DECIMAL: This is usually accomplished by looking up the number in a conversion table (see 7090 Reference Manual, Appendix B and C). It may be accomplished manually in the following manner:

Multiply each Octal position in turn by 8, starting with the high-order (left-most) position. Then, add the next number to the result, until the last digit is reached (this one is not to be multiplied).

EXAMPLE 1: $3327_8 = ?_{10}$

$$\begin{array}{r}
3327_8 \\
\times\ 8 \\ \hline
24 \\
+\ 3 \\ \hline
27 \\
\times\ 8 \\ \hline
216 \\
+\ 2 \\ \hline
218 \\
\times\ 8 \\ \hline
1744 \\
+\ 7 \\ \hline
1751_{10}
\end{array}$$

Result ($3327_8 = 1751_{10}$)

EXAMPLE 2: $426_8 = ?_{10}$

$$\begin{array}{r}
426_8 \\
\times\ 8 \\ \hline
32 \\
+\ 2 \\ \hline
34 \\
\times\ 8 \\ \hline
272 \\
+\ 6 \\ \hline
278_{10}
\end{array}$$

Result ($426_8 = 278_{10}$)

<u>CONVERTING FROM DECIMAL TO OCTAL</u>: This procedure is also generally accomplished by checking the conversion table, but it may be done manually in the following manner:

Successively divide the decimal figure by 8, until no further division is possible. The <u>Octal</u> result will be the last quotient figure, followed by each of the remainders, starting from the last and finishing with the first.

<u>EXAMPLE 1</u>: $1751_{10} = ?_8$

```
     218             27            3
 8 ) 1751       8 ) 218       8 ) 27
     16             16            24
      15             58            3
       8             56
      71              2
      64
       7
```

Result: $1751_{10} = 3327_8$

<u>EXAMPLE 2</u>: $278_{10} = ?_8$

```
     34              4
 8 ) 278        8 ) 34
     24             32
      38             2
      32
       6
```

Result: $278_{10} = 426_8$

<u>EXAMPLE 3</u>: $15273_{10} = ?_8$

```
     1909             238           29           3
 8 ) 15273       8 ) 1909     8 ) 238      8 ) 29
     8               16           16           24
      72              30           78           5
      72              24           72
       73              69           6
       72              64
        1               5
```

Result: $15273_{10} = 35651_8$

With what we have learned to this point, it becomes obvious that it is not necessary to add or subtract in BINARY form. Simply convert to OCTAL and from OCTAL to DECIMAL before doing the arithmetic operation.

<u>EXAMPLES</u>:

1. <u>BINARY</u> — <u>CHANGE TO OCTAL</u> — <u>CHANGE TO DECIMAL</u>

$$\begin{array}{rr} & 010\ 111 \\ + & \underline{110\ 010} \\ & ? \end{array} \qquad \begin{array}{rr} & 2\ 7_8 \\ + & \underline{6\ 2_8} \\ & ? \end{array} \qquad \begin{array}{rr} & 27 \\ \times & \underline{8} \\ & 16 \\ + & \underline{7} \\ & 23 \end{array} \qquad \begin{array}{rr} & 62 \\ \times & \underline{8} \\ & 48 \\ + & \underline{2} \\ & 50 \end{array} \qquad \begin{array}{r} 2\ 3_{10} \\ \underline{5\ 0_{10}} \\ 7\ 3_{10} \end{array}$$ Result

2. <u>BINARY</u> — <u>CHANGE TO OCTAL</u> — <u>CHANGE TO DECIMAL</u>

$$\begin{array}{rr} & 110\ 010 \\ - & \underline{010\ 111} \\ & ? \end{array} \qquad \begin{array}{rr} & 6\ 2_8 \\ - & \underline{2\ 7_8} \\ & ? \end{array} \qquad \begin{array}{rr} & 5\ 0_{10} \\ - & \underline{2\ 3_{10}} \\ & 2\ 7_{10} \end{array}$$ Result

3. <u>BINARY</u> — <u>CHANGE TO OCTAL</u> — <u>CHANGE TO DECIMAL</u>

$$\begin{array}{rr} & 001\ 010\ 100 \\ + & \underline{001\ 001\ 101} \\ & ? \end{array} \qquad \begin{array}{rr} & 1\ 2\ 4_8 \\ + & \underline{1\ 1\ 5_8} \\ & ? \end{array} \qquad \begin{array}{rr} & 124 \\ \times & \underline{8} \\ & 8 \\ + & \underline{2} \\ & 10 \\ \times & \underline{8} \\ & 80 \\ + & \underline{4} \\ & 84 \end{array} \qquad \begin{array}{rr} \times & 115 \\ \times & \underline{8} \\ & 8 \\ + & \underline{1} \\ & 9 \\ \times & \underline{8} \\ & 72 \\ + & \underline{5} \\ & 77 \end{array} \qquad \begin{array}{rr} & 8\ 4_{10} \\ + & \underline{7\ 7_{10}} \\ & 1\ 6\ 1_{10} \end{array}$$ Result

To set up a <u>BINARY</u> number (starting with a <u>DECIMAL</u> number), convert in the other direction.

4. $956_{10} = ?_2$

<u>DECIMAL</u> — <u>CHANGE TO OCTAL</u> — <u>WRITE THE OCTAL OUT IN BINARY FORM</u>

$$956_{10} = 1674_8 = \underbrace{001}_{1}\ \underbrace{110}_{6}\ \underbrace{111}_{7}\ \underbrace{100}_{4}$$

$$\begin{array}{r} 119 \\ 8\overline{)956} \\ \underline{8} \\ 15 \\ \underline{8} \\ 76 \\ \underline{72} \\ 4 \end{array} \qquad \begin{array}{r} 14 \\ 8\overline{)119} \\ \underline{8} \\ 39 \\ \underline{32} \\ 7 \end{array} \qquad \begin{array}{r} 1 \\ 8\overline{)14} \\ \underline{8} \\ 6 \end{array}$$

PROBLEMS:

12. $565_{10} = ?_8$

Result: $\boxed{}$

13. $565_8 = ?_{10}$

Result: $\boxed{}$

14. $1242_{10} = ?_2$

Result: $\boxed{}$

15. $010\ 101\ 110 = ?_{10}$

Result: $\boxed{}$

16. $135_{10} = ?_8$

Result: $\boxed{}$

17. $135_8 = ?_{10}$

Result: $\boxed{}$

18. $111\ 100\ 001 = ?_8$

Result: $\boxed{}$

19. Result of problem 18 $= ?_{10}$

Result: $\boxed{}$

WORK AREA

Work the problems in this space, then check your answers with the correct answers given on the next page.

<u>CORRECT ANSWERS</u>

12. 1065_8

$$\begin{array}{r} 70 \\ 8\,\overline{)565} \\ \underline{56} \\ 5 \end{array} \qquad \begin{array}{r} 8 \\ 8\,\overline{)70} \\ \underline{64} \\ 6 \end{array} \qquad \begin{array}{r} 1 \\ 8\,\overline{)8} \\ \underline{8} \\ 0 \end{array}$$

13. 373_{10}

14. 010 011 011 010 (2332_8)

15. 174_{10}

16. 207_8

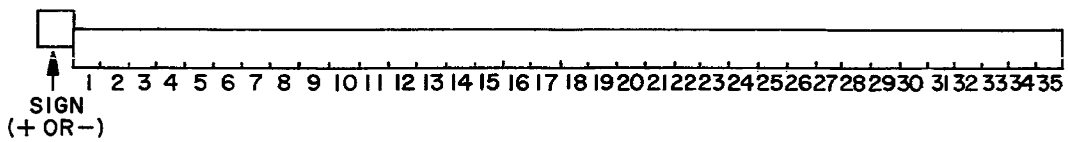
17. 93_{10}

18. 741_8

19. 481_{10}

LESSON 2

MACHINE WORDS: The Memory, or Storage Unit, of the 7090 contains space for 32,768 machine words. The term word, refers to a unit of information. It may be an instruction to the machine or a piece of data which will be processed by the machine. The 7090 is a fixed word length machine. This means that every machine word is exactly the same size as every other word. The words are numbered from 00000 through 32,767 and each word may be called upon by the programmer. This is termed addressing a word. The word itself is 36 positions (binary bits) in length and may be shown symbollically in the following manner:



A "zero" in the sign position indicates "+". A "one" indicates "-". This leaves 35 positions, or Binary bits, for the word itself.

WORK AREA

Work the problems in this space, then check your answers with the correct answers given on the next page.

PROBLEMS:

20. Convert $32,767_{10}$ to Octal. Result: [ ]

21. A machine word is always __________ positions in length.

22. Each word may be __________ by the programmer.

23. Convert the result of problem 20 to Binary.

    Result: [ ]

24. A plus sign (+) is always designated by a Binary __________.

25. A minus sign (-) is always designated by a Binary __________.

26. Machine words are numbered from ______ through __________.

27. A word may be either an __________ to the 7090, or a piece of __________.

CORRECT ANSWERS

20. 77777_8

21. 36

22. Addressed

23. 111 111 111 111 111

24. 0

25. 1

26. 00000 through 32,767

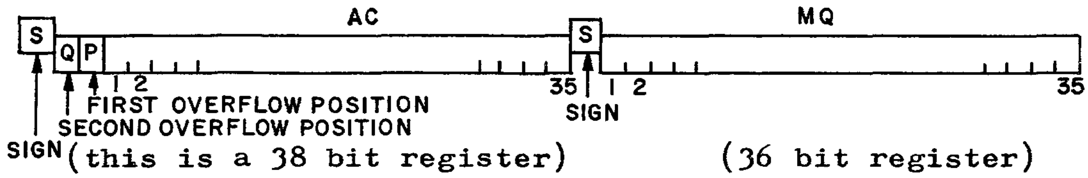
27. instruction   data

If any of your answers were incorrect, please turn back to page 15 and read it over again.

REGISTERS: There are several registers in the Central Processing Unit (CPU) of the 7090, which are used for specific processing actions. A brief description of each register will be given here.

1. AC (Accumulator) and MQ (Multiplier-Quotient) Registers:

All arithmetic operations are handled through these two registers. A great deal more will be said about them later. Symbolically represented, they look like this:

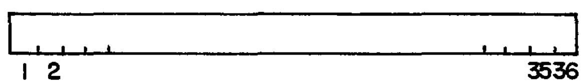


(this is a 38 bit register) (36 bit register)

These two registers may be considered to be working together, with the MQ as the right-most extension of the AC.

2. SI (Sense Indicator Register):

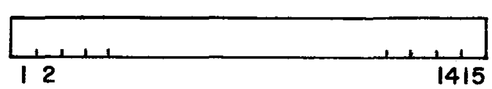
It is possible to manipulate individual bits in this register, using them as switches.



(36 bit register)

3. XR (Index Registers):

There are three Index Registers, which are referred to as XR 1, XR 2 and XR 4. Index Registers are extremely useful to count or decrement sequences of numbers and to move the program to subroutines and back to the main program from subroutines.



(15 bit registers)

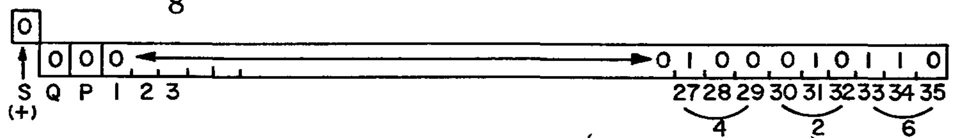
All of the registers will be discussed in detail as they become useful in programming. There are other registers which are not mentioned here because, although they are necessary for machine processing, they are not applicable to programmer manipulation. These registers are the Storage Register and Instruction Register.

AC AND MQ REGISTERS: All arithmetic operations are handled through these two registers.

Addition and Subtraction: These operations always take place in the AC and since the result may be larger than each of the figures being added or subtracted, positions "P" and "Q" are provided for any overflow that may occur.

One of the numbers (to be added or subtracted) is moved into the AC, going into the right-most portion of the register. Any unused portions would be filled with zeros at this point in time.

EXAMPLE: Move 426_8 into the AC



Then the add (or subtract) instruction is given, addressing the storage position where the other number is located. This will add (or subtract) into the number already stored in the AC. The result then may be moved from the AC to a specific location in storage, and further processing may continue.

Multiplication and Division: In these operations, the MQ is considered to be attached to the AC, to form a 72 bit register (not counting the sign positions). In multiplication, the most significant half of the product will be in the AC and the least significant half in the MQ. In division, the remainder will be in the AC, while the quotient will be in the MQ (including the sign). These operations will be discussed in much more detail later in the course.

Plus zero and Minus zero: It is quite often necessary to compare the number in the AC with a number in storage to determine whether the number in the AC is less than (<), equal to (=) or greater than (>) the number in storage. In these comparisons, it is important to understand that the computer considers +0 as greater than -0.

WORK AREA

Work the problems in this space, then check your answers with the correct answers given on the next page.

PROBLEMS:

28. Most of the registers used in the 7090, are ________ positions in length, containing one position for the ________ and ________ additional positions for the machine word.

29. Which register has two additional positions?__________

30. These two additional positions are used to take care of __________ in __________ and ___________ operations.

31. How many positions does an Index Register have? ______

32. The three Index Registers are called _____________, ______________ and _______________.

33. Identify the following signs:

    a. > ___________ ______

    b. < ___________ ______

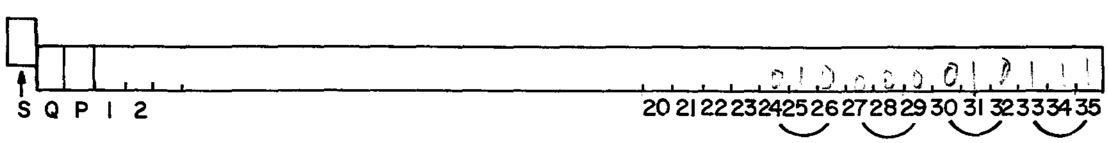
34. The ________ Register must be used for addition or subtraction.

35. In multiplication, the most significant half of the product will be in the ____________ Register.

36. In division, the quotient will be in the ____ Register.

37. Add the following figures, and show the result in the AC. Also show the sign:

$$
\begin{array}{r}
427_{10} \\
376_{10} \\
244_{10} \\
\hline
= \quad ?
\end{array}
$$



CORRECT ANSWERS

28. 36 sign 35

29. Accumulator (AC)

30. overflow addition subtraction

31. 15

32. XR1 XR2 XR4

33. a. Greater than

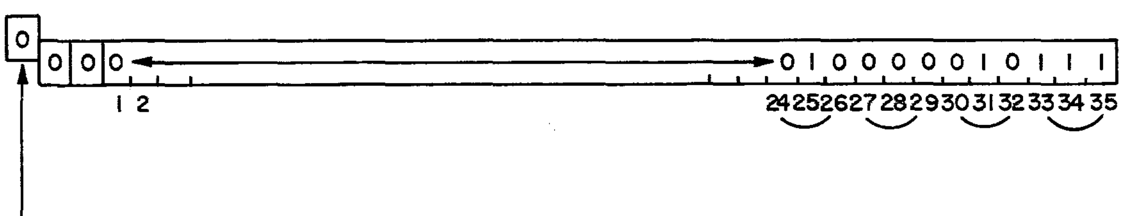
    b. Less than

34. AC

35. AC

36. MQ

37. $$\begin{array}{r} 427_{10} \\ 376_{10} \\ \underline{244_{10}} \\ 1047_{10} \end{array} = 2027_8$$



The sign is plus (+) unless indicated otherwise.

FORMAT OF INSTRUCTIONS: An instruction word consists of 35 Binary bits and a sign. It is divided into parts, each of which is named and performs a specific function. There are five major groupings of instructions which will be referred to as Type A, B, C, D and E. There are also three formats used by the DC (discussed on page viii), which will be shown at a later time.

TYPE A INSTRUCTION FORMAT

OP CODE	DECREMENT	TAG	ADDRESS (Y)
S,1,2	3 17	18 – 20	21 35

OP. CODE (Operation Code): This is always a 3 digit code found at the beginning of the word, as shown above. It tells the machine what operation is to be performed.

DECREMENT: This field is used for a group of instructions which test or change the contents of an Index Register. (To be discussed in detail later in the course.)

TAG: These 3 digits are used to identify the Index Register to be used (if any). (These will be discussed in detail later in the course.)

001 = XR1, 010 = XR2, 100 = XR4

ADDRESS: This is the location in storage of the data to be used with the instruction. This will be referred to as c(Y) (contents of Y - "Y" being the storage address where the data may be found) when discussing the various instructions.

TYPE B INSTRUCTION FORMAT

OP CODE	IA	NOT USED	TAG	ADDRESS (Y)
S,1 11	12-13		18- 20	21 35

OP. CODE: In this type instruction, Op. Code includes the sign and the first 11 positions.

IND. ADDR. (Indirect Addressing): This deals with address modification, as do the Index Registers. This will be discussed in detail later in the course. If "one" bits are in both positions 12 and 13, this is known as a flag for indirect addressing.

TYPE C INSTRUCTION FORMAT

OP CODE	COUNT	TAG	ADDRESS (Y)
S,1 9	10 17	18 -20	21 35

OP. CODE: In this type instruction, Op. Code includes the sign and the first 9 positions.

COUNT: This area contains bits which are tested during the execution of an instruction. More detail will be furnished as instructions of this type are used.

TYPE D INSTRUCTION FORMAT

OP CODE	NOT USED	MASK OR CONTROL
S,1	11	18 — 35

MASK: The Sense Indicator (SI) instructions use the address and tag fields as a mask. More detail on this later in the course.

TYPE E INSTRUCTION FORMAT

OP CODE	NOT USED	TAG	NOT USED	OP CODE
S,1	11	18-20		24 — 35

OP. CODE: In this type, Op. Code includes not only positions S and 1-11, but also positions 24-35. It is most important when using Type E instructions, not to place anything into what is normally the address portion, as this would have the effect of changing the Op. Code.

All of these instruction formats seem very confusing, but in reality a little further study will help to clarify them to a certain extent. Actual use of the various instructions will do more than anything else to straighten them out in the mind of the student. As the function of each instruction becomes clear, the various parts will also become clear as to use and function.

EXCEPTIONS: In one Type A instruction, positions 3-35 are not used. In one Type C instruction, the grouping of the bits is slightly different from that shown in the format.

The Op. Code always contains a sign (+ or -) and the binary code which tells the machine which operation it is to perform. For example: ADD, would be +00100000000 in binary form. It is more convenient to write this in octal: +0400. TRANSFER ON INDEX LOW would be: - 11000000000. In Octal: -3000.

Type A instructions (in Octal) always have a single non-zero digit, followed by three zeros. These zeros may be covered up by the decrement portion of the instruction without losing the instruction. Since the first Octal digit of the Op. Code is represented by only two Binary digits, Type A can only include 1000, 2000 and 3000 (also may be -1000, -2000, -3000). All other Op. Codes start with a zero after the Sign position and these are never Type A instructions.

Type B instructions may be distinguished by the fact that no part of the instruction is used for testing or control.

Type C has a "test" area in positions 10-17. The Octal representations of these instructions must end in 4, so that the last two digits will be zeros which may be overlapped by the Count field.

Type D has a "mask", or "control", area in the entire second half of the word, from 18 through 35.

Type E is easily distinguished from the others as the Op. Code is in two separate parts of the word (S, 1-11 and 24-35).

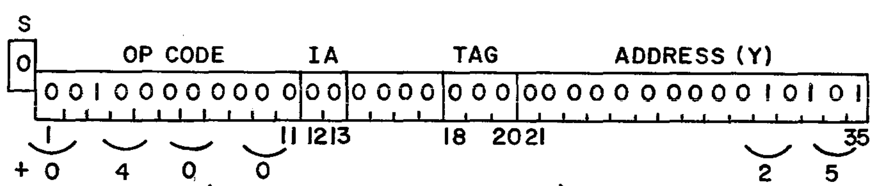
Not only does the 7090 have several different instruction formats, but it also has well over 150 different instructions. It is not necessary to memorize all of the instructions. The IBM 7090 Reference Manual lists all of them, including their Octal codes. About one-third of the instructions are basic and most commonly used. The greatest stress will be placed on these instructions throughout this course.

One final point before looking at some of the actual instructions. Although each <u>type</u> of instruction contains several <u>parts</u>, they are not all used in every instruction. For example, the Tag portion may be used if an Index Register is involved. Otherwise it is disregarded. In most of the instructions, the <u>Address</u> (contents of storage location Y) is needed so that the computer will know where to go to get the data that is to be processed and <u>all</u> instructions must have an <u>Operation Code</u>, so that the computer will know what operation to carry out.

Example of an instruction as it would look in storage:

ADD 2 1_{10} This means, "Add the contents of storage location 2 1_{10}."

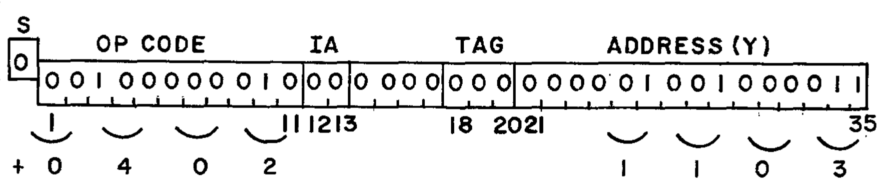
ADD = +0400
2 1_{10} = 2 5_8



(Type B Instruction)

Example 2:
SUB 579_{10} This means, "Subtract the contents of storage location 579_{10}."

SUB = +0402
579_{10} = 1103_8

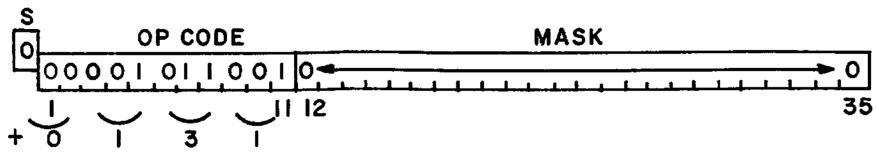


(Type B. Instruction)

Lesson 2, (cont'd)

Additional Examples: The meaning of these instructions is not important at this time.

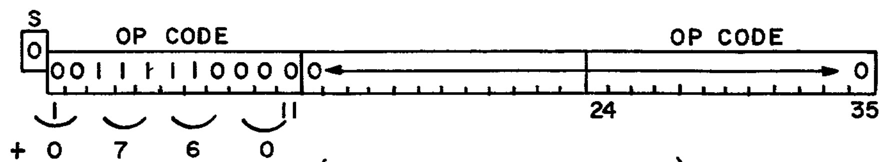
3. Instruction: XCA (+0131)



(TYPE D INSTRUCTION)

---

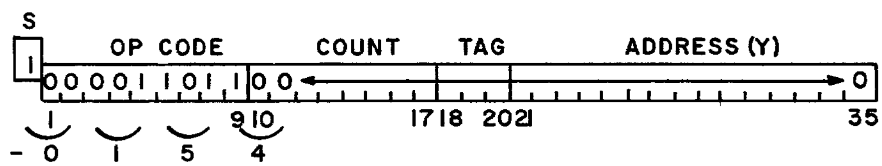
4. Instruction: RND (+0760)



(TYPE E INSTRUCTION)

---

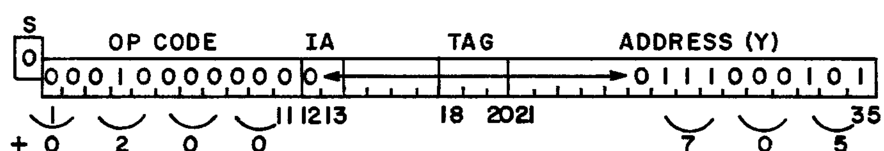
5. Instruction: CRQ (-0154)



(TYPE C INSTRUCTION)

---

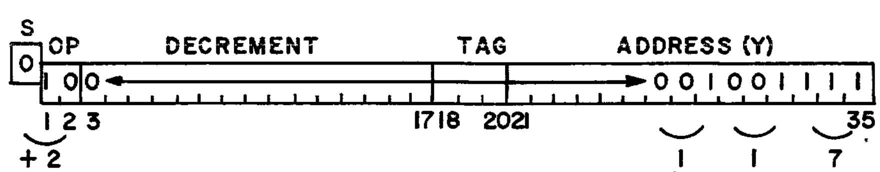
6. Instruction: MPY (+0200) Storage location 705_8



(TYPE B INSTRUCTION)

---

7. Instruction: TIX (+2000) Storage location 117_8



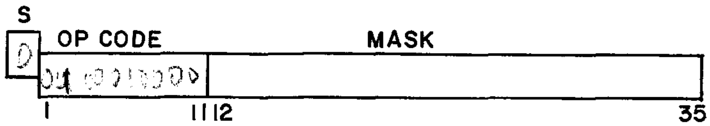
(TYPE A INSTRUCTION)

WORK AREA

Work the problems in this space, then check your answers with the correct answers given on the next page.

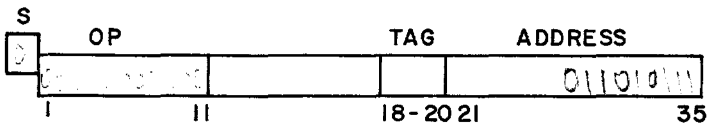
PROBLEMS: Write the instructions and addresses into the words below.

38. Instruction: HPR ($+0420_8$) Type D instruction.



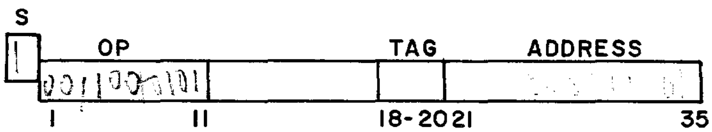
39. Instruction: HTR ($+0000_8$) Storage location 215_{10}
Type B.

$215_{10} = ?_8$



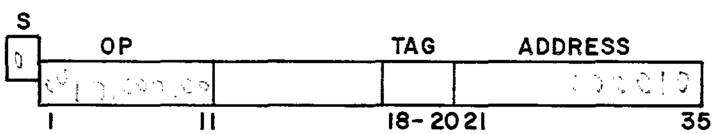
40. Instruction: STL (-0625_8) Storage location 57_{10}
Type B.

$57_{10} = ?_8$

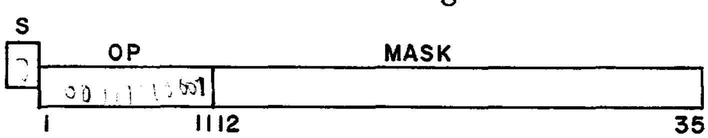


41. Instruction: CLA ($+0500_8$) Storage location 2_{10}
Type B.

$2_{10} = ?_8$

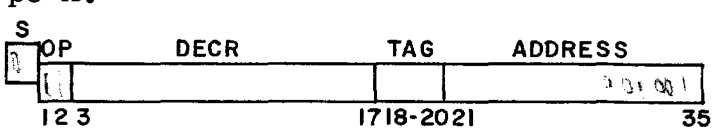


42. Instruction: NOP ($+0761_8$) Type D instruction.



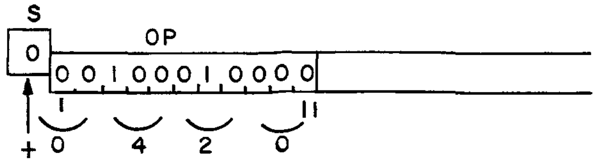
43. Instruction: TXH ($+3000_8$) Storage location 9_{10}
Type A.

$9_{10} = ?_8$

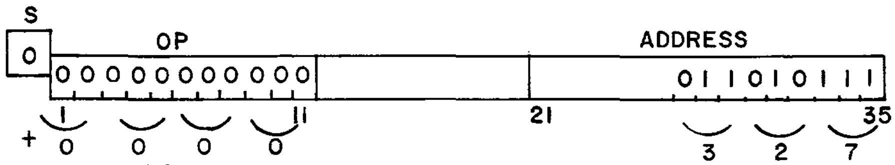


CORRECT ANSWERS

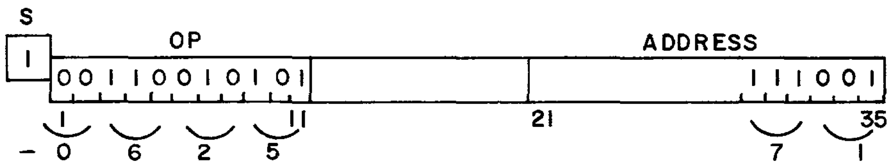
38.



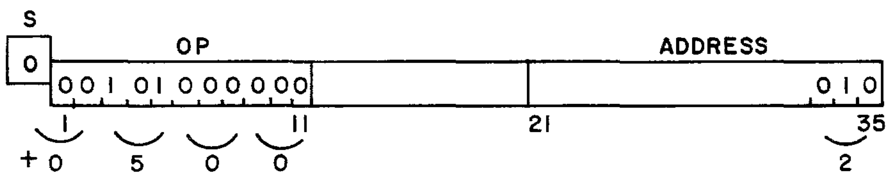
39. $215_{10} = 327_8$



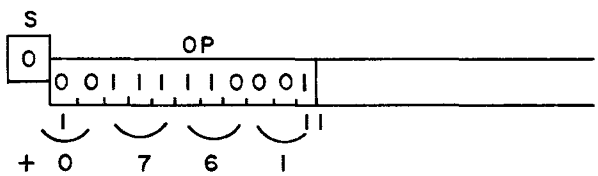
40. $57_{10} = 71_8$



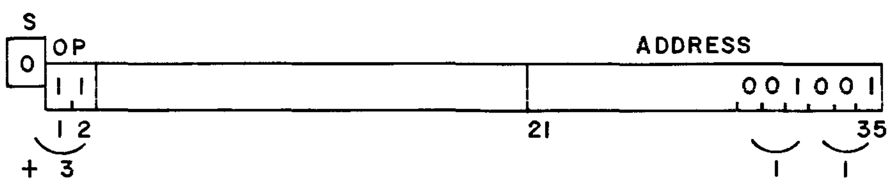
41. $2_{10} = 2_8$



42.



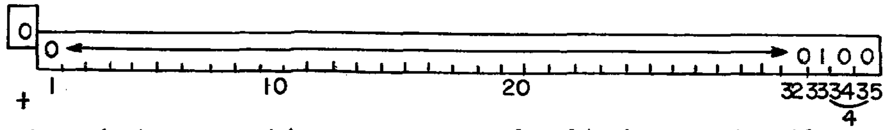
43. $9_{10} = 11_8$



LESSON 3

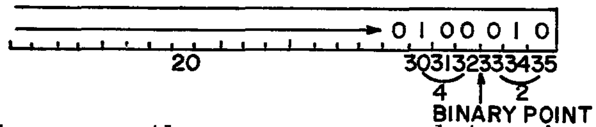
FIXED POINT NUMBERS: When it is easy to determine where a decimal point is to be placed in an arithmetic operation, fixed point instructions are used. A fixed point number contains a sign and 35 bit positions containing the number itself.

($+4_8$)



When fixed point operations are used, it is up to the programmer to decide where the decimal point is to be placed. The point that separates the whole number from the fraction is called a binary point.

($+4.2_8$)



It is called fixed point because the programmer determines the positioning of the point, as opposed to floating point, in which the point is automatically maintained by the computer.

BASIC FIXED POINT OPERATIONS:
INSTRUCTION: CLA (Clear and ADD) Octal code: +0500
FORMAT: (Type B)

Description: The c(Y) replace the c(AC). This means, "The contents of storage location, specified as Y, replace the contents of the Accumulator." Always specify the address of the piece of data, or information, that is to be moved. This tells the computer to move whatever it finds at that address. Positions P and Q are set to zero and the c(Y) remain unchanged.

INSTRUCTION: ADD (Add) Octal code: +0400
FORMAT: (Type B)

Description: The c(Y) are added algebraically to the c(AC) and the sum is placed in the AC. The c(Y) remain unchanged. Numbers of the same magnitude, but with different signs (+2, -2; +407, -407) will result in zero with the original sign of the AC.

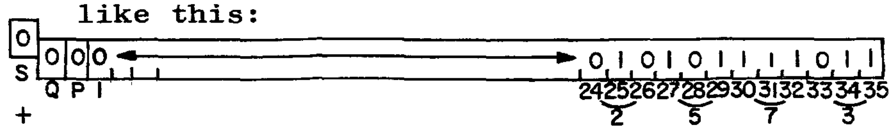
Examples of algebraic addition:

	+5	-5	+5	+3	+5 (Y)	-5 (Y)
	+3	-3	-3	-5	-5 (AC)	+5 (AC)
SUM:	+8	-8	+2	-2	-0	+0

Lesson 3, (cont'd)

Example 1: CLA 25 (This means, "Clear the contents of the AC and place the contents of storage word 25 into the AC.")

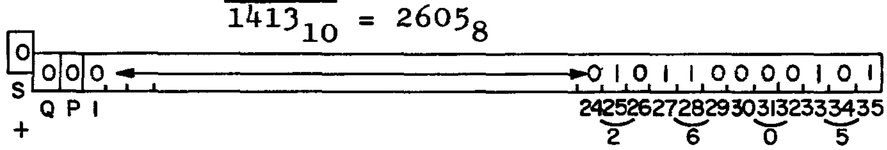
If the c(Y) (storage location 25) = 2573_8, after execution of the instruction the AC would look like this:



Now that we have a number in the AC, we can add another number to it. ADD 27 (This means, "Take the contents of the word at 27 and add it to the contents of the AC.")

If the c(Y) (storage location 27) = 12_8, after execution of the instruction the AC would look like this:

$$
\begin{array}{rrcl}
 & 2573_8 & = & 1403_{10} \\
+ & 12_8 & = & +\ 10_{10} \\
\hline
 & & & 1413_{10} = 2605_8
\end{array}
$$



Example 2:

Storage Location	Data in Storage
15	257_8
22	173_8

Add the two numbers together.

Step 1: Move c(15) into the AC with the CLA instruction.

Step 2: Add c(22) to it. Sum of the two numbers will be in the AC.

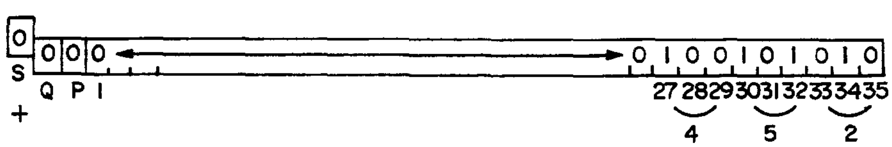
To get the sum (with pencil and paper) that will finally be in the AC, we must convert:

$$
\begin{array}{rcl}
257_8 & = & 175_{10} \\
173_8 & = & 123_{10} \\
\hline
 & = & 298_{10} = 452_8
\end{array}
$$

(Change the Octal to Decimal - add - then change the sum back to Octal)

INSTRUCTIONS: CLA 15
ADD 22

Contents of the AC after execution:



INSTRUCTION: SUB (Subtract) Octal code: +0402

FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

Description: The c(Y) are algebraically subtracted from the c(AC). The difference replaces the c(AC). The c(Y) remain unchanged.

Examples of algebraic subtraction:

	-5	+5	+5	+3	-5	-3	+3	-3	-2	+2
	+3	-3	+3	+5	-3	-5	-5	+5	-2	+2
Diff:	-8	+8	+2	-2	-2	+2	+8	-8	-0	+0

INSTRUCTION: MPY (Multiply) Octal code: +0200

FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

Description: The c(MQ) are multiplied algebraically by the c(Y). The product replaces the c(AC and MQ) with the most significant 35 bits in the AC and the least significant 35 bits in the MQ. Overflow is not possible and the product is positioned to the right with enough leading zeros to completely fill both registers.

Sign Control for algebraic Multiplication

Sign of multiplicand	+	-	+	-
Sign of multiplier	+	+	-	-
Sign of product	+	-	-	+

INSTRUCTION: DVH (Divide or Halt) Octal code: +0220

FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

Description: The c(AC-MQ) are divided algebraically by the c(Y). The quotient replaces the c(MQ) and the remainder replaces the c(AC). If division can not take place (ex: divisor of zero), the computer halts and a "divide-check" indicator turns on. The dividend must be placed into the AC-MQ prior to giving the DIVIDE instruction. If it occupies only one register, the programmer must clear the other, by placing zeros into it.

Sign Control for algebraic Division

Sign of divisor	+	+	-	-
Sign of dividend	+	-	+	-
Sign of quotient	+	-	-	+
Sign of remainder	+	-	+	-

INSTRUCTION: STO (store) Octal code: +0601

FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

Description: The c(AC) replace the c(Y). The sign and bits 1-35 of the AC move into the storage location specified by (Y). The c(AC) remain unchanged.

<u>INSTRUCTION</u>: LDQ (Load MQ Register) Octal code: +0560
<u>FORMAT</u>: (Type B)

OP CODE	IA		TAG	Y
S, 1	11 12-13	18- 20	21	35

<u>Description</u>: The c(Y) replace the c(MQ). The bits at the address (Y) move into the MQ. The c(Y) remain unchanged.

<u>INSTRUCTION</u>: STQ (Store from MQ Register) Octal code: -0600
<u>FORMAT</u>: (Type B)

OP CODE	IA		TAG	Y
S, 1	11 12-13	18-20	21	35

<u>Description</u>: The c(MQ) replace the c(Y). The bits in the MQ move into the storage location specified by (Y). The c(MQ) remain unchanged.

<u>INSTRUCTION</u>: HTR (Halt or Transfer) Octal code: +0000
<u>FORMAT</u>: (Type B)

OP CODE	IA		TAG	Y
S, 1	11 12-13	18- 20	21	35

<u>Description</u>: When this instruction is executed, the computer halts. If the operator presses the START button, the program will continue by going to the (Y) address for its next instruction. If the address given in (Y) is the same as that given for the HALT instruction, the computer will simply do another program stop.

<u>REVIEW AND EXPLANATION OF THE NINE INSTRUCTIONS COVERED</u>:

CLA - used to move data into the AC prior to an operation (i.e. add)
ADD - used to add the c(Y) address to the c(AC).
SUB - used to subtract the c(Y) address from the c(AC).
MPY - used to multiply, but first the multiplicand must be placed into the MQ. This is done with the LDQ instruction.
DVH - used to divide, but first the dividend must be placed into the AC-MQ. This is also done with the LDQ instruction. If we wish to move the quotient back to a storage address, the STQ instruction is used.
STO - used to move the c(AC) to a storage address. This would be used after add or subtract - or if the remainder of a division problem is to be saved. Also if data is to be moved from one storage location to another.
HTR - used to stop the program.

Another way to remember this is:

CLA moves data into the AC from storage.	Y	→	AC
STO moves data into storage from the AC.	AC	→	Y
LDQ moves data into the MQ from storage.	Y	→	MQ
STQ moves data into storage from the MQ.	MQ	→	Y

The other five instructions are: add, subtract, multiply, divide and halt. These are self explanatory.

EXAMPLES	PROGRAM	REMARKS
1. ADD A + B and place sum into storage at pos. 50	Step 1. CLA A 2. ADD B 3. STO 50 4. HTR	Move A into the AC Add B to A Store "sum" into loc. 50 Halt
2. Place the diff. of A-B into storage loc. 150	1. CLA A 2. SUB B 3. STO 150 4. HTR	Move A into AC Subtract B from A Store "difference" into 150 Halt
3. Place the prod. of AxB into loc. 520	1. LDQ A 2. MPY B 3. STO 520 4. HTR	Move A into the MQ Multiply A x B Store "product" into 520 Halt
4. Place the quotient of A÷B into loc. 600. Place "remainder into loc. 20	1. CLA 0 2. LDQ A 3. DVH B 4. STQ 600 5. STO 20 6. HTR	Place zeros in AC prior to Div. Move dividend into AC-MQ Divide A ÷ B Store "quotient" into 600 Store "remainder" into 20 Halt

Note: When the dividend is placed in the MQ (Step 2), the sign of the AC should be made to agree with the sign of the MQ to assure that algebraic division will take place if the dividend is negative. This means that a Long Left Shift of zero should be placed between Steps 2 and 3. This has been omitted here and in pages 34, 36 and 38. Since the Long Left Shift instruction has not yet been studied, it will be presumed that the dividends are positive numbers.

INSTRUCTION: TZE (Transfer on Zero) Octal code: +0100

FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

Description: If the c(AC) is zero, the next instruction is taken from the location specified by (Y). If the c(AC) is not zero, program will take the next instruction in sequence.

INSTRUCTION: TOV (Transfer on Overflow) Octal code: +0140

FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

Description: In addition and subtraction, if an overflow occurs, the AC overflow indicator is turned on. This instruction tests the indicator. If it is "on", it is turned "off" and the next instruction is taken from the location specified by (Y). If the indicator is "off", the program will take the next instruction in sequence.

Do not continue beyond page 36 until the use of these instructions is completely clear to you. If necessary, go back to page 27 and read through the lesson again.

FORMAT FOR WRITING A PROGRAM: In the problems and examples to follow, coding will be accomplished under the following headings:

LOC	OP	ADDRESS	REMARKS

LOC - refers to the storage location of the instruction or data. Instead of referring to "steps", we will assign storage locations to each instruction step.

OP - refers to the operation code.

ADDRESS - refers to the location containing the information or instruction with which the operation is concerned.

REMARKS - refers to a brief explanatory note of what is being accomplished. This is a very handy device for the programmer to use as it gives him a clear picture of what he is doing at all times.

EXAMPLES:

PROGRAM

1. Start the program in loc. 100 and the "if zero" part of the program in loc. 400. A is in loc. 50 and B is in loc. 60.

Place the sum of A + B into loc. 200. If sum is zero, also place the sum into loc. 210.

LOC	OP	ADDRESS	REMARKS
100	CLA	50	Move A into AC
101	ADD	60	Add A + B
102	STO	200	Sum into 200
103	TZE	400	If zero, program jumps to loc. 400 for next instruction.
104	HTR	104	If not zero, halt (loc. address repeated to force halt)
400	STO	210	Sum into 210
401	HTR	401	Halt

2. Start program in loc. 100. A is in loc. 50, B is in loc. 60.

Place the difference of A-B into loc. 200. If overflow occurs, go to loc. 150, place A into loc. 400 and B into loc. 450, then stop the prog.

LOC	OP	ADDRESS	REMARKS
100	CLA	50	Move A into AC
101	SUB	60	Subtract B from A
102	TOV	150	Test for overflow. If "yes", go to loc. 150 for next instr.
103	STO	200	If no overflow, store difference in 200
104	HTR	104	Halt
150	CLA	50	Move A into AC
151	STO	400	Store into loc. 400
152	CLA	60	Move B into AC
153	STO	450	Store into 450
154	HTR	154	Halt

WORK AREA

PROBLEMS:

	A	B	C	D
For all problems, use storage locations:	50	60	70	80

Start all programs in location 100 and any jumps in location 200.

44. Place the sum of A + B into location 400. If the sum is zero, also place A - B into location 300.

LOC	OP	ADDRESS	REMARKS

45. Place the sum of A + B + C into location 425.

LOC	OP	ADDRESS	REMARKS

46. Place the product of B x C into location 350.

LOC	OP	ADDRESS	REMARKS

47. Place the quotient of A ÷ D into location 325. Place the remainder into location 326.

LOC	OP	ADDRESS	REMARKS

CORRECT ANSWERS

PROBLEMS:

44.

LOC	OP	ADDRESS	REMARKS
100	CLA	50	Move A into AC
101	ADD	60	Add B to A
102	STO	400	Place "sum" into 400
103	TZE	200	If sum is zero, jump to 200 for next instr.
104	HTR	104	If not zero, halt.
200	CLA	50	Move A into AC again
201	SUB	60	Subtract B from A
202	STO	300	Place into 300
203	HTR	203	Halt

45.

LOC	OP	ADDRESS	REMARKS
100	CLA	50	Move A into AC
101	ADD	60	Add B to A
102	ADD	70	Add C to sum of B and A
103	STO	425	Place sum into loc. 425
104	HTR	104	Halt

46.

LOC	OP	ADDRESS	REMARKS
100	LDQ	60	Move B into MQ
101	MPY	70	Multiply by C
102	STO	350	Place into loc. 350
103	HTR	103	Halt

47.

LOC	OP	ADDRESS	REMARKS
100	CLA	0	Place zeros into AC
101	LDQ	50	Move A into MQ
102	DVH	80	Divide by D
103	STQ	325	Place quotient into loc. 325
104	STO	326	Place remainder into loc. 326
105	HTR	105	Halt

WORK AREA

PROBLEM: Use the same general instructions as on page 33.

48. Compute: $\frac{A\ B}{C - D}$ if an overflow occurs, place the number 5 (presently in loc. 90) into location 325 and halt. Otherwise, continue the problem and place the quotient into location 400 and the remainder into location 401 (See note below).

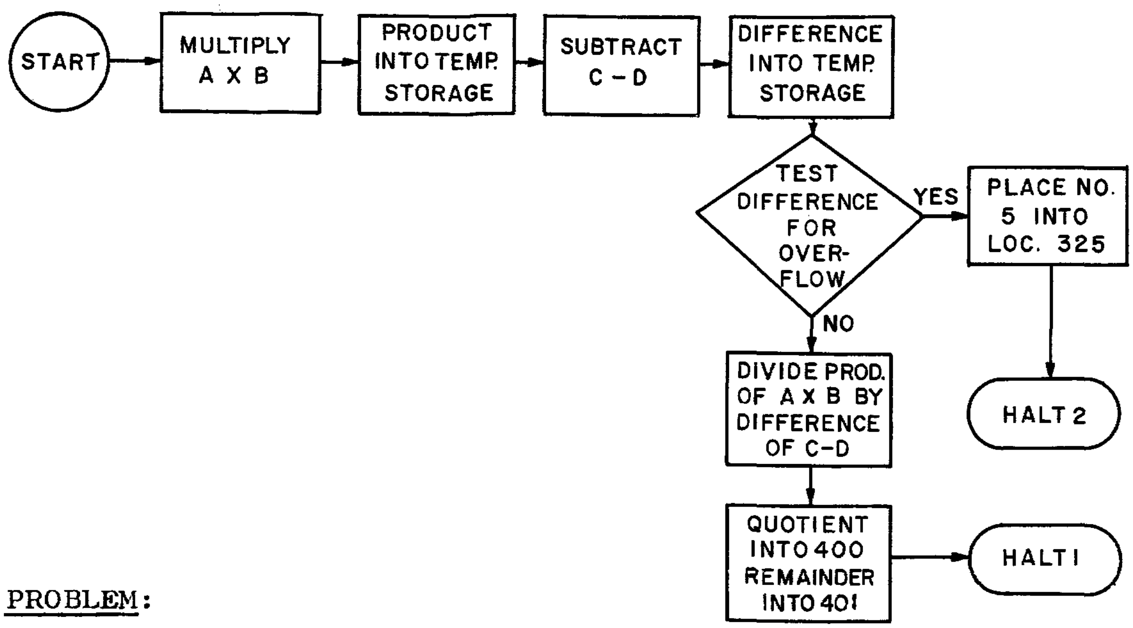
LOC	OP	ADDRESS	REMARKS
100			

NOTE: All arithmetic operations take place in the AC and MQ. If several different operations must be accomplished and the results need to be saved for a later operation, the results are moved to temporary storage locations and recalled from there when needed.

In the above problem, the result of A x B and the result of C - D must both be saved so that the final division may be accomplished. It makes no difference where they are placed in storage as long as those storage locations are not being used for anything else.

CORRECT ANSWERS

When a problem begins to be complicated, it should be flow charted before it is coded. A flow chart of this problem would look like this:



PROBLEM:

48.

LOC	OP	ADDRESS	REMARKS
100	LDQ	50	Move A into MQ
101	MPY	60	Multiply by B
102	STO	600	Place product into temporary loc. 600
103	CLA	70	Move C into AC
104	SUB	80	Subtract D from C
105	STO	650	Place difference into temporary loc. 650
106	TOV	200	Test for overflow. If "yes" jump to loc. 200 for next instruction
107	CLA	0	Move zeros into AC
108	LDQ	600	If no overflow, move result of mult. into MQ so that division may be accomplished.
109	DVH	650	Divide by result of subtraction.
110	STQ	400	Place quotient into loc. 400
111	STO	401	Place remainder into loc. 401
112	HTR	112	Halt - end of job
200	CLA	90	Move "5" into AC
201	STO	325	Place into loc. 325
202	HTR	202	Halt 2 - end of job

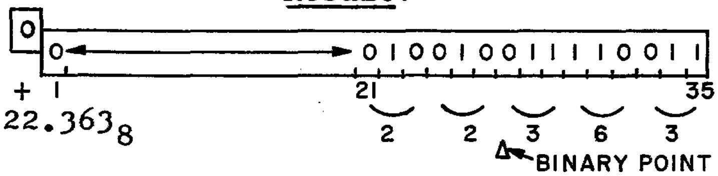
BINARY POINT: It was pointed out on page 27, that the Binary Point must be determined by the programmer when working with Fixed Point numbers. A few examples may clarify this further. Fraction conversion may be accomplished by referring to Appendix C, IBM 7090 Reference Manual.

EXAMPLES:

1. Add:

$$\begin{array}{r} 15.2_{10} \\ 3.274_{10} \\ \hline 18.474_{10} \end{array} = 22.363_8$$

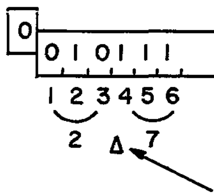
Result:



2. Multiply:

   a. Assign the first Octal integer (whole number) as a flag (converting to Binary)

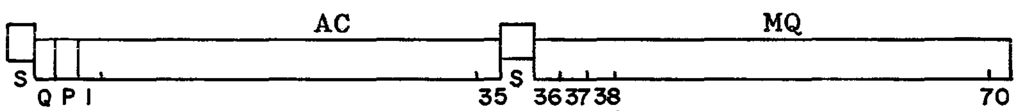
EXAMPLE: 2.7_8



Binary point between positions 3 and 4. Flag is 3 (we will call it B 3 - the B representing Binary and also indicating fixed point numbers)

   b. Set up a flag for both multiplicand and multiplier.

   c. After multiplication, consider the AC and MQ as one long 70 bit register (no count is taken of the S, Q, P in the AC, or the S in the MQ). The position of the Binary Point will be the sum of the two flags.

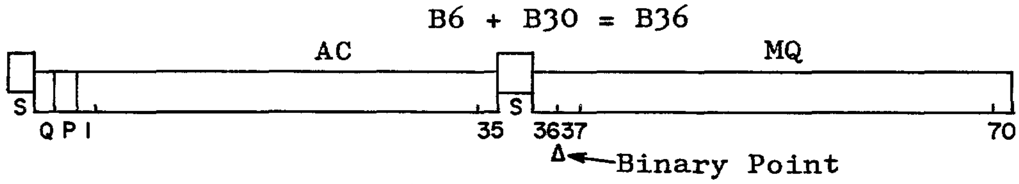


EXAMPLE: B 3 and B 35 = B 38 (this is the location of the B.P. in product.)

EXAMPLES:

1. Multiply a B6 number and a B30 number. Show the location of the Binary Point in the AC - MQ.

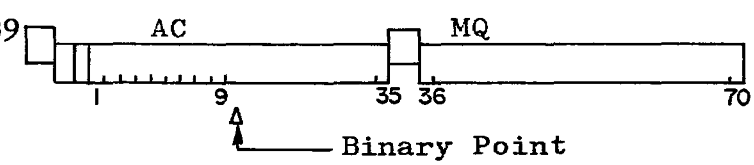
B6 + B30 = B36



2. Multiply 32.657_8 by 3.44444_8. Show the location of the Binary Point of the product in the AC - MQ.

Step 1: 32.657 (two Octal integers before the fraction)
Flag = B 6
3.44444 (one Octal integer before the fraction)
Flag = B 3

Step 2: B6 + B3 = B9

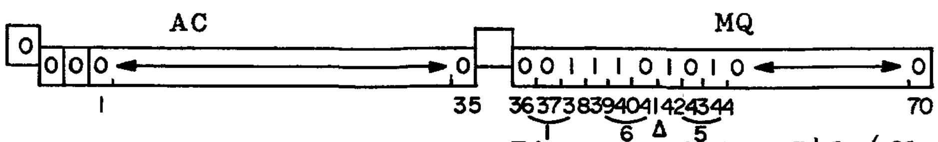


Lesson 3, (cont'd)

3. Division: Assign flags for both the divisor and the dividend, as in multiply operations. However, when the dividend is brought into the AC-MQ, consider both AC and MQ as a 70 bit register with the flag located in the proper position of the 70.

EXAMPLE: The dividend is 16.5_8

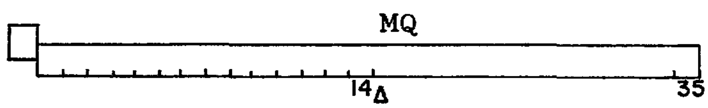
The first step is to clear the AC (CLA 0)
The second step is to load the dividend into AC-MQ (LDQ X)



Binary point = B41 (flag)

The quotient will be in the MQ and the remainder in the AC, so after divide has been accomplished, the MQ must again be considered as a 35 bit register.

If we divide by a B27 number, the quotient in the MQ will be 41-27=B14



EXAMPLE 2: If the divisor happens to be quite small, it is possible to lose a portion of the quotient.

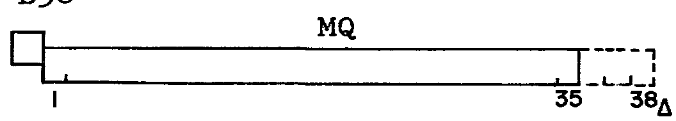
Divide 274.555_8 by 15.321_8

Step 1: Dividend 274.555_8 = B9 (3 Octal = 9 Binary)

Step 2: Move to MQ = B44 (35 in AC + 9 in MQ = 44)

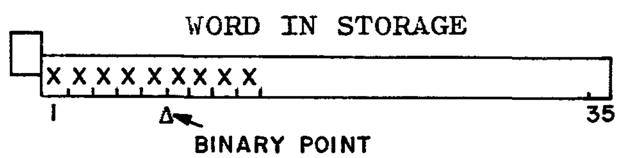
Step 3: Divide by a B6 number (15.321_8 = B6 2 Octal = 6 Binary)

Step 4: B44 - B6 = B38



The three trailing positions would be lost

In these examples and in the problems that follow, all numbers are assumed to be left adjusted. This means that the digits are located in the extreme left part of the word in storage.



WORK AREA

PROBLEMS: (All hypothetical numbers will be considered to be in Octal, left adjusted and all dividends loaded into the MQ)

Multiply and show location of Binary point of the product in AC - MQ.

49. XXX.X by XX.XXX

Product: B

50. .XXXXX by .XXXX

Product: B

51. X.XXXXX by X.XXXX

Product: B

52. XXXX. by XXXX.

Product: B

53. XXXXXXXXXX.X by XXX.X

Product: B

54. XXXXXXXXXXXX.XXX by XXXX.XXX

Product: B

55. .XXX by X.XXX

Product: B

Divide and show location of Binary point of the quotient in the MQ.

56. X.XX) XXX.XX

Quotient: B

57. .XXXXXXX) XXXXX.XXX

Quotient: B

58. XXXXXXXX.X) XXXXX.XXXX

Quotient: B

59. .XX) XXX.XX

Quotient: B

60. XXXXXXXXXXX) XX.XX

Quotient: B

61. .X) .X

Quotient: B

62. XX.X) XX.XX

Quotient: B

CORRECT ANSWERS

PROBLEMS:

49. B9 + B6 = B15

50. B0 + B0 = B0

51. B3 + B3 = B6

52. B12 + B12 = B24

53. B30 + B9 = B39

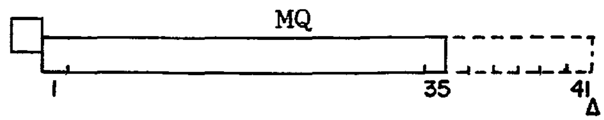
54. B36 + B12 = B48

55. B0 + B3 = B3

56. B3 ⟌ B9

B35 + B9 = B44 (after move into MQ)

B44 - B3 = B41



57. B21 ⟌ B15

B35 + B15 = B50 (dividend)

B50 - B21 = B29 (in MQ)

58. B24 ⟌ B15

B35 + B15 = B50

B50 - B24 = B26

59. B0 ⟌ B9

B35 + B9 = B44

B44 - B0 = B44 (9 trailing positions would be lost)

60. B33 ⟌ B6

B35 + B6 = B41

B41 - B33 = B8

61. B0 ⟌ B0

B35 + B0 = B35

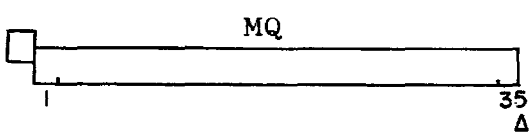
B35 - B0 = B35

62. B6 ⟌ B6

B35 + B6 = B41

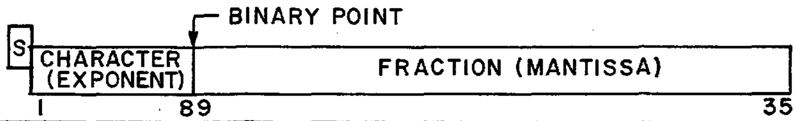
B41 - B6 = B35

Problems 61 and 62 show that whenever the dividend and divisor have the same "B" number, the Binary Point will be at B35 in the MQ.



LESSON 4

FLOATING POINT NUMBERS: If the range of numbers in a calculation is apt to be quite large or unpredictable, fixed point numbers no longer serve the purpose because it becomes impossible to calculate the position of the Binary Point. An alternative set of Floating Point instructions are available and should be used for such calculations. With these instructions, the Binary Point is automatically maintained between the 8th and 9th digit of the word. A Floating Point number is stored in a word as shown below:



The fraction (also called the Mantissa) is always stored in positions 9-35 and the characteristic (exponent) is in positions 1-8. This portion must be explained in more detail:

A floating point number may be expressed as a signed proper fraction multiplied by some power of 10. The number is normal (or normalized) if the power is chosen in such a way that the decimal point is to the left of the most significant digit.

EXAMPLES:

$.350_{10} = .35 \times 10^0$

$3.50_{10} = .35 \times 10^1$

$35.0_{10} = .35 \times 10^2$

$350._{10} = .35 \times 10^3$

Note that the powers of 1, 2, and 3 are in direct ratio to the number of places the decimal point is moved to the left.

- - - - - - - - - - - - - - -

$.035_{10} = .35 \times 10^{-1}$

$.0035_{10} = .35 \times 10^{-2}$

If the decimal point is moved to the right, it works in the same way except, that the power is then negative.

A floating point Binary number may be expressed in the same manner as the Decimal numbers above except that it will be multiplied by some power of 2.

EXAMPLES:

$.001 = .100 \times 2^{-2}$ (Binary point moved two positions to right)

$100.000 = .100 \times 2^{3}$ (Binary point moved three positions to left)

If the number is normal, bit position 9 will always be a 1. If it is not normal, bit position 9 will always be a zero.

The characteristic is formed by adding +128 to the exponent (the exponent being the number of the power). Converting to Octal: +128 = +200.

EXAMPLE: $5._{10} = 5._8 = 101._2$ $\quad 101. = .101 \times 2^3$ Add 200 +3 to go into the characteristic. The fraction goes into the Mantissa portion of the word.

Since there are only eight positions in the characteristic, the leftmost Binary position is dropped. If an overflow occurs, it may be checked by the program.

Lesson 4, (cont'd)

ADDITIONAL EXAMPLES:

1. Show normalized, floating point 10_{10} as it would look in a machine word.

Step 1: Chg from Dec to Oct to Bin
$10_{10} = 12_8 = 001\ 010._2$

Step 2: Move Binary point
$001\ 010. = .1010 \times 2^4$

Step 3: Add Exp to Oct 200
$200 + 4 = 204$

Final Step: Move result of Step 3 into Charact. Move result of Step 2 into Mantissa

0	10000100	1010 ←——→ 0
	+2 0 4	35

Note that although the Octal number was 12_8, in the Mantissa, it now looks like a 5. The programmer must be aware of this apparent change.

---

2. Show normalized floating point $.0039_{10}$ as it would look in a machine word.

Step 1: $.0039_{10} = .002_8 = .000\ 000\ 010_2$

Step 2: $.000\ 000\ 010 = .10 \times 2^{-7}$

Step 3: $200_8 - 7_8 = 128_{10} - 7_{10} = 121_{10}$
$= 171_8$

Step 4: Step 3 into Charact. Step 2 into Mantissa

0	01111001	100 ←——→ 0
	+1 7 1	35

When the exponent will be a minus, the Octal 200 and the exponent must be converted to Decimal, then converted back to Octal.

---

3. Show normalized floating point 44_{10} as it would look in a machine word.

Step 1: $44._{10} = 54._8 = 101\ 100._2$

Step 2: $101\ 100. = .101100 \times 2^6$

Step 3: $200 + 6 = 206_8$

Final Step:

0	10000110	10110 ←——→ 0
	+2 0 6	35

---

4. Show normalized floating point -20_{10} as it would look in a machine word.

Step 1: $-20._{10} = -24._8 = -010\ 100._2$

Step 2: $-010\ 100. = -.10100 \times 2^5$

Step 3: $200 + 5 = 205_8$

Final Step:

0	10000101	1010 ←——→ 0
	+2 0 5	35

Warning: Be sure to remember that the Characteristic is always derived in Octal. Very bad mistakes can be made if the exponent is not converted to Octal before adding to 200_8.

WORK AREA

PROBLEMS:

63. Show normalized floating point 3_{10} as it would look in a machine word.

Step 1:

Step 2:

Step 3:

Final Step:

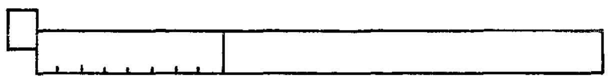
64. Show normalized floating point $.003_{10}$ as it would look in a machine word.

Step 1:

Step 2:

Step 3:

Final Step:



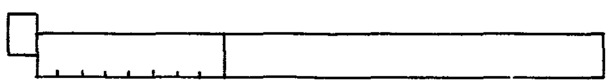
65. Show normalized floating point 232_{10} as it would look in a machine word.

Step 1:

Step 2:

Step 3:

Final Step:



CORRECT ANSWERS

PROBLEMS:

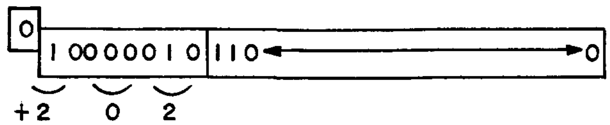
63.

Step 1: $3_{10} = 3_8 = 011._2$

Step 2: $011. = .11 \times 2^2$

Step 3: $200 + 2 = 202_8$

Final Step:



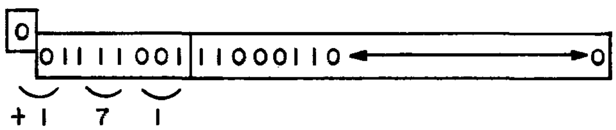
64.

Step 1: $.003_{10} = .00306_8 = .000\ 000\ 011\ 000\ 110_2$

Step 2: $= .1100011 \times 2^{-7}$

Step 3: $200_8 - 7_8 = 128_{10} - 7_{10} = 121_{10} = 171_8$

Final Step:



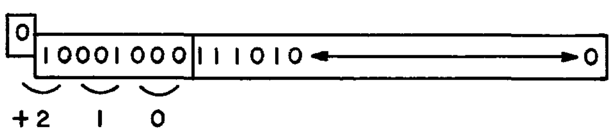
65.

Step 1: $232_{10} = 350_8 = 011\ 101\ 000._2$

Step 2: $= .11101 \times 2^{10}$ (8 not permitted in Octal - always jumps to 10 after 7)

Step 3: $200 + 10 = 210_8$

Final Step:



The material covered on page 41 may be entirely new to the student, even to the terms "power", "exponent","normalized", etc. If it is new, please go over it a second time and make up some additional problems to get extra practice in this area. Use the Octal/Decimal conversion tables (Appendix B and C) in the 7090 Reference Manual to speed up the conversion of both integers (whole numbers) and fractions.

FLOATING POINT ARITHMETIC: The location of the decimal point, or Binary point, is an extremely important problem in programming. Just as in "pencil-and-paper" arithmetic, decimal points must be lined up and additional zeros must be added where required.

EXAMPLE: Add + 100.0 and - 0.1002. If they were not lined up, they would look like this:

$$\begin{array}{l} +\ 100.0 \\ -\ 0.1002 \end{array}$$

To line them up, the lower number must be shifted to the right two places and three trailing zeros must be added to the upper number:

$$\begin{array}{lr} + & 100.0\boxed{000} \\ - & 0.1\ 002 \end{array}$$

The same numbers in floating point (Decimal) form, normalized, would look like this:

$$\begin{array}{l} +\ .1000 \times 10^3 \\ -\ .1002 \times 10^0 \end{array}$$

To equalize the exponent, the lower number is again shifted to the right and trailing zeros are added to the upper number, as follows:

$$\begin{array}{l} +\ .1000\ \boxed{000}\ \times 10^3 \\ -\ .\boxed{000}1\ 002\ \times 10^3 \end{array}$$

If the result is not <u>normal</u>, it must be shifted right to finish with a normalized number:

$$\begin{array}{r} +.1000000 \times 10^3 \\ -.0001002 \times 10^3 \\ \hline .0998998 \times 10^3 \end{array} = .998998 \times 10^2$$

Since the programmer does not usually work with actual numbers, but with quantities where only the maximum and minimum size is known, the problem becomes much greater. This text must, of necessity, be limited to fundamentals of 7090 programming, therefore this will not be covered in detail here.

FLOATING POINT OPERATIONS:

INSTRUCTION: FAD (Floating ADD) Octal code: +0300

FORMAT: (Type B)

OP CODE	IA		TAG	Y

S,1 11 12-13 18-20 21 35

Description: The floating point number in Y is added algebraically to the floating point number in the AC. The most significant part of the number is in the AC as a normalized floating point number and the least significant part is in the MQ as a floating point number with a characteristic 33 less than the AC characteristic. The sign in both the AC and MQ will be that of the larger factor.

INSTRUCTION: FSB (Floating Subtract) Octal code: +0302

FORMAT: (Type B)

OP CODE	IA		TAG	Y

S,1 11 12-13 18- 20 21 35

Description: The floating point number in Y is subtracted algebraically from the floating point number in the AC. The result is always normalized and located in the AC.

INSTRUCTION: FMP (Floating Multiply) Octal code: +0260

FORMAT: (Type B)

OP CODE	IA		TAG	Y

S,1 11 12-13 18-20 21 35

Description: The floating point number in Y is multiplied algebraically by the floating point number in the MQ. The most significant part of the product will be in the AC and the least significant part in the MQ. If either of the numbers is not normalized, the product may or may not be in normalized form.

INSTRUCTION: FDH (Floating Divide or Halt) Octal code: +0240

FORMAT: (Type B)

OP CODE	IA		TAG	Y

S,1 11 12-13 18-20 21 35

Description: The c(AC) are divided algebraically by the c(Y). The quotient will be in the MQ and the remainder will be in the AC. If the size of the fractional part of the AC is equal to or greater than twice the fractional part of the number in Y (this would only happen in unnormalized numbers), or if the number in Y is zero, the Divide Check Indicator turns on and the computer stops. The quotient will be in normalized form if both the dividend and divisor were normalized.

INSTRUCTION: ALS (Accumulator Left Shift) Octal code: +0767
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

Description: The c(AC) are shifted to the left the number of places specified in positions 28-35 of the address portion of the instruction. Vacated positions are automatically filled with zeros. If the instruction calls for a shift larger than the bit capacity of the AC, it will be completely filled with zeros.

INSTRUCTION: ARS (Accumulator Right Shift) Octal code:+0071
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

Description: Identical to the ALS instruction except that the shift is to the right.

INSTRUCTION: TPL (Transfer on Plus) Octal code: +0120
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

Description: If the sign position of the AC is positive (Binary zero), the computer takes its next instruction from location Y. If the sign is negative (Binary one), the computer goes to the next instruction in sequence.

INSTRUCTION: TMI (Transfer on Minus) Octal code: -0120
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

Description: If the sign position of the AC is negative (Binary one), the computer takes its next instruction from location Y. If it is positive (Binary zero), the computer goes to the next instruction in sequence.

INSTRUCTION: XCA (Exchange AC and MQ) Octal code: +0131
FORMAT: (Type D)

OP CODE	
S,1	11

Description: The c(AC) and the c(MQ) are exchanged. Positions P and Q in the AC are cleared to zeros.

Lesson 4, (cont'd)

EXAMPLES: Use storage locations as follows:

A	B	C	D
50	60	70	80

Start program in location 100 and any jumps in location 200.

1. AB + CD = T (floating point numbers) Place T into location 400. If T is positive, compute A-B and place the difference into location 450.

LOC	OP	ADDRESS	REMARKS
100	LDQ	50	Move A into MQ
101	FMP	60	Multiply A x B
102	STO	300	Product stored in temporary loc.
103	LDQ	70	Move C into MQ
104	FMP	80	Multiply C x D
105	FAD	300	Add product of AB to product of CD
106	STO	400	Place sum (T) into location 400
107	TPL	200	If T is +, go to loc. 200 next instr.
108	HTR	108	If T not +, end of program

200	CLA	50	Move A into AC
201	FSB	60	Subtract B from A
202	STO	450	Place difference into loc. 450
203	HTR	203	Halt 2 - end of program

---

2. $\frac{A}{D}$ = P (floating point numbers) If P is negative, place into location 400. If not negative, place into location 450 and do not place into location 400.

LOC	OP	ADDRESS	REMARKS
100	LDQ	50	Move A into MQ
101	FDH	80	Divide by D
102	TMI	200	If sign of quotient is -, go to loc. 200 next instruction
103	STQ	450	If sign not -, store quotient into location 450
104	HTR	104	Halt - end of job

200	STQ	400	Store quotient into loc. 400
201	HTR	201	Halt 2 - end of job

EXAMPLES Continued:

3. $A - B = W$ (floating point) Place W into location 900.

LOC	OP	ADDRESS	REMARKS
100	CLA	50	Move A into AC
101	FSB	60	Subtract B from A
102	STO	900	Store difference into loc. 900
103	HTR	103	Halt - end of job

PROBLEM: (Storage locations the same as for examples on page 48)
(Also, X in location 90)

66. $\frac{A + BX}{C} + X^2 = T$ (floating point) Place T into location 500.

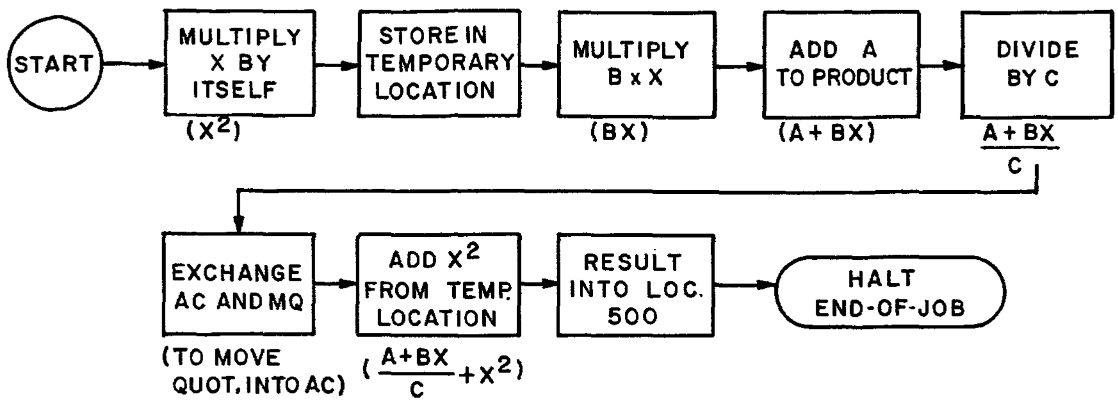
Flow chart the problem before attempting to code it.

LOC	OP	ADDRESS	REMARKS
100			

CORRECT ANSWER

PROBLEM:

66.



LOC	OP	ADDRESS	REMARKS
100	LDQ	90	Move X into MQ
101	FMP	90	Multiply X by X to get X^2
102	STO	300	Store result in temporary loc.
103	LDQ	60	Move B into MQ
104	FMP	90	Multiply B by X
105	FAD	50	Add A to product
106	FDH	70	Divide by C
107	XCA		To move quotient from MQ to AC
108	FAD	300	Add X^2 to previous total
109	STO	500	Store final result into loc. 500
110	HTR	110	Halt - end of job

If the XCA instruction had not been used (loc. 107), it would have taken two instructions to move the quotient from the MQ into the AC, so that addition could be accomplished. The alternative would have been to move the quotient from the MQ into storage and back from storage into the AC. Use of the XCA simplifies this for us.

OVERFLOW AND UNDERFLOW: The "characteristic" contains eight bit positions. If all eight were filled with ones, the resultant number would be $377_8 = 255_{10}$. We add $+128_{10}$ to the exponent to derive the characteristic, therefore any characteristic larger than $+177_8$ ($+127_{10}$) would cause an overflow (the result is too large for storage to contain). Also, any characteristic below -200_8 (-128_{10}) would cause an underflow.

The maximum and minimum characteristic capability of the machine is $+127_{10}$ and -128_{10}. If these figures are exceeded, the computer will put the address plus one of the instruction causing the trouble into the address portion of location 0000. An identifying code which tells whether an overflow or underflow occurred and whether the most significant result is in the AC or MQ, is placed in the Decrement portion of location 0000. The computer continues by executing the instruction at 0010_8, and continuing on from there. This is called a floating point trap and the overflows and underflows are called spills.

The spill codes are as follows:

Operation	AC	MQ	Decrement
Add, Subtract		underflow	0 0 0 1
Multiply	underflow	underflow	0 0 1 1
Round	overflow		0 1 1 0
Round	overflow	overflow	0 1 1 1
Divide		underflow	1 0 0 1
Divide	underflow		1 0 1 0
Divide	underflow	underflow	1 0 1 1
Divide		overflow	1 1 0 1

These codes are used to aid the programmer in checking for overflow and underflow conditions. The programmer places a transfer instruction in location 0010_8, transferring the program to a routine which is designed to take care of the overflow or underflow condition. Every programming group has such a routine developed and ready for use with most programs.

Lesson 4, (cont'd)

INSTRUCTION: NZT (Storage Not Zero Test) Octal code: -0520

FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1 — 11	12-13		18-20	21 — 35

Description: If the contents of Y are not zero, the computer skips one instruction. If the c(Y) are zero, the computer takes the next instruction in sequence. The c(Y) remain unchanged.

EXAMPLE: (Use storage locations as in the earlier examples)

If location 400 contains zeros, place the sum of floating point A + B into it. If it does not contain zeros, place the sum of A + B into location 600. Show a partial program to accomplish this action.

LOC	OP	ADDRESS	REMARKS
100	CLA	50	Move A into AC
101	FAD	60	Add B to A
102	NZT	400	Test Loc. 400 for zeros
103	TRA	200	If zeros in 400, trans. to location 200
104	STO	600	Store sum in 600 (since the test showed no zeros - or the program would never get this instruction)
105	HTR	105	Halt - end of job
200	STO	400	Store sum into 400
201	HTR	201	Halt 2 - end of job

Lesson 4, (cont'd)

INSTRUCTION: ZET (Storage Zero Test) Octal Code: +0520

FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

Description: If the contents of Y are zero, the computer skips one instruction. If the c(Y) are not zero, the computer takes the next instruction in sequence. The c(Y) remain unchanged. This is exactly the reverse of the NZT instruction.

EXAMPLE: (Use storage locations as in earlier examples)

If location 400 contains zeros, place the sum of floating point A + B into it. If it does not contain zeros, place the sum of A + B into location 600. Show a partial program to accomplish this action. (Note: This is the same problem as the one on page 52, but notice the difference in the program when the ZET instruction is used).

LOC	OP	ADDRESS	REMARKS
100	CLA	50	Move A into AC
101	FAD	60	Add B to A
102	ZET	400	Test loc. 400 for zeros
103	TRA	200	If not zero in 400, transfer to 200
104	STO	400	Store sum into 400 (since test must have showed zeros - or the program would not have reached this instr.)
105	HTR	105	Halt - end of job
200	STO	600	Store sum into 600
201	HTR	201	Halt 2 - end of job

<u>MOST SIGNIFICANT AND LEAST SIGNIFICANT</u>: In Floating Add and Floating Multiply, the statement was made that the most significant part of the result would be in the AC and the least significant part in the MQ.

Under normal circumstances, the sum and/or the product may be considered to be in the AC and the STO (Store) instruction is used to move the data back into memory.

Occasionally this will cause trouble for the following reason; in a floating point number, the first eight positions of the word are taken up by the Characteristic. This leaves 27 positions (or 9 Octal digits) for the sum or product. If the numbers to be added or multiplied are large enough to result in a sum or product larger than 9 Octal digits, the least significant portion will be in the MQ and will be lost if the data is moved back into storage with a STORE instruction.

<u>EXAMPLE</u>: Add (Octal)

```
                1000000000.
              +          1.0012
              ----------------------
        SUM:    1000000001.00120000
```

This will be in AC since only 9 Octal will fit into frac.

This will be in MQ, also in the fraction part (posit. 9-35)

In this case, if only the STO instruction were used, a very important part of the number would be lost. The STQ must also be used and the total sum stored in two words since it is too large to fit into one word.

Unless the programmer knows that this is an important factor in a particular program, he may forget about the least significant portion, but if it is important to save fractions to the very last point, then he must arrange in the program to protect the result of FAD and FMP.

LESSON 5

SYMBOLIC CODING: We have been using symbolic operation codes because they are simpler and more easy to remember than either Binary or Octal (ADD is more easily remembered than $+0400_8$ or the Binary form: 000 100 000 000). Symblic coding may also be used in the other parts of an instruction (address, tag, decrement).

In a large program, keeping track of actual addresses can become extremely difficult and error prone. If the programmer wanted to add A + B, it would be much simpler to write: CLA A; ADD B, than to assign specific storage locations to each symbol.

When writing programs in "symbolic," every symbol used in the program must be defined by the programmer, preferably at the end of the program. This is most easily explained with an example:

EXAMPLE: Add A + B. Store sum in HOLD.

LOC	OP	ADDRESS	REMARKS
	CLA	A	Move "A" into AC
	ADD	B	Add A + B
	STO	HOLD	Sum to loc. HOLD
END	HTR	END	Halt (since actual locations are no longer used, placing the same symbol (END) in both LOC and ADDRESS, accomplishes the purpose of permanently halting the program.
A	BSS	1	Allocates 1 Word of Storage to "A"
B	BSS	1	Allocates 1 Word of Storage to "B"
HOLD	BSS	1	Allocates 1 Word of Storage to "HOLD"

The BSS instruction is a pseudo-instruction (explained on page 60). The important thing to remember is that any symbol used in the address, tag, or decrement part of an instruction must be defined in the LOC field, as shown above.

The symbol itself may be anything the programmer desires, but it must be six characters or less, in length and there must be at least one non-numeric character.

If a symbol is used in the address field, but does not appear in the location field, it is undefined. If it appears more than once in the location field, it is multiple-defined. In either case, the program will not be executed by the computer.

ADDITIONAL EXAMPLE:

1. Store into symbolic location X, the sum of the contents of locations A and B. If the sum is zero, do not store the sum into X. Instead, store the contents of symbolic location FEED 1 into X.

LOC	OP	ADDRESS	REMARKS
	CLA	A	Move "A" into AC
	ADD	B	Add A + B
	TZE	JUMP	If sum is zero, go to JUMP for next instruction
STORE	STO	X	If not zero, store sum into X
STOP	HTR	STOP	Halt - end of job
JUMP	CLA	FEED 1	Move "FEED 1" into AC
	TRA	STORE	Transfer back to loc. STORE, which will now move FEED 1 (which is in the AC) into X.
A	BSS	1	$\left.\right\}$ Assigns storage Locations to A, B, and FEED 1
B	BSS	1	
FEED 1	BSS	1	

Notice that each symbol in the address field has just one counterpart in the location field.

The word JUMP was arbitrarily used to jump the program if the sum was zero. Any other symbol would have worked just as well, as long as it was carried over to the location field (J1, XYZ, or whatever).

A symbol was placed in the location field of the STORE instruction so that a return could be made on the TRANSFER. This was not absolutely necessary, but it saved two instructions, because after moving FEED 1 into the AC, we have to store it into X, then Halt. These two instructions were already available to us, therefore it was not necessary to repeat them. This little procedure is called a loop.*

Any symbol may be used in the address field as many times as is necessary, but it may only be defined in the location field once.

Study the above example until it is completely clear before continuing with the problems on page 57.

* If a certain group of instructions are to be executed several times during the course of a program, it is extremely wasteful to repeat the instructions over and over again. It is more practical to include a few instructions that will take care of any necessary modifications and that will allow the single set of instructions to be used repeatedly but coded only once. The example above is not really a loop since it is not to be repeated over and over, but it will give the student an idea of how a loop works without going into the details of address modification.

WORK AREA

PROBLEMS: Write in "symbolic".

67. Compute A - B. If an overflow occurs, store result in Y. Otherwise store result in Z.

LOC	OP	ADDRESS	REMARKS

68. Compute AX + X^2 (fixed length numbers). Place the sum into symbolic location T. If the sum is zero, place the contents of symbolic location P into T instead of the sum of the original computation.

LOC	OP	ADDRESS	REMARKS

CORRECT ANSWERS

PROBLEMS:

67.

LOC	OP	ADDRESS	REMARKS
	CLA	A	Move "A" into AC
	SUB	B	Subtract A - B
	TOV	J1	If overflow, go to J1 for next instr.
	STO	Z	No overflow, store result into Z
END	HTR	END	Halt - end of job
J1	STO	Y	Store result into Y
	TRA	END	Transfer to loc. END, which halts the program.
A	BSS	1	Allocate store locations to A, B, Y and Z
B	BSS	1	
Y	BSS	1	
Z	BSS	1	

68.

LOC	OP	ADDRESS	REMARKS
	LDQ	X	Move X into MQ
	MPY	X	Multiply by X (X^2)
	STO	HOLD	Move into temporary storage loc.
	LDQ	A	Move "A" into MQ
MPY	MPY	X	Multiply by X
	ADD	HOLD	Add X^2 to product
	TZE	J1	If sum is zero, go to J1 for next instr.
STORE	STO	T	If not zero, place sum into T
END	HTR	END	Halt - end of job
J1	CLA	P	Move "P" into AC
	TRA	STORE	Go to loc. STORE for next instr. (this will move "P" (now in AC) into "T", then halt)
A	BSS	1	Allocate Store locations to A, X, T and P
X	BSS	1	
T	BSS	1	
P	BSS	1	

SYMBOLIC CODING SHEET: Pre-printed sheets are available to the programmer to be used in program writing. The following is a typical coding sheet layout:

709 SYMBOLIC CODING SHEET				
PROGRAM____ JOB NO.____ DATE____ PAGE__ OF__				
LOCATION 1 6	7	OPERATION 8 15	ADDRESS, TAG, DECREMENT 16 COMMENTS 72	IDENTIFICATION 73 80

LOCATION - Start in Column 1. A symbol is used here if the program is to refer back to a previous instruction (example: pag. 56, TRA STORE). Also used to jump the program away from the normal flow (example: pg 56, TZE JUMP). Also used to define any symbol used in the original problem (example: pg 56, BSS A, D, FEED 1).

OPERATION - Start in Column 8. Symbolic Op Code, 3 to 7 characters in length.

ADDRESS, TAG, DECREMENT - May start in column 12, but better to always start in column 16. There must be at least one blank between Op Code and the variable field. Address, tag and decrement are to be separated by commas. If remarks are used, separate from the last variable field by a blank.

IDENTIFICATION - First card is generally marked with a descriptive title and the rest of the cards numbered sequentially (0000, 0010, 0020, etc.). There are two reasons for this: (1) if you wish to find a card in a large program, it is easy to check the number on the program print-out and go right to it and (2) if the card deck should accidentally be dropped, it can easily be sorted back into order. Columns 73-80 may be left blank.

In writing the little practice problems, always remember that if coding sheets were available, LOC would start in column 1, OP would start in column 8 and ADDRESS, in column 16.

Special symbols must be used for the assembly program to recognize symbol arithmetic in the variable field. The symbols are as follows:

Add	+
Subtract	-
Multiply	*
Divide	/

Multiply A by B, may no longer be written AB, as in normal algebraic notation. It must be written: A * B.

PSEUDO OPERATION CODES: These codes are so named because they are not true machine operation codes. They do not have an Octal equivalent and they do not become a part of the actual program. They are instructions to (FAP) the assembly program (which will change the symbolic program written by the programmer into Binary form), executed by the assembly program and then forgotten.

PSEUDO OP: COUNT
DESCRIPTION: This must be the first card of the symbolic card deck (refer to page x in introduction). The total number of cards in the program deck will be specified in the address field - written as a decimal integer.

PSEUDO OP: END
DESCRIPTION: This must be the last card of the symbolic card deck. It tells the assembly program that the symbolic program being converted to Binary, is finished. The END card must be in every program.

PSEUDO OP: BSS (Block Started by Symbol)
DESCRIPTION: This causes the assembly program to reserve a block of storage locations. The number specified in the address portion of the instruction indicates the number of storage locations to be reserved. It does not indicate a storage address (see pages 55 and 56 for examples of BSS).

The programmer should not assume that locations reserved by BSS contain zeros. It is always safer to use the STZ instruction to clear these areas out prior to using them for processing.

WORK AREA

PROBLEM:

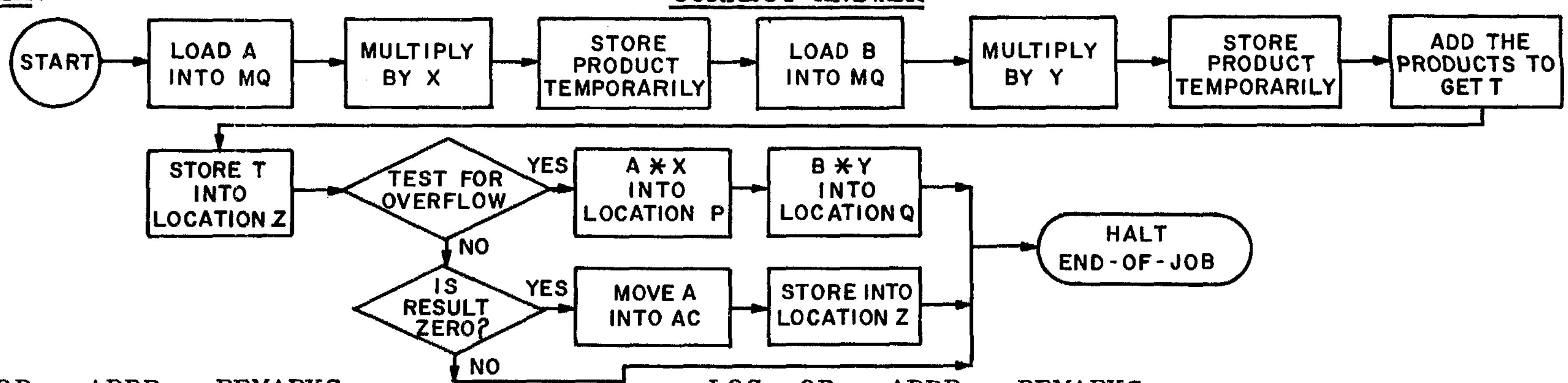
69. Compute: A * X + B * Y = T (fixed point numbers). Store T in location Z. If an overflow occurs, store product of A * X into location P and product of B * Y into location Q and Halt. If result of addition is zero, place the contents of location A into location Z and Halt. If not zero, original computation is OK, so Halt program. Flow-chart this problem before attempting to code it. Use the pseudo op. codes that are necessary.

LOC	OP	ADDRESS	REMARKS	LOC	OP	ADDRESS	REMARKS

PROBLEM:

69.

CORRECT ANSWER



LOC	OP	ADDR	REMARKS
	COUNT	30	Total of 30 instructions
	LDQ	A	Load A into MQ
	MPY	X	Multiply by X
	STO	HOLD	Store temporarily
	LDQ	B	Load B into MQ
	MPY	Y	Multiply by Y
	STO	HOLD 1	Store temp. but it is still in AC also
	ADD	HOLD	Add two products
	STO	Z	Total into loc. Z
	TOV	J1	Go to J1 if overflow
	TZE	J2	Go to J2 if zero
END	HTR	END	If neither overflow nor zero - Halt

This is the end of the main program. The rest is used only if overflow occurs (J1) or if the sum is zero (J2). Allocation of storage locations is necessary to define the symbols used in the address field.

LOC	OP	ADDR	REMARKS
J1	CLA	HOLD	Move HOLD into AC
	STO	P	Store into loc. P
	CLA	HOLD 1	Move HOLD 1 into AC
	STO	Q	Store into loc. Q
	TRA	END	Transfer to loc. END, which halts the program
J2	CLA	A	Move A into AC
	STO	Z	Store into loc. Z
	TRA	END	Transfer to END - Halt
Y	BSS	1	
A	BSS	1	
X	BSS	1	
HOLD	BSS	1	
B	BSS	1	Allocate storage locations to symbols used in the program
HOLD1	BSS	1	
Z	BSS	1	
P	BSS	1	
Q	BSS	1	
	END		Last card

Count up total number of instructions used and place into address field of first card (COUNT).

SYMBOLIC LANGUAGE: In discussing symbolic coding (page 55), we have mentioned that symbols may be used as long as they are defined in the program. It may be worth while to give a definition of the various types of symbols and the names of each type.

ELEMENT - any plain symbol is called an element.
(i.e. AA, BETA, HOLD, END, TOTAL, A1, A2, A3)

TERM - a combination of elements, separated by the multiply (*) or divide (/) signs, are called terms.
(i.e. A * B, X^2 / D, X * Y * Z, SUBTOT / CONST)

EXPRESSION - terms and elements, separated by the add (+) or subtract (-) signs, are called expressions.
(i.e. A * B + Z, A / B - X, A9 + HOLD - X * Y)

USE OF ASTERISKS AND PLUS OR MINUS: The * (asterisk) is the sign used for multiply, but it has a variety of other uses in coding. Some of these may be shown by the following examples:

OP	ADDRESS	MEANING
TRA	* + 3	Transfer to "present location of the instruction" plus 3 (in other words, transf. to 3 instr. past the transfer instruction).
TRA	* - 4	Transfer to "present loc. of the instr." minus 4.
CLA	* + 2	Clear and Add "present loc. of instr." + 2.
STO	* * 2	Store into this location times 2 (or double what is in this location.)
CLA*	* *	Go to loc. 0000 and use the address and tag of 0000 to get the location to put into AC.

It is also possible to use a symbol + or -, as follows:

TRA	HOLD + 2	Transfer to the location containing HOLD, plus two instructions (in other words, the second instruction past HOLD in the program). This is very handy for loops.

EXAMPLES:

1. Suppose that somewhere in a program, you want to be sure that the Divide Check Indicator is Off. This would be accomplished in the following manner:

OP	ADDRESS	REMARKS
DCT		Test Indicator and turn if Off
TRA	* + 1	Trans. to next instruction in sequence

It would not have been right to continue right on with the program after giving the DCT, because under one condition the computer takes the next instruction and under another condition it skips one instruction. A NOP could also have been used instead of the TRA * + 1.

2. On a Halt or Transfer instruction (HTR), we have been using the same symbol in the Loc. and Address fields to make sure that the program comes to a permanent halt. This may be accomplished very simply as follows:

OP	ADDRESS
HTR	*

This accomplishes the same purpose because when the operator pushes the START button, the program goes right back to the Halt instruction and the machine will not restart.

Simply remember that the first * in address field indicates location and the second indicates multiply.

ADDRESS

```
*  *  100<--times 100
^
|______location of itself
```

PROBLEM:

70. Three fixed point numbers are stored in loc. STO 1, STO 2 and STO 3. Find the number which is algebraically the largest and check the sign. If it is minus, place in loc. HOLD and halt. If it is plus, place in loc. STAND and halt.

FLOW CHARTING:

The importance of flow charting can not be overemphasized. A problem such as the one above, should not be coded by a novice, until it is analyzed and flow charted.

Flow charting may be generalized or detailed depending on the desires and needs of the programmer, but usually, the larger and more complex the problem, the more detailed the flow chart should become.

It is extremely important to follow the problem through all possible paths until a logical conclusion is reached. The flow chart is a map and should be followed when coding a problem. How the map is drawn is not important as long as it is understandable to anyone who looks at it. For this reason, it is a good policy to use standardized symbols, such as the ones shown on page ix, at the beginning of the book.

Always flow chart the normal flow of the operation first, leaving the unusual possibilities hanging open. Then go back and run down each possibility in turn until all are covered. Never leave any loose ends open as the computer has no way of deciding what to do if it hits a loose end.

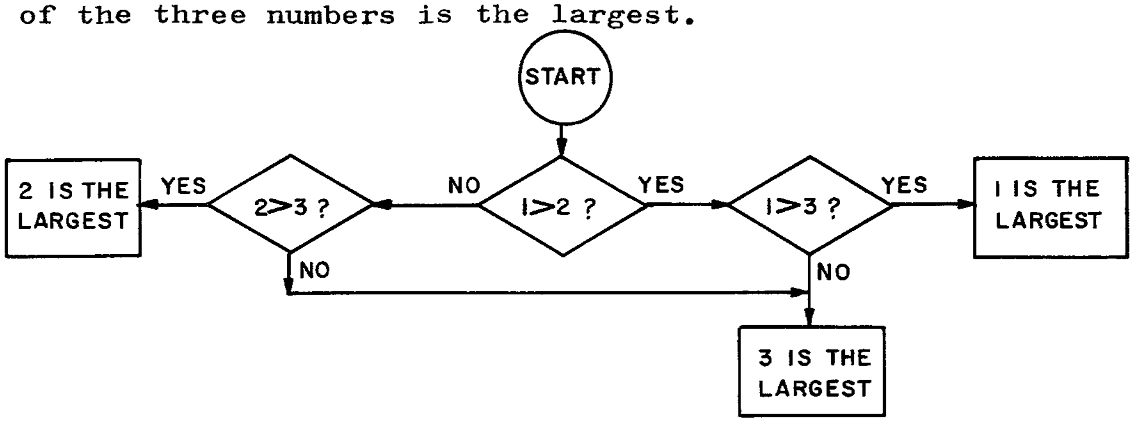
A programmer is responsible for any run he writes even after a considerable length of time has elapsed and it is no longer fresh in his mind. If changes or modifications need to be made at a later date, he can refresh his memory by reviewing the flow chart and he can insert the change more easily by understanding just where in the program it should go.

The program shown above is repeated on the following page and the analysis and development of a flow chart is shown. The problem is to be coded on page 67.

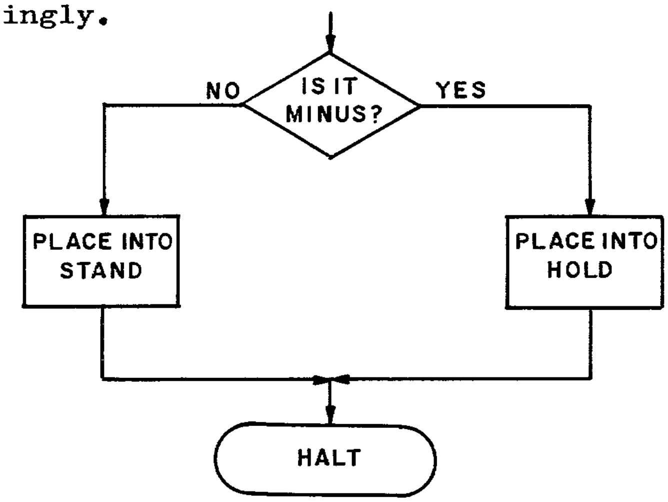
PROBLEM 70 - (Restated)

Three fixed point numbers are stored in loc. STO 1, STO 2 and STO 3. Find the number which is algebraically the largest and check the sign. If it is minus, place in loc. HOLD and halt. If it is plus, place in loc. STAND and halt.

First step in the analysis is to determine which one of the three numbers is the largest.



When the largest number has been located, we must discover whether it is plus or minus, then place into HOLD or STAND accordingly.



The question "Is it minus?" must be asked three times (once for each of the three possibilities in the first half of the flow chart). Once the flow chart has been developed, the process of coding becomes fairly routine. Of course, it is necessary to be familiar with the instructions and what each one can do, to make the job of coding easier.

WORK AREA

PROBLEM 70:

LOC	OP	ADDRESS	REMARKS

CORRECT ANSWER

PROBLEM 70:

```
LOC       OP       ADDRESS    REMARKS

          COUNT    30
          STZ      HOLD
          STZ      STAND      NOTE:  This instruction ex-
                                     plained in Lesson 6
START     CLA      STO 1
          SUB      STO 2      Is 1 > 2?
          TMI      C2vs3      If result -, 1 < 2, so go to
                              cmp 2 vs 3
          CLA      STO 1      If result +, go on to next cmp
          SUB      STO 3      Is 1 > 3?
          TMI      ISLG       If result -, 1 < 3, so 3 is
                              largest, go to ISLG
          CLA      STØ 1      If result +, 1 is largest, move
                              back into AC
          TMI      ISLG 2     If sign -, go to STORE, to place
                              in HOLD
          STO      STAND      If sign +, store in loc. STAND
END       HTR      END        Halt - end of job
-------------------------------------------------------------
C2vs3     CLA      STO 2
          SUB      STO 3      Is 2 > 3?
          TMI      ISLG       If sign -, 2 < 3, so 3 is
                              largest, go to ISLG
          CLA      STO 2      If sign +, 2 is largest, move
                              back into AC
          TMI      * + 2      If 2 is -, go to STORE, to place
                              in HOLD
          TRA      END-1      If 2 is +, go to STORE, to place
                              in STAND
          STO      HOLD       If 2 is +, store in HOLD
          TRA      END        Go to END, to halt program
-------------------------------------------------------------
ISLG      CLA      STO 3
          TMI      ISLG-2     If 3 is -, go to 2nd Instr. be-
                              fore ISLG to store HOLD
          TRA      END-1      If 3 is +, go to END-1 to store
                              STAND
-------------------------------------------------------------
HOLD      BSS      1        }
STAND     BSS      1        }
STØ 1     BSS      1        } Allocate storage locations
STØ 2     BSS      1        }
STØ 3     BSS      1        }
          END
```

LESSON 6

ADDITIONAL INSTRUCTIONS:

INSTRUCTION: DVP (Divide or Proceed) Octal code: +0221
FORMAT: (Type B)

OP CODE	IA	//////	TAG	Y
S,1	11 12-13		18-20 21	35

Description: This instruction is identical to the DVH instruction (page 29), with one extremely important exception. If division can not take place, the "divide-check" indicator turns on as in the DVH instruction, but instead of stopping the computer, it continues to the next instruction in sequence. If this instruction is used, it is usual to check the indicator immediately after the Divide instruction. with a DCT instruction.

INSTRUCTION: RND (Round) Octal code: +0760 0010
FORMAT: (Type E)

OP CODE	//////	TAG	///	OP CODE
S,1	11	18-20	24	35

Description: Used particularly after divide operations. If the product of multiplication is to be rounded, a special instruction (Multiply and Round) is available. If position 1 of the MQ contains a 1, position 35 of the AC is increased by 1. If position 1 of the MQ contains a zero, the AC remains unchanged. In either case, the MQ remains unchanged. AC overflow is possible, so a test for overflow should be made after the Round instruction.

INSTRUCTION: DCT (Divide Check Test) Octal code: +0760 0012
FORMAT: (Type E)

OP CODE	//////	TAG	///	OP CODE
S,1	11	18-20	23	35

Description: If the Indicator is "on", it is turned "off" and the computer takes the next instruction in sequence. If the Indicator is "off", the next instruction is skipped and the computer takes the following instruction.

The Indicator is "on" under two Divide conditions only; (1) if the divisor is zero and (2) if the c(AC) are greater than or equal to the c(Y). The only other way the Indicator may be turned "on" was discussed briefly under Floating Divide on page 46.

Usually all "check" indicators are turned off at the beginning of a program. If a Divide instruction is not carried out, the indicator is turned on and the DCT instruction always turns it off again. The DCT is usually followed by a "Transfer" or "No Operation" instruction (see pages 73 and 77). The next instruction in the normal flow of the program is always the second instruction after the DCT.

Lesson 6, (cont'd)

EXAMPLES:

1. $\frac{A}{B} = T$ Assume fixed point numbers and round the results. If division does not take place, put the dividend into loc. SET. Otherwise put the quotient (T) into loc. GET and the remainder into loc. GOT.

LOC	OP	ADDRESS	REMARKS
	LDQ	A	Move A into MQ
	CLA	ZERO	AC Must be cleared before divide
	LLS	ZERO	To make sign of AC agree with MQ (see Note, page 31)
	DVP	B	Divide by B
	DCT		Divide-Check Test
	(*)TRA	JUMP	If no divide, go to loc. JUMP
	RND		If divide, round result
	STQ	GET	Put quotient into loc. GET
	STO	GOT	Put remainder into loc. GOT
	HTR	*	Halt - end of job
JUMP	STQ	SET	Put dividend into loc. SET
	HTR	*	Halt 2 - end of job

2. $\frac{A}{B} = T$ Assume fixed point numbers and round the result. If division does not take place, turn off indicator and continue program. Otherwise put (T) into loc. SET.

LOC	OP	ADDRESS	REMARKS
	LDQ	A	Move A into MQ
	CLA	ZERO	AC Must be cleared before divide
	LLS	ZERO	To make sign of AC agree with MQ
	DVP	B	Divide by B
	DCT		To turn off indicator if no div.
	(**)NOP		To skip one instruction after DCT
	RND		If divide, round result
	STQ	SET	If divide, T into loc. SET (If no divide, dividend (A) into loc. SET)
	HTR	*	Halt - end of job

* See page 73
** See page 77

WORK AREA

PROBLEMS:

71. $A^3 + \frac{B}{C} = T$ Assume fixed point numbers and round the result. If no division, turn off indicator. Place T into loc. HOLD. Assume all Binary points at position 35 (B35).

LOC	OP	ADDRESS	REMARKS

72. $\frac{A\ B}{D} = T$ Assume fixed point numbers and round result. If no division, place dividend into loc. SET. Otherwise place T into loc. HOLD and the remainder into loc. HOLD + 1.

LOC	OP	ADDRESS	REMARKS

CORRECT ANSWERS

PROBLEMS:

71.

LOC	OP	ADDRESS	REMARKS
	LDQ	A	Move A into MQ
	MPY	A	Multiply by itself (A^2)
	MPY	A	Multiply by itself (A^3)
	STO	TEMP	Store into temporary loc.
	CLA	ZERO	To clear AC prior to Divide
	LDQ	B	Move B into MQ
	LLS	ZERO	To make sign of AC agree with MQ
	DVP	C	Divide by C
	DCT		Turn off indicator if no divide
	NOP		To skip one instruction
	RND		Round result of division
	XCA		To move quotient from MQ to AC
	ADD	TEMP	Add A^3 from temp. loc.
	STO	HOLD	Place T into loc. HOLD
	HTR	*	Halt - end of job

72.

LOC	OP	ADDRESS	REMARKS
	LDQ	A	Move A into MQ
	MPY	B	Multiply by B
	CLA	ZERO	To clear AC prior to Divide
	LLS	ZERO	To make sign of AC agree with MQ
	DVP	D	Divide product of A x B by D
	DCT		Divide check test
	TRA	JUMP	If no divide, go to loc. JUMP
	RND		If divide, round result
	STQ	HOLD	Place T into loc. HOLD
	STO	HOLD + 1	Place remainder into loc. HOLD + 1
	HTR	*	Halt - end of job
JUMP	STQ	SET	Place dividend into loc. SET
	HTR	*	Halt 2 - end of job

INSTRUCTION: STZ (Store Zeros) Octal code: +0600

FORMAT: (Type B)

OP CODE	IA		TAG	Y

S,1 11 12-13 18-20 21 35

DESCRIPTION: The c(Y) are replaced by zeros. The sign at the (Y) address is made a plus. This is a very useful instruction. An example of this was shown in Lesson 5.

INSTRUCTION: LLS (Long Left Shift) Octal code: +0763

FORMAT: (Type B)

OP CODE	IA		TAG	Y

S,1 11 12-13 18-20 21 35

DESCRIPTION: The c(AC), including positions P and Q, and the c(MQ) are treated as one long register. The shifting of bits to the left is determined by the number placed into positions 28-35 of the instruction. This is not to be confused with an address in storage. The sign of the AC is made to agree with the sign of the MQ. If a non-zero bit is shifted into position P, the AC overflow indicator is turned on and any bits shifted past position Q are lost.

INSTRUCTION: LRS (Long Right Shift) Octal code: +0765

FORMAT: (Type B)

OP CODE	IA		TAG	Y

S,1 11 12-13 18-20 21 35

DESCRIPTION: This is identical to the LLS instruction above except that the shift is to the right from the AC to the MQ. In this instruction, the sign of the MQ is made to agree with the sign of the AC and bits shifting past position 35 of the MQ are lost.

INSTRUCTION: TRA (Transfer) Octal code: +0020

FORMAT: (Type B)

OP CODE	IA		TAG	Y

S,1 11 12-13 18-20 21 35

DESCRIPTION: This instruction is used as an unconditional transfer. The computer takes its next instruction from the storage location specified in the (Y) address portion of the instruction.

Lesson 6, (cont'd)

EXAMPLES:

1. We want the sign of A to be the same as the sign of B. Show a partial program to accomplish this.

OP	ADDRESS	REMARKS
LDQ	B	Place B into MQ
CLA	A	Place A into AC
LLS	0	Nothing moves except the sign from B (in MQ) to A (in AC)

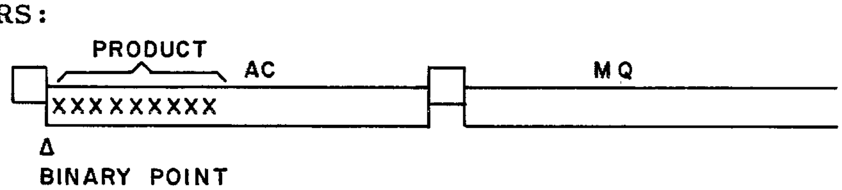
This could also be done with a long right shift:

OP	ADDRESS	REMARKS
LDQ	A	Place A into MQ
CLA	B	Place B into AC
LRS	0	Moves sign from B (in AC) to A (in MQ)

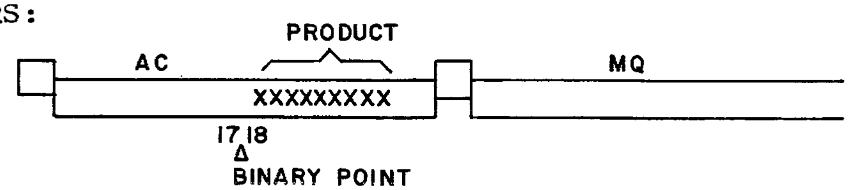
2. We want to multiply two fixed point numbers (A+B) and we want the most significant digits to be to the right of the Binary point, which is to be between positions 17 and 18 (B17). Show a partial program to accomplish this.

OP	ADDRESS	REMARKS
LDQ	A	Place A into MQ
MPY	B	Multiply by B
LRS	17	The Binary point in fixed point numbers is always assumed to be in front of the first position, unless otherwise indicated. Therefore, the product (in AC) must be moved right 17 positions to place it to the right of the desired Binary point position.

Before the LRS:



After the LRS:

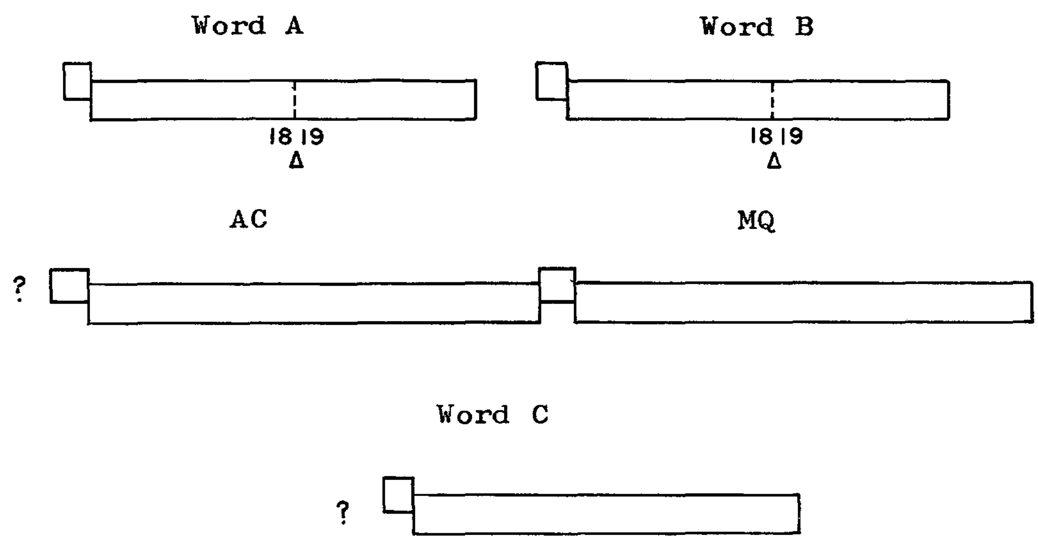


In this case, since all the action was in the AC, the ARS instruction could have been used equally effectively. The only difference is that with the LRS, the sign of the MQ is made to agree with the sign of the AC.

WORK AREA

PROBLEMS

73. Assume two fixed point numbers (A and B) with the Binary point of each fixed between positions 18 and 19 (B 18). Multiply A by B and place the product from the AC into Loc. C. Where will the Binary point be located within location C. Show the location of the Binary point in the AC-MQ before the final Move instruction is given.



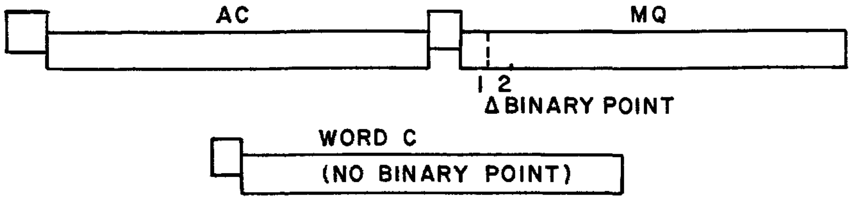
74. In the problem above, shift left so that the Binary point will be between positions 18 and 19 of the AC. Show a partial program to accomplish this.

LOC	OP	ADDRESS	REMARKS
	LDQ	A	Place A into MQ
	MPY	B	Multiply by B
	LLS	$\boxed{}$?	Shift left ? positions
	STO	C	Store from AC into Loc. C
	HTR	*	Halt - end of job

CORRECT ANSWER

PROBLEMS:

73.



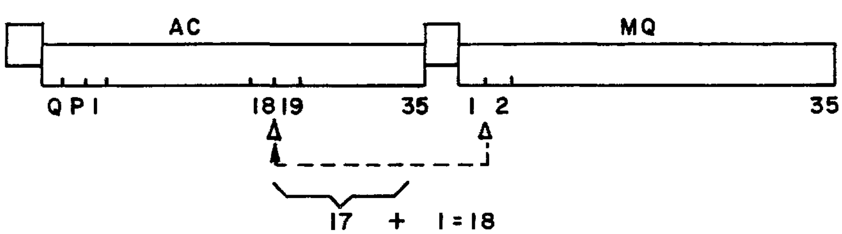
The Binary point will be between positions 1 and 2 of the MQ. As in any problem in multiplication, the product will have as many Binary points as the sum of the digits to the right of the positions in the two numbers being multiplied (18 + 18 = 36). Since the product fills the entire AC and MQ, the 36 rightmost positions will be beyond the Binary point. Therefore, there will be no Binary point in the AC or in Word C.

In decimal arithmetic, if you multiply XX.XXX by X.XXX, the product will have six decimal places. It is no different in Binary multiplication.

74.

LOC	OP	ADDRESS	REMARKS
	LDQ	A	Place A into MQ
	MPY	B	Multiply by B
	LLS	18	Shift left 18 positions
	STO	C	Store from AC into Loc. C
	HTR	*	Halt - end of job

The Binary point was in the MQ, between positions 1 and 2. The long Left Shift 18 would move the point 18 positions to the left, between 18 and 19 of the AC.



INSTRUCTION: CAS (Compare AC with Storage) Octal code: +0340
FORMAT: (Type B)

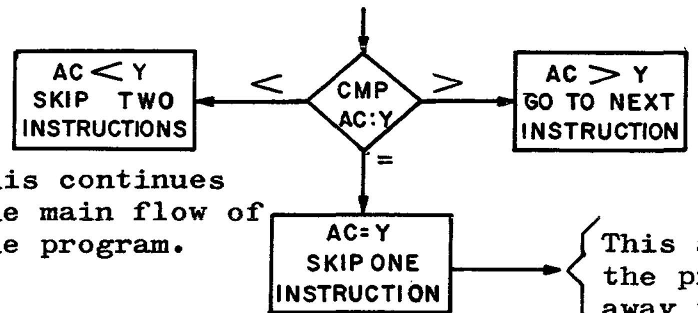
DESCRIPTION: This is the only instruction that allows for a three way branch.

1. If the c(AC) are algebraically greater than the c(Y), the computer takes the next instruction in sequence.

2. If the c(AC) are algebraically equal to the c(Y), the computer skips one instruction.

3. If the c(AC) are algebraically less than the c(Y), the computer skips two instructions.

This could be flow charted as follows:



This should cause the program to branch to a subroutine away from the main flow of the program.

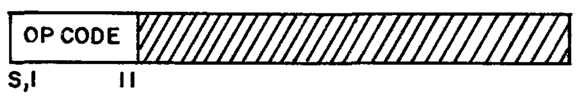
This continues the main flow of the program.

This should also cause the program to branch away from the main flow.

EXAMPLE:

LOC	OP	ADDRESS	REMARKS
100	CAS	350	Compare AC with Loc. 350
101	(If c(AC) > c(350), the computer takes this instr.)		
102	(If c(AC) = c(350), the computer takes this instr.)		
103	(If c(AC) < c(350), the computer takes this instr.)		

INSTRUCTION: NOP (No Operation) Octal code: +0761
FORMAT: (Type D)



DESCRIPTION: This causes no action on the part of the computer. It merely skips this instruction and continues to the next instruction in sequence. One example of the use of NOP was shown on page 70. Another use would be if only a two way decision is needed after the CAS instruction. For example, if both the > and = should take the program to the same place, the instruction after CAS should be NOP.

EXAMPLE: Use locations as in previous examples.

Compute in floating point: $(A + B)C = T$ Compare T with c(SET). If $T > c(\text{SET})$, subtract A - B and store result in GET. Otherwise store T in Loc. GET + 1.

LOC	OP	ADDRESS	REMARKS
	CLA	A	Move A into AC
	FAD	B	Add B to A
	XCA		Move sum from AC to MQ to prepare for multiplication
	FMP	C	Multiply by C
	CAS	SET	Compare AC with c(SET)
	TRA	JUMP	If AC >, take next instr. from loc. JUMP
	NOP		Skip = compare, since both = and < go the same way
	STO	GET + 1	If < or =, store T into Loc. GET + 1
___	HTR	*	Halt - end of job
___	CLA	A	Move A into AC
	FSB	B	Subtract A - B
	STO	GET	Store into Loc. GET
	HTR	*	Halt 2 - end of job

WORK AREA

PROBLEM:

Use locations as in previous problems.

75. Compare A and B. If $A > B$, store A in loc. SET and B in loc. BET. If $A = B$, store A in loc. SET and B in loc. GET. If $A < B$, store A in loc. SET and B in loc. LET.

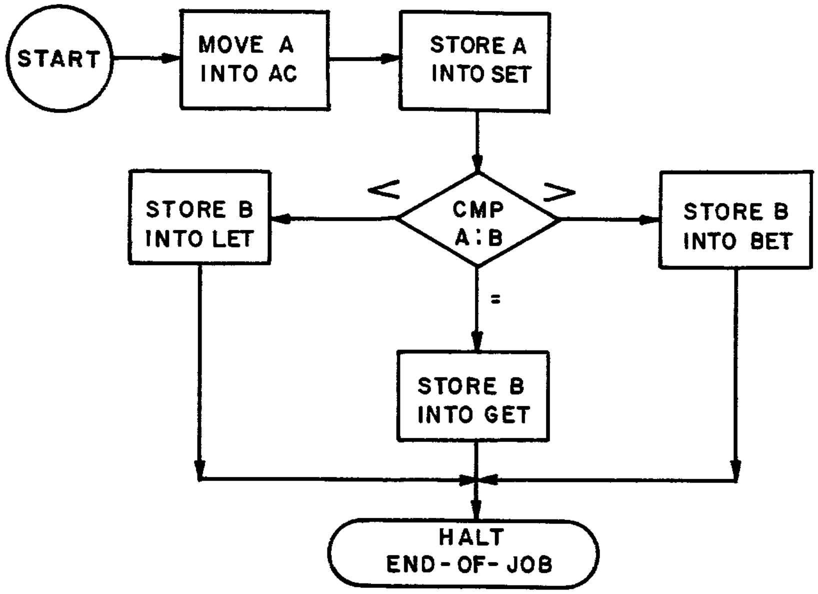
It would be worth while to take a piece of scratch paper and flow chart this problem before attempting to code it. Always use as few instructions as possible.

LOC	OP	ADDRESS	REMARKS

CORRECT ANSWER

PROBLEM:

75.



LOC	OP	ADDRESS	REMARKS
	CLA	A	Move A into AC
	STO	SET	Store A into loc. SET
	CAS	B	Compare A (in AC) with B
	TRA	JUMP	If AC >, go to loc. JUMP
	TRA	JUMP 1	If AC =, go to loc. JUMP 1
	CLA	B	Move B into AC
	STO	LET	AC <, store B in loc. LET
	HTR	*	Halt - end of job
JUMP	CLA	B	Move B into AC
	STO	BET	Store B into loc. BET
	HTR	*	Halt 2
JUMP 1	CLA	B	Move B into AC
	STO	GET	Store B into Loc. GET
	HTR	*	Halt 3

Since A goes into location SET under all three conditions, it is easier to do it at the beginning than to repeat it three times.

LESSON 7

ADDITIONAL PSEUDO OP. CODES:

PSEUDO OP. PZE (Plus Zero)
DESCRIPTION: This pseudo op. code is primarily used to provide constants in desired parts of a register. It describes one word only and places zeros into the Sign and positions 1 and 2 of the word. The address, tag, and decrement may be specified in the normal manner.

Examples:	PZE		Places zeros in Addr., Tag, and Decr.
	PZE	3	Places a 3 into Address
	PZE	0,3	Places a 3 into Tag
	PZE	0,0,3	Places a 3 into Decrement
	PZE	3,3,3	Places a 3 into all three fields

Examples of the use of this pseudo op. code may be found in Lesson 8.

PSEUDO OP. EQU (Equivalent or Equals)
DESCRIPTION: This pseudo op. code is used to define a symbol. It means, "The symbol in the location field is equivalent to whatever is placed in the Address field." It may also be used to equate one symbol to another.

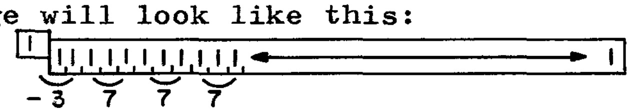
Examples:	Hold	EQU 300	Hold = 300
		CLA ALPHA*HOLD	move ALPHA multiplied by 300 (HOLD) into AC.
	A	EQU 10	A = 10
	X	EQU 3 * 2 + 2	X = 8

PSEUDO OP. OCT (Octal Data)
DESCRIPTION: This pseudo op. code defines a constant as an Octal number. If the number of digits written in the address field is less than 12, the assembly program always right adjusts.

Example: It is desired to place all (Binary) ones into a word called X.

LOC.	OP.	ADDRESS
X	OCT	-377777777777

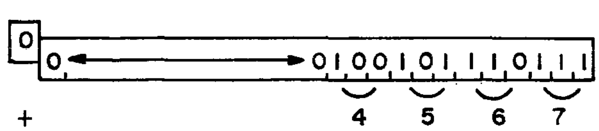
The word in storage will look like this:



If we wish to fill positions 24-35 of word X with 4567_8

LOC	OP	ADDRESS
X	OCT	4567

Result:



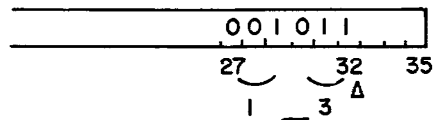
(right adjusted)

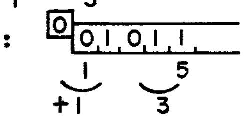
PSEUDO OP: DEC (Decimal Data)
DESCRIPTION: This pseudo op. code defines a constant as a Decimal number. The following three rules must be observed in writing constants in Decimal notation.

1. If floating point, must contain decimal point(.) or (E), but not (B).
2. If fixed point, must contain (B) or be completely free of all three signs (.) (E) (B).
3. If data contains (B), it is fixed point even if (.) or (E) is also used.

Examples: fixed point numbers (refer back to page 37)

LOC	OP	ADDRESS	
X	DEC	11B32	the number 32 designates the Binary point position. $(11_{10} = 13_8)$
X	DEC	11.B32	(same as above)
X	DEC	11B5	Binary point after position 5:
X	DEC	11	If Binary point is not designated, it is presumed to be after position 35.





floating point numbers

X	DEC	3.1415926B8	non-integer (has fraction part) Binary point at position 8
X	DEC	11E	(In floating point, the Binary point is always fixed between positions 8 and 9)
X	DEC	11E10	This means: 11×10^{10}
X	DEC	11E3	This means: 11×10^{3}
X	DEC	11.9	

The only limitations to the number of Decimal numbers that may be written in one line is that they may not extend beyond column 71 and that they must be separated by commas.

FAP recognizes Decimal, Octal, and Hollerith data. Hollerith is used primarily for headings and titles and will not be discussed in detail here. Sufficient to say that Hollerith was one of the developers of electrical contact reading for the 1890 census. His work led to the present day punched card system and his name is associated with certain notation, primarily alphabetic, which is made acceptable to the computer by the use of the BCI or BCD pseudo op. codes or a literal (mentioned on the next page).

USE OF CONSTANTS AND LITERALS: Most programs deal with a certain amount of data and very few programs are written without the use of a number of constants. Constants are usually made a part of the program, while the data, although it may be part of the program, usually is used at the time the program is executed.

A literal is specified by the equal (=) sign located in position 16 of the Coding Sheet (first position of the Address field). Literals are usually not used with pseudo op. codes and they are most easily explained by examples, as follows:

OP	ADDRESS	REMARKS
SUB	= 5	Subtract 5
ADD	= 5	Add 5
	= Ø2777	Octal literal (2777)
	= H JONES	Hollerith literal (JONES)

Constants are usually set up with the pseudo op. codes OCT or DEC. There are times when it is more practical to use a literal. For example, if we were to add 5 to a sequence of numbers, it could be accomplished in either of two ways: (1) when the place to add was reached simply write the instruction: (ADD = 5) and (2) set up a constant with some label such as NOW (NOW DEC 5), and when the place to add was reached, write the instruction (ADD NOW). This second method takes one additional instruction to define the constant (5).

EXAMPLES OF OCTAL AND DECIMAL CONSTANTS:

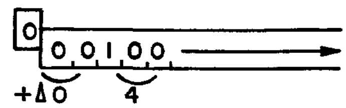
1. Show the Octal representation of the bits in a storage location, of the following: (each Octal no. represents 3 Binary digits except the first, which represents only 2).

a.	DEC	-7	-000000000007
b.	DEC	9	+000000000011
c.	DEC	18	+000000000022
d.	DEC	11B11	+001300000000
e.	DEC	11.B29	+000000001300

EXAMPLES--continued

f. DEC 11B0 — +000000000000
This would shift the entire number out of the register as the Binary point (or unit position of the number) is to be in the zero position.

g. DEC .125B0 — +040000000000
Here the Binary point is zero again, but the fraction part will go to the right of the point.
$.125_{10} = .1_8 = 0.001_2$ In Binary, the word would look like this:



h. OCT -2777 — -000000002777

i. OCT 12345 — +000000012345

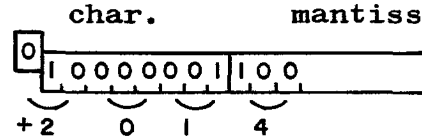
j. Floating Point — (refer to page 41)

1. DEC 7E - 6 — This means: 7×10^{-6}

2. DEC 1. —

char. mantissa

In Binary:



In Octal:

2	0	1	4	0	0

$1_{10} = 1_8 = 001._2$ = Normalized $.1 \times 2^1$
$(200_8 + 1_8 = 201_8)$

3. DEC 5.17E2 — This means: $5.17 = 10^2$

$517_{10} = 1005_8 = 001000000101._2$

$= .100000010100 \times 2^{12}$ (Octal)
$(200_8 + 12_8 = 212_8)$

char. mantissa

2	1	2	4	0	2	4

WORK AREA

PROBLEMS:

76. Show the Octal representation of the following constants: (Also indicate the sign)

a. DEC 35

b. DEC 35.

c. OCT -377777

d. DEC 27B26

e. OCT (blank)

f. DEC -3.5E1

g. DEC .171875

h. DEC 5.498B0

CORRECT ANSWER

PROBLEM 76:

	Sign	Word	
a.	+	0 0 0 0 0 0 0 0 0 0 4 3	Fixed point
b.	+	2 0 6 4 3 0 0 0 0 0 0 0	Floating point because of the decimal point.
c.	−	0 0 0 0 0 0 3 7 7 7 7 7	Right adjusted
d.	+	0 0 0 0 0 0 0 3 3 0 0 0	Binary point fixed at position 26.
e.	+	0 0 0 0 0 0 0 0 0 0 0 0	Blank after the pseudo-op. code means zero.
f.	−	2 0 6 4 3 0 0 0 0 0 0 0	$35_{10} = 43_8 = 100011._2 =$ $.100011 \times 2^6 (200_8 + 6_8 = 206_8)$
g.	+	1 7 6 5 4 0 0 0 0 0 0 0	$.171875_{10} = .130_8 =$ $.001011000_2 = .101100 \times 2^{-2}$ $128_{10} - 2_{10} = 126_{10} = 176_8$
h.	+	3 7 7 0 0 0 0 0 0 0 0 0	$.498_{10} = .377_8$ the whole number is lost, since the Binary point is set at zero

Examples of the use of a constant in a program may be found on pages 100 and 102.

WORK AREA

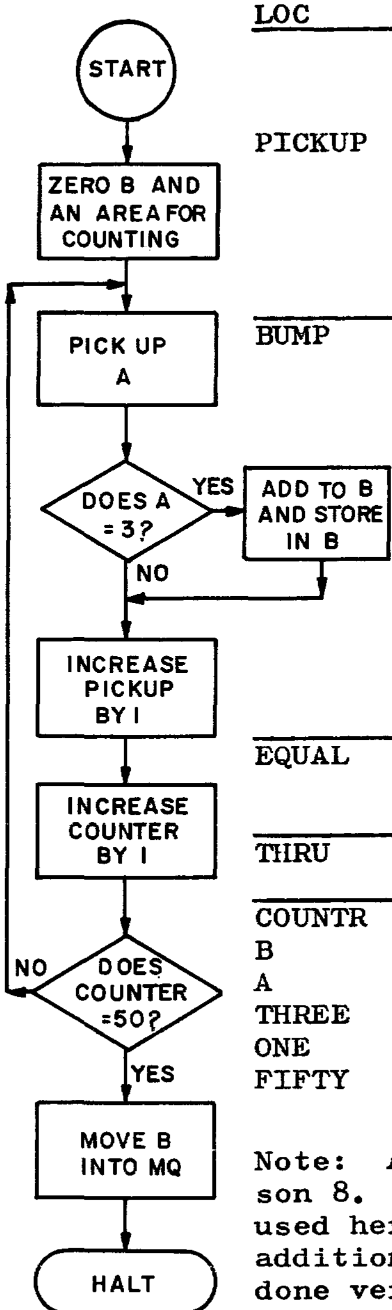
PROBLEM:

77. Fifty floating point numbers are in loc. A through A + 49. All words that are equal to 3, will be added and placed into loc. B. Display B in the MQ when job is done. Flow chart the problem before attempting to code it.

LOC	OP	VARIABLE	REMARKS

CORRECT ANSWER

PROBLEM 77:



LOC	OP	VARIABLE	REMARKS
	COUNT	28	
	STZ	B	
	STZ	COUNTR	
PICKUP	CLA	A	Move A into AC
	CAS	THREE	compare AC with 3
	TRA	*+2	not equal, go to 2nd instr.
	TRA	EQUAL	is =, go to EQUAL routine
BUMP	CLA	PICKUP	Increase A to A + 1, etc. (see Note below)
	ADD	ONE	
	STO	PICKUP	
	CLA	COUNTR	Increase counter by 1
	ADD	ONE	
	STO	COUNTR	
	SUB	FIFTY	To check if counter = 50
	TZE	THRU	If = 50, go to THRU
	TRA	PICKUP	Otherwise, back to PICKUP to go through loop again
EQUAL	FAD	B	Add to B
	STO	B	Store in B
	TRA	BUMP	go back to BUMP rout.
THRU	LDQ	B	Move B to MQ
	HTR	*	Halt - end of job
COUNTR	PZE		
B	PZE		
A	BSS	50	
THREE	DEC	3.0	
ONE	DEC	1	
FIFTY	DEC	50	
	END		

Note: Address modification is covered in Lesson 8. Essentially, the three instructions used here modify the address, using algebraic addition. This type of programming must be done very carefully because of the possibility of making mistakes. All variables involved here are positive. If the instruction at PICKUP had a negative operation code, the desired address modification would not be obtained.

INSTRUCTIONS: The instructions that follow make it possible to store part of the contents of the AC into the corresponding part of a word in storage.

INSTRUCTION: STA (Store address) Octal code: +0621
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

DESCRIPTION: The c(AC) positions 21-35, replace the c(Y) positions 21-35. The contents of Y (S1-20) and the c(AC) remain unchanged.

INSTRUCTION: STD (Store DECREMENT) Octal code: +0622
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

DESCRIPTION: The c(AC) positions 3-17 replace the c(Y) positions 3-17. The contents of Y (S,1,2,18-35) and the c(AC) remain unchanged.

INSTRUCTION: STT (Store TAG) Octal code: +0625
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

DESCRIPTION: The c(AC) positions 18-20 replace the c(Y) positions 18-20. The contents of Y (S, 1-17, 21-35) and the c(AC) remain unchanged).

INSTRUCTION: STP (Store Prefix) Octal code: +0630
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

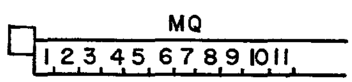
DESCRIPTION: The c(AC) positions P, 1, 2 replace the c(Y) positions S, 1, 2. The contents of Y (3-35) and the c(AC) remain unchanged.

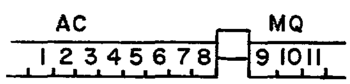
EXAMPLES:

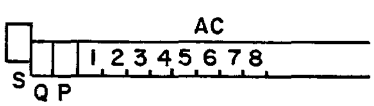
1. Place the TAG of the word presently in location TOTAL, into loc. A1. Place the Address into loc. A2.

LOC	OP	ADDRESS	REMARKS
	COUNT	11	Total of 11 cards used for program.
	STZ	A1	Clear out loc. A1
	STZ	A2	Clear out loc. A2
	CLA	TOTAL	Move TOTAL into AC
	STT	A1	Store TAG into A1
	STA	A2	Store Address into A2
	HTR	*	Halt
TOTAL	BSS	1	Allocate storage space for symbols used
A1	BSS	1	
A2	BSS	1	
	END		End of program

2. The Op. Code of the instruction in loc. HOLD is CLA. Store this Op. Code into loc. AB2. This must be done in a rather devious way since the instructions just covered do not move digits 1-11.







LOC	OP	ADDRESS	REMARKS
	COUNT	10	Total of 10 cards in program
	STZ	AB2	Clear out loc. AB2
	LDQ	HOLD	Move HOLD into MQ
	LLS	8	Shift left to move Op. Code into AC (leaving off the last 3 digits since they are Octal zero).
	ALS	27	AC left shift to put Op. Code in the proper place in AC
	STO	AB2	Store from AC to loc. AB2
	HTR	*	Halt
HOLD	BSS	1	Allocate storage positions
AB2	BSS	1	
	END		End of program.

WORK AREA

PROBLEM:

78. A Type B instruction is in location HOLD. Move the Op. Code into loc. B1, the TAG into loc. B2 and the Address into loc. B3.

LOC	OP	ADDRESS	REMARKS

CORRECT ANSWER

PROBLEM 78:

LOC	OP	ADDRESS	REMARKS
	COUNT	17	
	STZ	B1	
	STZ	B2	
	STZ	B3	
	LDQ	HOLD	Move HOLD into MQ
	LLS	11	Shift left 11 places to move Op. Code into AC (since we do not know what the last Octal no. of Op. Code is, we must move it all).
	ALS	24	AC left shift to move Op. Code into proper position in the AC (11 + 24 = 35)
	STO	B1	Store Op. Code into B1
	CLA	HOLD	Move HOLD into AC
	STT	B2	Move TAG into B2
	STA	B3	Move Address into B3
	HTR	*	Halt - end of job
HOLD	BSS	1	
B1	BSS	1	
B2	BSS	1	
B3	BSS	1	
	END		

LESSON 8

USE OF INDEX REGISTERS: The primary use of Index Registers is for purposes of counting and address modification. The 7090 contains three Index Registers, commonly referred to as XR1, XR2, and XR4 (please refer back to page 17, par. 3 and page 21, definition of TAG). There is no provision for a sign, so the contents of an Index Register are always considered to be positive.

PRESUMPTIVE AND EFFECTIVE ADDRESSES: When an address is to be modified by using an Index Register, a TAG is specified. In this case, the address of the instruction is not the true address, but is called the presumptive address. The true address (called the effective address) is the presumptive address minus the contents of the specified Index Register.

EXAMPLE: CLA 200,2 This tells the computer to place the contents of location 200 minus the contents of XR2 into the AC. If XR2 contained a 10, the effective instruction would be:

CLA 190

In this way, the address of the instruction has been modified.

ADDRESS MODIFICATION: There are many reasons why an address should be modified in a program. For example, if we want to add a fixed amount to a large number of sequential addresses. This could be accomplished by a large series of ADD instructions, but it would be extremely wasteful of storage. It is much more advantageous to give the ADD instruction once, modified by an Index Register which will be incremented or decremented in a loop which will continue until all of the desired addresses are modified.

A more detailed example of this process involves instructions which are found on pages 94 and 95. The examples on pages 97 and 98 attempt to show the process of address modification and counting in greater detail.

Two, and even three, Index Registers may be used, depending on the complexity of the problem. Pages 103 and 104 go into more detail on the use of multiple Index Registers.

It is extremely important to understand Indexing and the reasoning behind the use of Index Registers because they are used very extensively in programming. For this reason it is recommended that Lesson 8 be studied and restudied until all points have been understood.

INSTRUCTIONS: The following instructions are used to load and store the contents of index registers. The TAG specifies the Index Register (or Registers) to be affected (see page 21 for Binary codes for Index Registers.

INSTRUCTION: LXA (load Index from Address) Octal code: +0534
FORMAT: (Type B)

DESCRIPTION: The address part of the c(Y) (positions 21-35) replaces the number in the specified Index Register (XR). The c(Y) are unchanged.

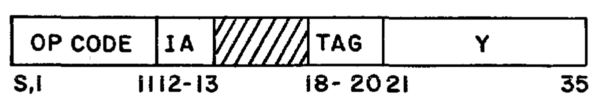
INSTRUCTION: LXD (Load Index from Decrement) Octal code: -0534.
FORMAT: (Type B)

DESCRIPTION: The decrement part of the c(Y) (positions 3-17) replaces the number in the specified Index Register (XR). The c(Y) are unchanged.

INSTRUCTION: AXT (Address to Index True) Octal code: +0774
FORMAT: (Type B)

DESCRIPTION: This is identical to the LXA instruction above except that instead of the contents of Y moving into the Index Register, whatever is in Y will move into it. See examples on page 96.

INSTRUCTION: TSX (Transfer and Set Index) Octal code: +0074
FORMAT: (Type B)



DESCRIPTION: This instruction places the 2's complement of the instruction counter contents into the Index Register specified by the TAG.

```
EXAMPLE:  10010110
          01101001  1's compl. (simply reverse)
                 1  2's compl. (add 1)
          --------
          01101010
```

INSTRUCTIONS: The following instructions are used to test or modify (or both test and modify) the contents of the Index Register specified by the TAG.

INSTRUCTION: TIX (Transfer on Index) Octal code: +2000
FORMAT: (Type A)

OP	DECREMENT	TAG	Y
S,1,2 3	17	18-20 21	35

DESCRIPTION: If the contents of the Index Register, specified by the TAG, are greater than the Decrement, the number in the Index Register is reduced by the Decrement and the next instruction is taken from the location specified by Y. Otherwise, the TAG remains unchanged and the computer goes on to the next instruction in sequence.

INSTRUCTION: TXI (Transfer with Index Incremented)
Octal code: +1000
FORMAT: (Type A)

OP	DECREMENT	TAG	Y
S,1,2 3	17	18-20 21	35

DESCRIPTION: The decrement portion of the instruction (pos. 3-17) is added to the contents of the Index Register specified by the TAG. The resulting sum moves into the Index Register and the computer then takes its next instruction from the location specified by Y.

INSTRUCTION: TXL (Transfer on Index Low or Equal)
Octal code: -3000
FORMAT: (Type A)

OP	DECREMENT	TAG	Y
S,1,2 3	17	18-20 21	35

DESCRIPTION: If the contents of the Index Register, specified by the TAG, are less than or equal to the Decrement, the next instruction is taken from the location specified by Y. Otherwise, the computer takes the next instruction in sequence.

INSTRUCTION: TXH (Transfer on Index High) Octal code: +3000
FORMAT: (Type A)

OP	DECREMENT	TAG	Y
S,1,2 3	17	18-20 21	35

DESCRIPTION: If the contents of the Index Register, specified by the TAG, are greater than the Decrement, the next instruction is taken from the location specified by Y. Otherwise, the computer takes the next instruction in sequence.

Lesson 8, (cont'd)

EXAMPLES:

	LOC	OP	VARIABLE FIELD (Address, Tag, Decrement)	REMARKS
1.		LXA	HOLD, 2	15 is loaded into XR2 (Defined by the PZE below)
		-		
		-		
	HOLD	PZE	15	
2.		LXD	J1, 1	6 is loaded into XR1 (The PZE defines 10 for Address, 3 for Tag and 6 for Decrement.)
		-		
		-		
	J1	PZE	10, 3, 6	
3.		TSX	HOLD, 4	Computer transfers to loc HOLD and sets XR4 equal to minus the loc of the TSX. Thus a transfer to 1, 4 at HOLD will return the computer to the location of the TSX plus 1.
		HTR	*	
		-		
		-		
	HOLD	TRA	1, 4	
4.		AXT	200, 1	This means: move the digits 200_{10} into XR1. Not the contents of loc. 200, but the actual numbers (200) move into XR1.
5.		TIX	Start, 2, 5	This means: if c(XR2) are greater than the Decrement of 5, the number in XR2 is reduced by 5, and control is transferred to location START. Otherwise, on to the next instruction.
6.		TXI	AB2, 2, 7	This means: add Decrement of 7 to the c(XR2) and transfer control to loc AB2.
7.		TXL	HOLD, 4, 13	This means: if c(XR4) are less than or equal to the Decrement of 13, transfer control to location HOLD. Otherwise, on to the next instruction.
8.		TXH	HOLD, 1, 3	This means: if c(XR1) are greater than the Decrement of 3 transfer control to location HOLD. Otherwise, on to the next instruction.

EXAMPLE:

PROBLEM: A block of 20 numbers are stored consecutively in storage, beginning in location TABLE. Store this block of numbers in the same order in storage beginning with location XYZ. Show a partial program to accomplish this action.

LOC	OP	VARIABLE FIELD	REMARKS
	LXA	STORE, 2	Move 20 to XR2
START	CLA	Table + 20, 2	Move 1,2,3---through 20 to AC
	STØ	XYZ + 20, 2	Move 1 to loc. XYZ, 2 to XYZ + 1, etc.
	TIX	START, 2, 1	If c(XR2) is greater than 1, subtract 1 and go to START
	HTR	*	Halt - end of job
STORE	PZE	20	Set up one word containing 20 in address field
TABLE	BSS	20	Allocate 20 storage positions to TABLE
XYZ	BSS	20	Allocate 20 storage positions to XYZ

Let us examine what has been accomplished by this program:

(1) Since there are 20 numbers, 20 is loaded into an Index Register.

(2) The CLA instruction moves the first of the 20 numbers into the AC. (It says, "move TABLE + 20 - XR2 (which contains 20))." Therefore the first of the 20 numbers in loc. TABLE goes into the AC.

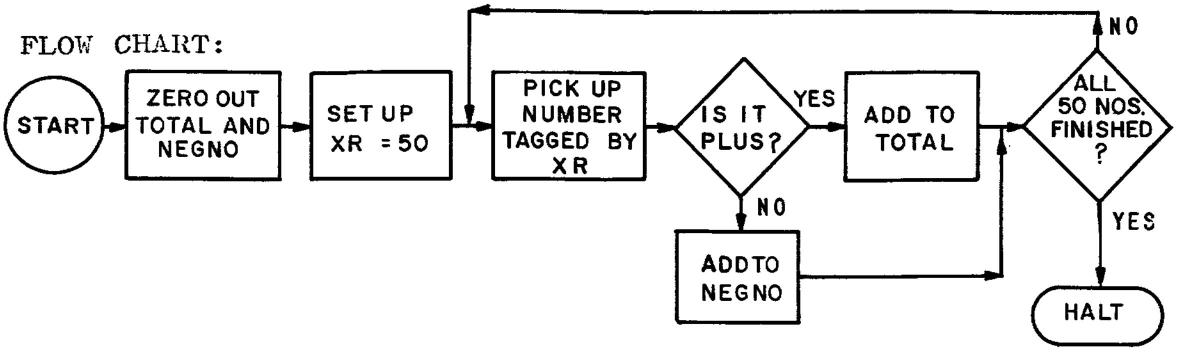
(3) The STO instruction works the same way, XYZ + 20 - 20 = XYZ.

(4) The next step is to compare the contents of XR2 with the Decrement of 1. The number in XR2 is reduced by the Decrement of 1, so XR2 now stands at 19, the program goes back to START and goes through the loop again, moving the second number since now we have TABLE + 20 - 19. Again, 1 drops from the Index Register and this continues until XR2 finally stands at 1, at which time all 20 numbers have been moved and since XR2 is equal to the Decrement of 1, the program goes on to the HALT instruction and the job is done.

Please review this example until it is thoroughly understood. Read over the TIX instruction on page 95, as this problem demonstrates its use very effectively.

EXAMPLE: Given 50 floating point numbers stored in DATA through DATA + 49. Sum all positive numbers and store in location TOTAL. Sum all negative numbers and store in location NEGNO. Show a partial program to accomplish this action.

FLOW CHART:



LOC	OP	VARIABLE	REMARKS
	STZ	TOTAL	Zero out position TOTAL
	STZ	NEGNO	Zero out position on NEGNO
	AXT	50, 2	Set up 50 in XR2
LOOP	CLA	DATA + 50, 2	Move DATA + 50-50 into AC This moves in first word (DATA)
	TMI	NEG	If number is negative, jump to loc. NEG.
	FAD	TOTAL	If not -, must be +, add to whatever is in TOTAL
	STO	TOTAL	Move from AC, back to storage loc. TOTAL
TEST	TIX	LOOP, 2, 1	Has XR2 dropped to 1? If not, take off 1 and go back to LOOP (the second time through LOOP, move DATA + 50-49, or DATA + 1 into AC.
	HTR	*	HALT - When the program has gone through LOOP 50 times, XR2 will = 1 and that is the finish of the job.
NEG	FAD	NEGNO	If no. was - in first instr. past LOOP, program comes here and add to whatever was in NEGNO.
	STO	NEGNO	Move from AC to storage loc. NEGNO
	TRA	TEST	Go back to test XR to see if finished.
TOTAL	BSS	1	Allocate 1 position to TOTAL
NEGNO	BSS	1	Allocate 1 position to NEGNO

Notice that AXT was used to set up 50 in Index Register, rather than LXA. This saves one instruction as we don't need to set up a constant with the PZE instruction.

WORK AREA

PROBLEM:

79. Twenty fixed point numbers are stored consecutively, starting in location HOLD. Twenty other fixed point numbers are stored consecutively, starting in location STAND. Place HOLD - STAND into location TOTAL, HOLD + 1 - STAND + 1 into loc. TOTAL + 1, HOLD + 2 - STAND + 2 into loc. TOTAL + 2, etc. If an overflow occurs, replace that difference by one bits in all positions of the word. Show a partial program to accomplish this action.

LOC	OP	VARIABLE FIELD	REMARKS

CORRECT ANSWER

PROBLEM 79:

LOC	OP	VARIABLE FIELD	REMARKS
	TOV	* + 1	Make sure overflow indicator is off.
	AXT	20, 1	Place 20 into XR1
START	CLA	HOLD + 20, 1	Move HOLD, HOLD + 1, HOLD + 2, etc. to AC
	SUB	STAND +20, 1	Subtract HOLD - STAND etc.
	TOV	GO	If overflow, jump to GO
	STO	TOTAL +20, 1	Differences into TOTAL, TOTAL + 1, etc (or all 1's)
	TIX	START, 1, 1	If c(XR1) is greater than 1, go to START
	HTR	*	Otherwise HALT - end of job.
GO	CLA	OCTAL	Replace overflow difference with all ones.
	TRA	START + 3	Go to third instruction past START
OCTAL	OCT	-377777777777	Set up Octal constant to produce all ones.
HOLD	BSS	20	
STAND	BSS	20	Allocate storage locations
TOTAL	BSS	20	

This works exactly the same as the example shown on page 97. One or two additional things were thrown in, but these should not have obscured the basic problem of moving a series of numbers from one place in storage to another place in storage.

WORK AREA

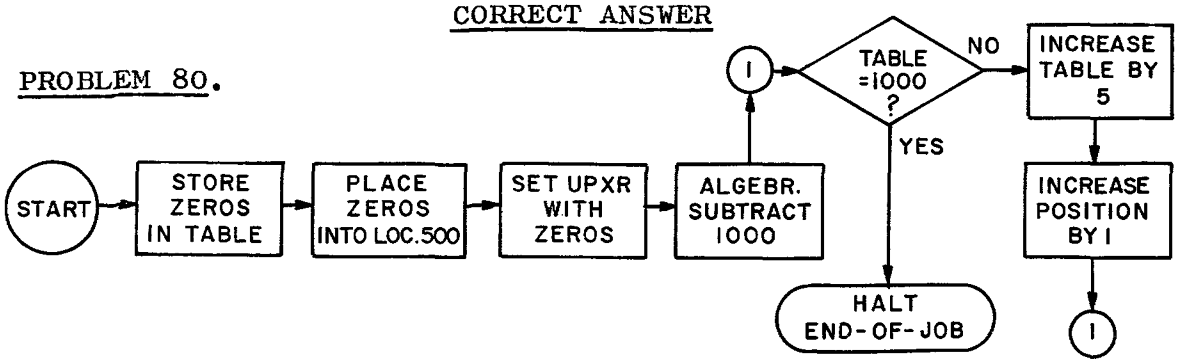
PROBLEM:

80. Generate a table of numbers from 0 through 1000, in increments of 5. Store the first number in location 500. Show a partial program to accomplish this action. Flow chart the problem on scratch paper before starting to code.

LOC	OP	VARIABLE	REMARKS

CORRECT ANSWER

PROBLEM 80.



LOC	OP	VARIABLE	REMARKS
	STZ	TABLE	Store zeros into TABLE
	CLA	TABLE	Move TABLE into AC
	STO	500	Store TABLE (containing zeros) into loc. 500
	AXT	0, 1	Set up XR1 with zeros
LOOP	SUB	= 1000	Algebraic subtract a storage position containing 1000
	TZE	HALT	If TABLE = 1000, go to HALT
	CLA	TABLE	If not 1000, bump
	ADD	FIVE	table by
	STO	TABLE	five
	TXI	*+1, 1, -1	Bump XR1 by -1 and go to next instruction (*+1)
	STO	500, 1	Store this number away
	TRA	LOOP	Go back through the LOOP again
HALT	HTR	*	Halt - end of job
FIVE	DEC	5	Set up constant of 5 to increment the numbers.
TABLE	DEC	3.9	Set up a number at random in location called TABLE. This will be zeroed out by first STZ instruction.

In this case by using TXI and using -1, we are actually increasing the address of the STORE by 1. Since we started with zero, the program continues through the LOOP until 1000 is reached, at which time it transfers to HALT.

Note the connector 1 in the flow chart above. Connectors are used in flow charting instead of crossing over lines. This is particularly necessary in large flow charts that cover more than one page or where there are a number of returns to earlier parts of the flow chart.

USE OF TWO OR THREE INDEX REGISTERS: The problem to be solved may be complex enough to require more than one Index Register. The computer allows the programmer the capability of using two, or even three, Index Registers at the same time to do different jobs.

For example, if we wished to move 50 sequential words located at A through A + 49, to location B through B + 49 and we also wanted to move every tenth word to C through C + 4, this could be accomplished by setting up two Index Registers (one to make 50 moves and the other to pick up every tenth move).

In the same manner, if the problem calls for three different types of action at the same time, three Index Registers may be used to control the action.

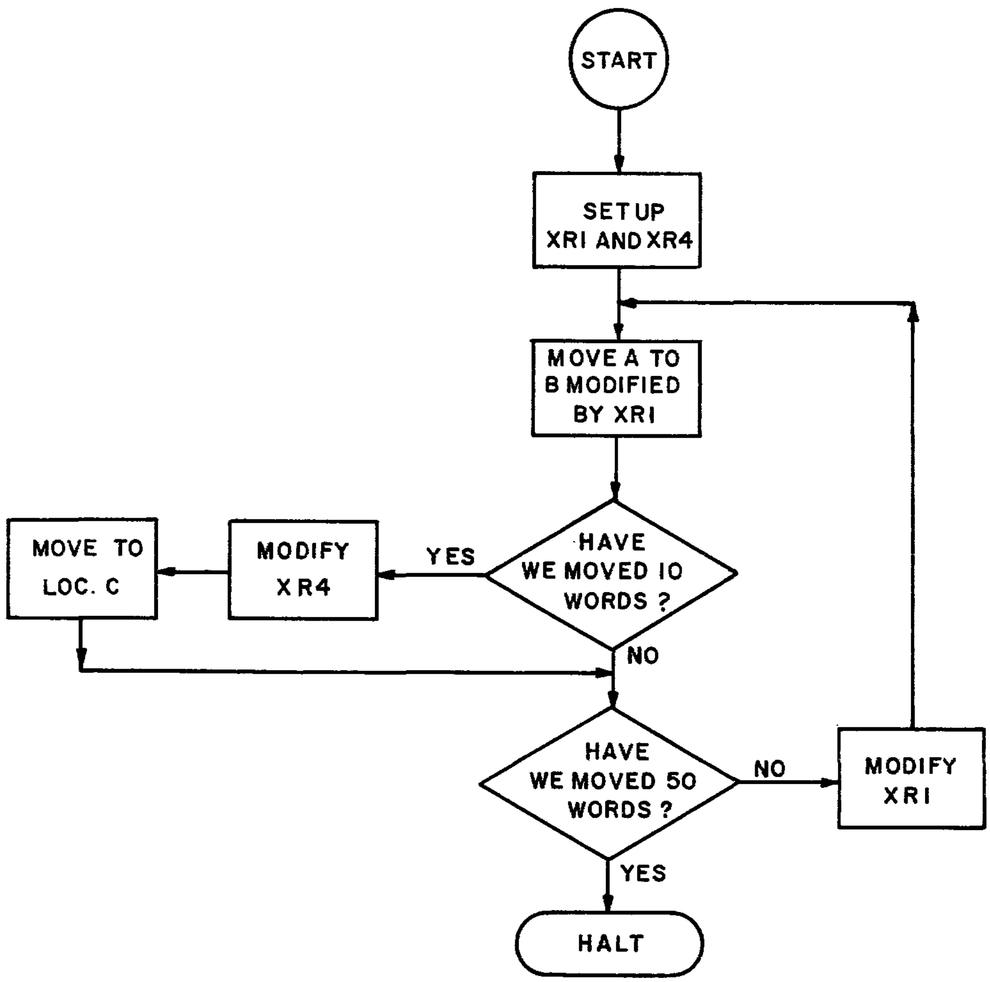
Generally, Index Registers are used in the execution of a LOOP, where the program goes around and around the LOOP until that part of the job is finished. In using more than one Index Register, great care must be taken that the two (or three) loops do not interfere with each other and that each one does its own job.

On the following pages, the simple example given above will be flow charted and programmed as an example of the use of two Index Registers. Follow it through carefully before attempting the problem on page 106.

EXAMPLE:

Move 50 sequential words located at A through A + 49 to location B through B + 49. Also move every tenth word to location C through C + 4. Use XR1 to make the 50 word move and XR4 to pick up every tenth word.

FLOW CHART



PROGRAM

LOC	OP	VARIABLE	REMARKS
	COUNT	19	
	AXT	50, 1	Set up XR's
	AXT	5, 4	
LOOP	CLA	A + 50, 1	Move A into AC
	STO	B + 50, 1	Store in loc. B
CMPXR4	TXL	MXR4, 1, 40	Moved 10 words?
	TIX	LOOP, 1, 1	No, moved 50 words?
	HTR	*	Through - Halt
MXR4	STO	C + 5, 4	Put 10th word into C
	CLA	CNST 1	Put 40 into AC
	SUB	= 10B17	Subtract 10
	STD	CMPXR4	Changes 40 to 30, etc. (by storing Decrement of AC to replace 40, etc.)
	STO	CNST 1	Save for next subtract
	TXI	CMPXR4 + 1, 4, -1	Decrement XR4
CNST 1	DEC	40B17	Constant for XR4 in Decr.
A	BSS	50	Allocate storage to A, B, and C
B	BSS	50	
C	BSS	5	
	END		

WORK AREA

PROBLEM:

81. Expand the problem on page 104 as follows:

Move 50 sequential words located at A through A + 49 to location B through B + 49. Also move every fifth word to location C through C + 9 and every tenth word to location D through D + 4. Use XR1 to make the 50 word move, XR2 to pick up every fifth word and XR4 to pick up every tenth word.

FLOW CHART

WORK AREA

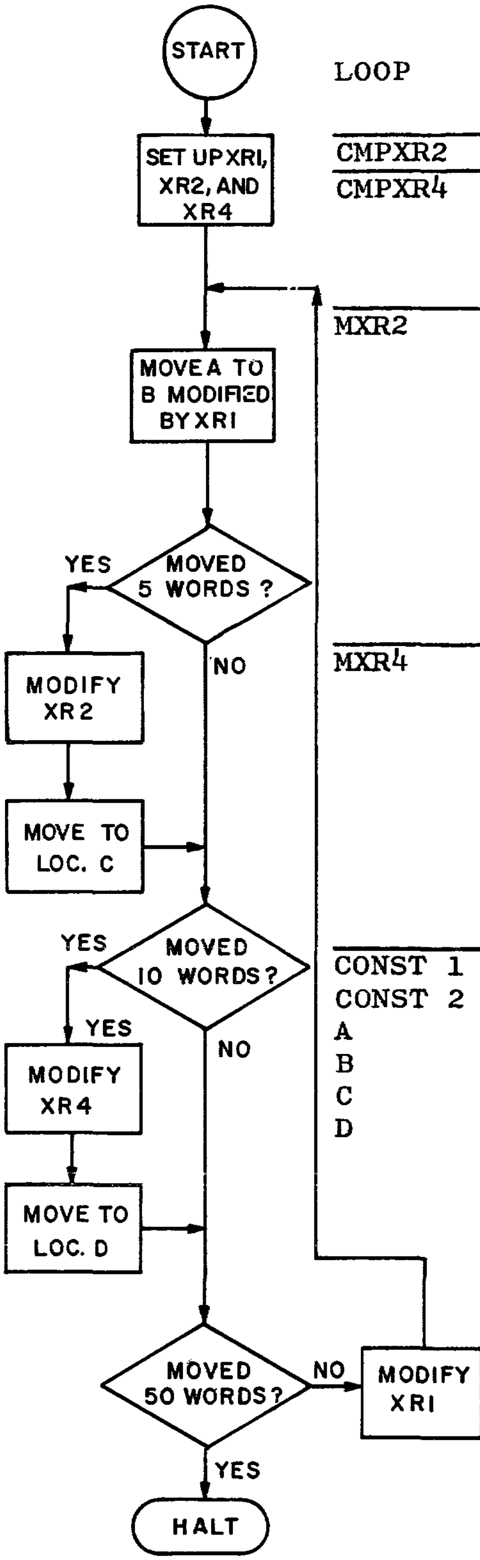
PROGRAM:

LOC	OP	VARIABLE	REMARKS

CORRECT ANSWER

PROBLEM 81:

FLOW CHART



LOC	OP	VARIABLE	REMARKS
	COUNT	31	
	AXT	50, 1	Set up XR's
	AXT	10, 2	
	AXT	5, 4	
LOOP	CLA	A + 50, 1	Move A to AC
	STO	B + 50, 1	Store in loc. B
CMPXR2	TXL	MXR2,1,45	moved 5 words?
CMPXR4	TXL	MXR4,1,40	no. moved 10 words?
	TIX	LOOP,1,1	no. 50 words, through?
	HTR	*	Halt - finished
MXR2	STO	C + 10,2	Put 5th word in C
	XCA		Save AC in MQ
	CLA	CONST 1	Put 45 into AC
	SUB	=5B17	Subtract 5
	STD	CMPXR2	This changes 45 to 40, etc.
	STO	CONST 1	Save for next sub.
	XCA		Restore AC from MQ
	TXI	CMPXR4, 2, -1	Decrement XR2
MXR4	STO	D + 5, 4	Put 10th word into D
	CLA	CONST 2	Put 40 into AC
	SUB	=10B17	Subtract 10
	STD	CMPXR4	This changes 40 to 30, etc.
	STO	CONST 2	Save for next sub.
	TXI	CMPXR4 +1, 4, -1	Decrement XR4
CONST 1	DEC	45B17	Constant for XR2
CONST 2	DEC	40B17	Constant for XR4
A	BSS	50	Allocate storage to A, B, C and D
B	BSS	50	
C	BSS	10	
D	BSS	5	
	END		

LESSON 9

QUICK REFERENCE

INSTRUCTIONS AND THEIR MEANINGS

1. MISCELLANEOUS INSTRUCTIONS

Refer to Page No.	Instruction	Meaning
47	XCA (+0131)	EXCHANGE AC AND MQ - Reverses two fields
30	HTR (+0000)	HALT AND TRANSFER - Halts program, if restart, goes to Y.
77	NOP (+0761)	NO OPERATION - Program continues with next instruction.

2. FIXED POINT ARITHMETIC INSTRUCTIONS

Refer to Page No.	Instruction	Meaning
27	ADD (+0400)	ADD - Add Y to AC
29	SUB (+0402)	SUBTRACT - Subtract Y from AC
29	MPY (+0200)	MULTIPLY - Multiply Y by MQ, product in AC (and MQ if needed)
69	RND (+0760-0010)	ROUND - Increase AC by Binary 1 if posit. 1 of MQ contains 1.
29	DVH (+0220)	DIVIDE OR HALT - AC and MQ are dividend, Y is Divisor, Quotient in MQ, remainder in AC. If can't divide, Halt.
69	DVP (+0221)	DIVIDE OR PROCEED - As above, except that if can't divide, continue with program with Div. check light on.
69	DCT (+0760-0012)	DIVIDE CHECK TEST - If indicator on, takes next instruction. If indicator off, skips one instr.

3. FLOATING POINT ARITHMETIC INSTRUCTIONS

Refer to Page No.	Instruction	Meaning
46	FAD (+0300)	FLOATING ADD - Add Y to AC
46	FSB (+0302)	FLOATING SUBTRACT - Subtract Y from AC
46	FMP (+0260)	FLOATING MULTIPLY - Multiply Y by MQ
46	FDH (+0240)	FLOATING DIVIDE OR HALT - AC divided by Y. Quotient in MQ, remainder in AC. If can't divide, HALT.

4. SHIFTING INSTRUCTIONS

Refer to Page No.	Instruction	Meaning
47	ALS (+0767)	AC LEFT SHIFT - The AC shift left no. position in Y 28-35.
47	ARS (+0071)	AC RIGHT SHIFT - As above, only shift to the right.
73	LLS (+0763)	LONG LEFT SHIFT - AC and MQ as one register, shifted left, no. places specified in Y 28-35.
73	LRS (+0765)	LONG RIGHT SHIFT - As above, only shift to the right.

Refer to
Page No.

5. STORE AND LOAD INSTRUCTIONS:

27	CLA	(+0500)	CLEAR AND ADD - Move Y into AC
29	STO	(+0601)	STORE - Move AC into Y
30	LDQ	(+0560)	LOAD MQ REGISTER - Move Y into MQ
30	STQ	(-0600)	STORE FROM MQ REGISTER - Move MQ into Y
73	STZ	(+0600)	STORE ZEROS - Move zeros into Y, sign to +
89	STA	(+0621)	STORE ADDRESS-From AC_{21-35} to Y_{21-35}
89	STD	(+0622)	STORE DECREMENT-From AC_{3-17} to Y_{3-17}
89	STT	(+0625)	STORE TAG - From AC_{18-20} to Y_{18-20}
89	STP	(+0630)	STORE PREFIX-From $AC_{S,1,2}$ to $Y_{S,1,2}$

6. TRANSFER INSTRUCTIONS (NO INDEX):

73	TRA	(+0020)	TRANSFER - Trans. to instr. spec. by Y
31	TZE	(+0100)	TRANSFER ON ZERO - If AC = Zero transfer to Y Otherwise on to next instr.
31	TOV	(+0140)	TRANSFER ON OVERFLOW-If AC overflow indicator on, transfer to Y, otherwise on to next instruction.
47	TPL	(+0120)	TRANSFER ON PLUS - If sign of AC +, transfer to Y, otherwise to next instr.
47	TMI	(-0120)	TRANSFER ON MINUS-If sign of AC-, trans. to Y, otherwise to next instr.
77	CAS	(+0340)	COMPARE AC WITH Y-If c(AC) > c(Y) go to next instr. If =, skip one instr. If <, skip two instr.
52	NZT	(-0520)	STORAGE NOT ZERO TEST - If c(Y) are not 0, skip instr. If c(Y) are 0, on to next instr.
53	ZET	(+0520)	STORAGE ZERO TEST - This is the opposite of NZT instr.

7. TRANSFER INSTRUCTIONS (INDEX):

95	TIX	(+2000)	TRANSFER ON INDEX-If c(XR) > Decr., XR reduced by Decr. and on to Y. Otherwise on to next instr.
95	TXI	(+1000)	TRANS. WITH INDEX INCREMENTED - Adds Decr. to XR and on to Y
95	TXL	(-3000)	TRANS. ON INDEX LOW OR EQUAL-If c(XR) < or = Decr. go to Y, otherwise on to next instr.
95	TXH	(+3000)	TRANS. ON INDEX HIGH - If c(XR) > Decrement, go to Y, otherwise on to next instruction.
95	TSX	(+0074)	TRANS. AND SET INDEX - Places 2's compl. of instruction CTR into XR, next instruction from loc. Y.

Refer to Page No.		
	8. INDEXING INSTRUCTIONS:	
94	LXA (+0534)	LOAD INDEX FROM ADDRESS - $c(Y)_{21-35}$ Moves into specified XR.
94	LXD (-0534)	LOAD INDEX FROM DECREMENT - $c(Y)_{3-17}$ Moves into specified XR.
94	AXT (+0774)	ADDRESSEE TO INDEX TRUE - Positions_{21-35} of this instruction, moves into specified XR.
	9. PSEUDO OPERATION CODES:	
60	COUNT	- COUNT - First card of symbolic deck. Gives number of cards in program.
60	END	- END - Last card of symbolic deck.
60	BSS	- BLOCK STARTED BY SYMBOL - Allocates block of storage. First loc. of block tagged by a symbol.
81	PZE	- PLUS ZERO - Assigns one word and puts zeros into S, 1, 2. Can specify address, tag, decrement.
81	EQU	- EQUIVALENT - Used to define a symbol.
81	OCT	- OCTAL DATA - Data generating, series of variables.
82	DEC	- DECIMAL DATA - Data generating, decimal integers, fixed pt. or floating pt.

REVIEW AND SELF-TEST

The following pages touch on those areas with which the student should now be familiar. Page references will be given with the correct answers and it is suggested that the reference be checked on all questions answered incorrectly.

Consider this to be a self-administered, open book quiz. There will be 25 questions covering the first eight lessons and a problem to be flow-charted and coded. Answer all the questions and complete the coding before checking the correct answers. The correct answers to the 25 questions may be found on page 116 and the correct solution to the problem on pages 117 and 118.

Subtract two points for each question missed (if half a question is missed, subtract one point) and subtract one point for each coding error from a total possible of 100. Total score on the two parts should be 70 or over and three hours is maximum time for the entire quiz.

The quick reference of the 43 instructions and 7 pseudo op. codes at the beginning of this lesson, is to aid the student in the quick recall of instructions.

PROBLEMS

83. Convert 759_{10} to Binary notation.

84. Add: 001 101 110 011
+ 000 100 011 001

85. a. Which is considered greater by the computer?
circle one
+ 0   - 0

b. A Binary "one" in the sign position of a word indicates____________

86. In Division, the Quotient is always in the __________ register.

87. Show the Op. Code of STQ, as it would look in storage.

1 2 3 4 5 6 7 8 9 10 11

88. a. Show a machine word containing the following fixed point number:

7342.1231_8

b. Indicate the position of the Binary point.

89.

Instructions	Contents of A	Result in AC:
CLA A	127_8	
SUB B		
	Contents of B	
	36_8	

90. a.

Add: (+)	Subtract: (-)	Multiply: (X)
+35	+35	+35
-39	-39	-39

Sign of Result:

Divide: (÷) $\frac{+35}{-39}$ Quo. Rem.

91. Show the following in normalized form:

a. $765._{10}$ [x10] c. $.00276_{10}$ [ ]

b. 22.16_{10} [x10] d. 100.011_2 [ ]

92. Show the "characteristic" of the following floating point number:

22_{10}

CHAR.	
1 2 3 4 5 6 7 8	

93. Show the entire floating point word for the following number:

326_{10}

Char.	Mantissa

94. Add two fixed point numbers (A + B). Move so that the Binary point in the AC will be between positions 9 and 10.

OP	VARIABLE	
CLA	A	(BO)
ADD	B	(BO)
[ ] ?	[ ] ?	

95. The only instruction allowing for a three-way branch, is [ ]

96. In writing a program on a Symbolic Coding Sheet, the Loc. Code is placed starting in column______, the Op. Code starts in column______ and the Address in column_____. Comments may not extend beyond column_____.

97. To indicate whether each of the following is an Element, Term or Expression, use the following symbols. Element: E, Term: T, Expression: X.

a. 500/7520 [ ] d. HOLD + A^2 * c [ ]

b. TOTAL [ ] e. ABZ * AB3/X [ ]

c. ALPHA * BETA [ ] f. A + B * C + X^2 - Z [ ]

98. TRA *+2 means: ______________________________

99. HOLD PZE 15, 2, 27. Show the contents of storage location HOLD in Binary form(leave Sign and pos. 1 and 2 blank).

100. Show the Octal representation of the following constants:

a. DEC 26B26

b. OCT 2211

c. DEC .003906B0

d. DEC 7.

101. Take the result of problem 99 and apply the following instructions:

```
STA  200
STD  300
STT  400
```

Show the pertinent portions of the above locations after the instructions have been executed.

200 300 400

102. When an instruction contains a TAG, the address of the instruction is called the __________address.

103. Index Register 2 looks like this: 0 ← → 0 1 1 0

The instruction is: TIX A9, 2, 3.

a. After instruction executed, how will XR2 look?

b. Will control go to A9 or to next instruction?_______

104. Contents of XR4

Instruction: TXL HOLD, 4, 124
Control would be transferred to______________.

105. a. What do we put into an Index Register, to change the address of an instruction from 130 to 125?______________

b. Which instruction is best used for this purpose?

__________________.

106. Instruction: STO ** 5 This means:______________

____

107. ADD = 250 means:__________________________

____

PROBLEM:

108. Given 10 floating point numbers located in AA through AA + 9. Given one floating point number located in BONE. The numbers that are greater than zero and algebraically less than or equal to BONE, will be added together in location TOTAL and those that are greater than BONE will be added together in location HOLD. Ignore numbers less than or equal to zero. Flow chart before attempting to code the problem.

LOC	OP	VARIABLE	REMARKS

CORRECT ANSWER

PROBLEM

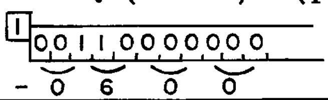
83. $759_{10} = 1367_8$
$= 001\ 011\ 110\ 111_2$ (pg. 11)

84. $1563_8 = 883_{10}$
$0431_8 = 281_{10}$
$1164_{10} = 2214_8 =$
$010\ 010\ 001\ 100_2$ (pg. 12)

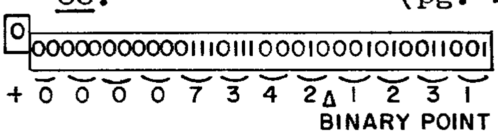
85. a. +0 (pg. 18)
b. minus (-) (pg. 15)

86. MQ (pg. 18)

87. STQ (-0600) (pg. 23)



88. (pg. 27)

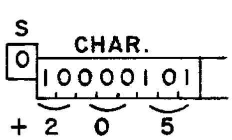


89. $127_8 = 87_{10}$
$36_8 = 30_{10}$ (pg. 28)
$57_{10} = 71_8$

90. a. $\boxed{-}$ b. $\boxed{+}$ c. $\boxed{-}$
d. $\boxed{-}$ Q, $\boxed{+}$ R (pgs. 27,29)

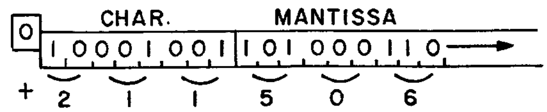
91. a. $.765 \times 10^3$
b. $.2216 \times 10^2$
c. $.276 \times 10^{-2}$
d. $.100011 \times 2^3$
(pg. 41)

92. $22_{10} = 26_8 = 010110._2$
$= .10110 \times 2^5 (200+5=205_8)$

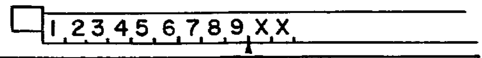


(pg. 41)

93. $326_{10} = 506_8 = 101000110._2$
$= .10100011 \times 2^{11}$
$(200 + 11 = 211_8)$



94. $\boxed{\text{LRS}}$ $\boxed{9}$ (pg. 74)



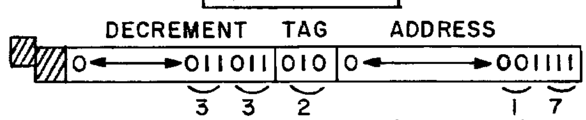
95. $\boxed{\text{CAS}}$ (pg. 77)

96. 1, 8, 16, 72 (pg. 59)

97. a. T d. X
b. E e. T
c. T f. X (pg. 63)

98. Transfer to the second instr. beyond the "Transfer" instr.
(pg. 63)

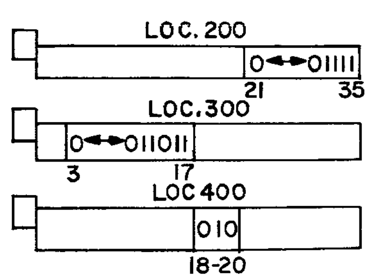
99. $\boxed{15_{10} = 17_8 \text{ Address}}$ $\boxed{2_{10} = 2_8 \text{ Tag}}$
$\boxed{27_{10} = 33_8 \text{ Decrement}}$



(pg. 81)

100. a. +000000032Δ000
b. +000000002211
c. +002000000000
d. +203700000000
(pg. 83)

101.



(pg. 89)

CORRECT ANSWER

102. Presumptive (pg. 93)

103.
a. | 0 ←——→ 011 |

b. A9 (pg. 95)

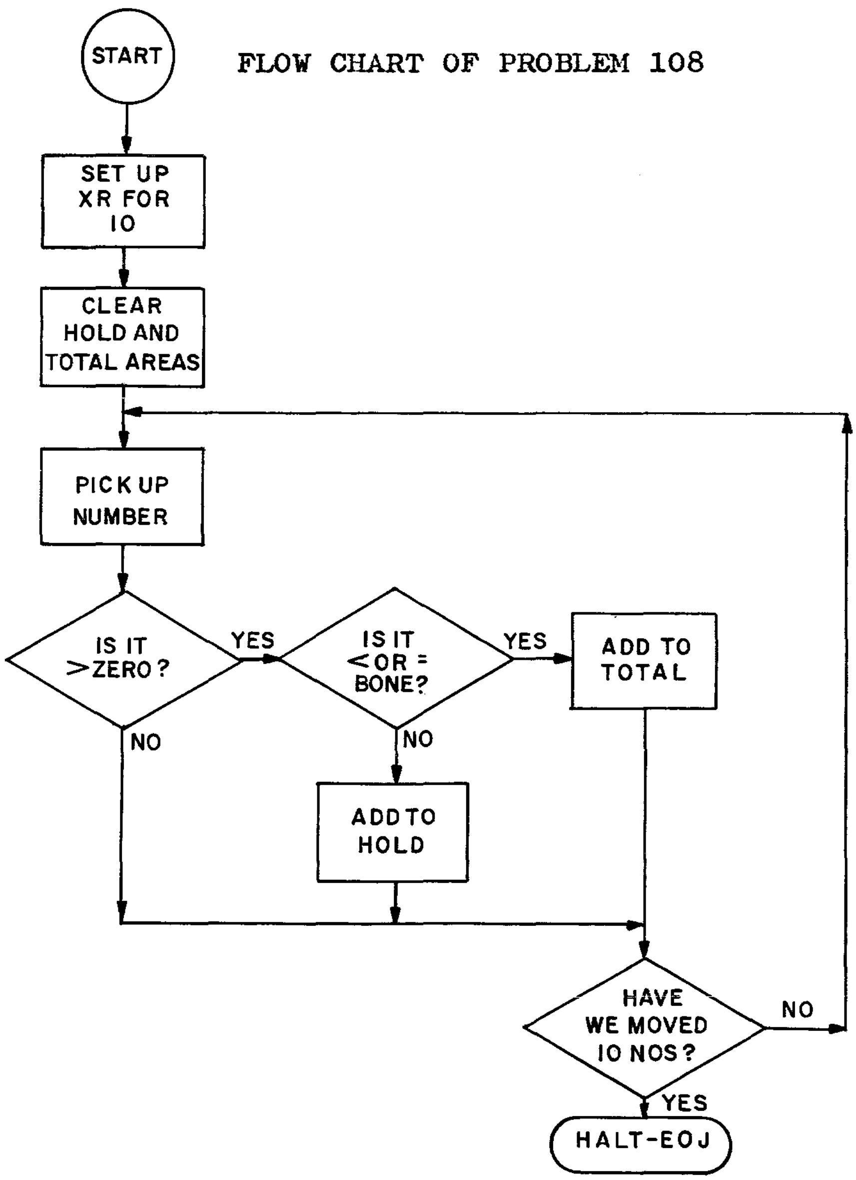
104. HOLD (XR4 of 123 is less than Decrement of 174). Remember that all instructions are written in Decimal unless otherwise specified. (see page 96)

105. a. 5
b. AXT (pg. 96)

106. Store the contents of the AC into this location times 5. (pg. 63)

107. Add a constant of 250. (pg. 83)

FLOW CHART OF PROBLEM 108



CORRECT ANSWER

PROBLEM 108:

```
LOC     OP      VARIABLE            REMARKS
        COUNT   22
START   AXT     10, 1
        STZ     HOLD
        STZ     TOTAL
        CLA     AA + 10, 1
        TPL     CHKBI               If   zero, go to CHKBI
        TIX     START + 3, 1, 1     Back to the CLA instr.
        HTR     *

CHKBI   CAS     BONE                Compare with BONE
        TRA     ADDHI               (AC > )
        TRA     * + 1               (AC = )
        FAD     TOTAL               (AC < )
        STO     TOTAL
        TRA     CHKBI-2             Transfer to TIX instr.

ADDHI   FAD     HOLD
        STO     HOLD
        TRA     CHKBI-2             Transfer to TIX instr.

BONE    BSS     1
AA      BSS     10
HOLD    BSS     1                   Allocate storage
TOTAL   BSS     1                   locations.
        END
```

LESSON 10

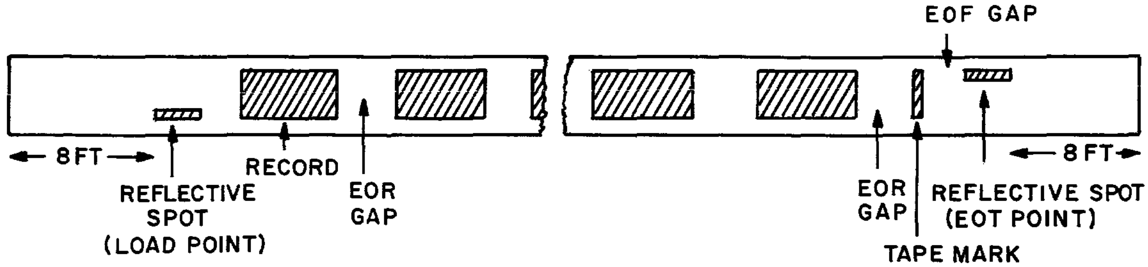
TAPE: On page viii, at the beginning of the book, several paragraphs were included on Data Channels and on tape. It may be worthwhile to review it at this time. There are a number of terms used in connection with tape, that the beginner must familiarize himself with before he can start the study of tape handling.

Proper handling of Input and Output is one of the most difficult areas to learn in programming. This course will not attempt to cover it in an exhaustive manner as only experience can give the programmer a complete understanding of this topic. The major aspects and instructions will be covered--enough so that a general understanding will be gained by the student.

REFLECTIVE SPOT: A normal tape is about 2400 feet long. It takes 6 to 8 feet on each end to wind on the tape drives. The tape has a little magnetic mark, called reflective spot, near the beginning. This is the Load Point of the tape (where Read or Write will begin). There is also a reflective spot near the end of the tape, beyond which writing should not be done. Checking for the reflective spot at the end of the tape must be done by the program.

TAPE MARK, END-OF-RECORD GAP, END-OF-FILE GAP: At the bottom of the page is a symbolic representation of a tape which shows all of the areas named here. A tape record contains the same bits that we have been dealing with in computer storage except that they are stored on tape as magnetic spots. Between the groups of magnetic spots are blank areas of tape, approximately $\frac{3}{4}$ inch wide. These are called end-of-record gaps. The gap after the last record on tape is called the end-of-file gap. This last gap and the tape mark which precedes it, constitute the end-of-file and when this is reached, the tape may be rewound and unloaded from the tape drive. It must be understood that an end-of-file (designated by the tape mark) is a record just like any other record on tape.

Total Length of Tape - 2400 feet



INPUT/OUTPUT INSTRUCTIONS AND COMMANDS

1. MISCELLANEOUS

RTD (READ TAPE DECIMAL) Octal code: +0762. Channel (A through H) must be specified (i.e. RTDA). This instruction, followed by an RCH instruction causes the computer to read one record into storage. Reading will be accomplished from the Input/Output device specified in Y. The Channel must also be specified in Y. Tape density must be compatible. In other words, attempting to read a tape in one density, that was recorded in another density, will cause both detected and undetected errors.

WTD (WRITE TAPE DECIMAL) Octal code: +0766. Channel (A through H) must be specified (i.e. WTDA). This instruction without the accompanying RCH instruction causes 3.75 inches of blank to be written. It is used to jump over a bad spot in the tape. With the RCH (page 121), a normal record is written on tape.

---

2. INPUT/OUTPUT OPERATIONS

BSR (BACKSPACE RECORD) Octal code: +0764. This instruction causes the tape, designated by Y, to back up until an end-of-record gap or load point is reached. It is used in the tape error routines. Channel (A-H) must be specified.

WEF (WRITE END-OF-FILE) Octal code: +0770. This instruction causes the tape, designated by Y, to write an end-of-file gap and a tape mark, indicating the end-of-file (EOF). Channel (A-H) must be specified.

REW (REWIND) Octal code: +0772. This instruction causes the tape, designated by Y, to rewind to the load point. At this time it is ready to be run again. Channel (A-H) must be specified.

RUN (REWIND AND UNLOAD) Octal code: -0772. This instruction causes the tape, designated by Y, to rewind to the load point and automatically set to be unloaded. Channel (A-H) must be specified.

---

3. CONTROL INSTRUCTIONS

TCO (TRANSFER IF CHANNEL IN OPERATION) Octal code: +0060. If the specified channel (A-H) is in operation, the computer takes its next instruction from location Y. If the channel is not in operation, the computer takes the next instruction in sequence.

TRC (TRANSFER ON REDUNDANCY) Octal code: +0022
The Channel (A-H) must be specified. This concerns the internal parity check. If parity is bad, an indicator turns on. The indicator is tested with this instruction. If the indicator is on, it is turned off and the computer takes its next instruction from Location Y. If the indicator is off, the computer takes the next instruction in sequence.

TEF (TRANSFER ON END-OF-FILE) Octal code: +0030
When the EOF gap is reached while reading, an indicator is turned on. This instruction tests the indicator. If it is on, it is turned off and the computer takes its next instruction from location Y. If it is off, the computer takes the next instruction in sequence. Channel (A-H) must be specified.

---

4. CHANNEL INDICATORS:

BTT (BEGINNING-OF-TAPE TEST) Octal code: +0760.
Channel (A-H) must be specified. If there is a backspace (BSR) given when tape is at load point, an indicator turns on. This tests the indicator. If it is on, it is turned off and the computer takes the next instruction in sequence. If it is off, the computer skips one instruction.

ETT (END-OF-TAPE TEST) Octal code: -0760.
Channel (A-H) must be specified. When end of tape is reached on writing, an indicator turns on. This tests the indicator. If it is on, it is turned off and the computer takes the next instruction in sequence. If it is off, the computer skips one instruction.

---

5. INPUT/OUTPUT TRANSMISSION INSTRUCTION:

RCH (RESET AND LOAD CHANNEL) Octal code: +0540 (for Channel A). Channel (A through H) must be specified. This instruction must be given immediately following a Read Select or a Write Select instruction, if transmission of data is to occur. The computer will not Read into storage or Write on tape unless the RCH instruction is present.

---

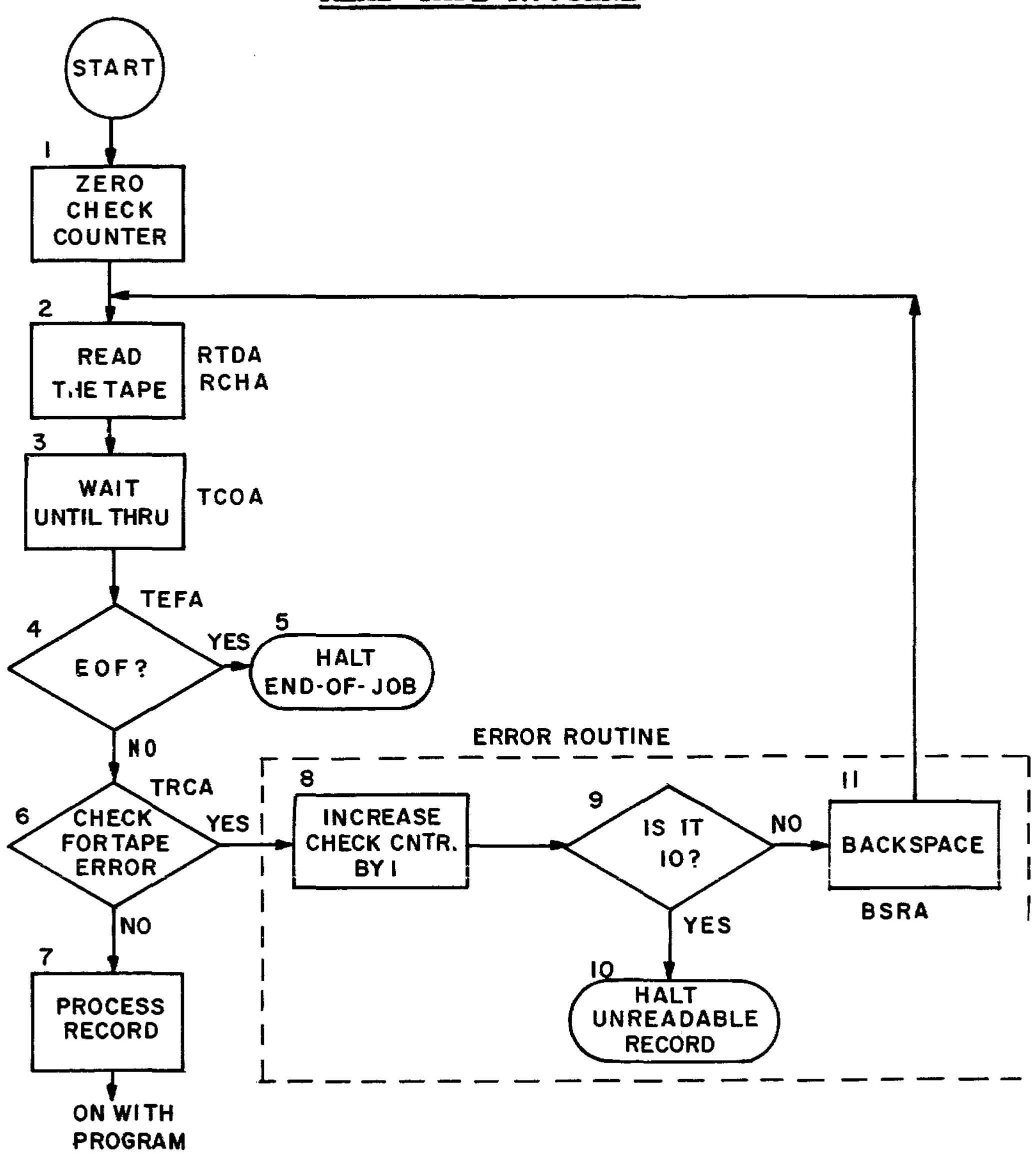
6. DATA CHANNEL COMMANDS

IOCD (I/O UNDER COUNT CONTROL AND DISCONNECT)
For input--this command will read the number of words specified in the Decrement, beginning with the word specified by the Address.
For output--outputs the number of words specified in the Decrement, beginning with the word specified by the Address. After completion, stops the execution of any other Channel Command.

IORT (INPUT/OUTPUT OF A RECORD AND TRANSFER)
Input--always disconnects the Channel at the end of a record or when the count in the Decrement goes to zero (whichever comes first).
Output--writes a record containing the number of words specified in Decrement portion of the Command. Starts to write from what is in the Address portion of the Command.

If a Load Channel Command (LCH) is waiting, the next Command will be taken from the Address portion of the Load Channel, otherwise a normal disconnect occurs.

READ TAPE ROUTINE



EXPLANATION

An initial decision is made to try to read the tape ten times in the event of a bad piece of tape. There is an internal "bit" check (called parity check) which tells the computer if there is anything wrong with what it is reading.

Block 1: A counter is set up at zero to keep track of reading until ten "reads" are reached.

Block 2: A tape record is read by the computer.

Block 3: No further processing until end-of-record is reached.

Block 4: Test for end-of-file.

Block 5: If it is end-of-file, there is nothing more to be read, so the tape is rewound and unloaded.

Block 6: Check for tape error (called parity check).

Block 7: If there is no tape error, the program continues with its normal processing of the record which is now located in computer storage.

Block 8: If there is a tape error (called parity error), increase the Check Counter by one until a total of ten tries have been made to read the tape.

Block 9: Check to see if the Counter is at 10.

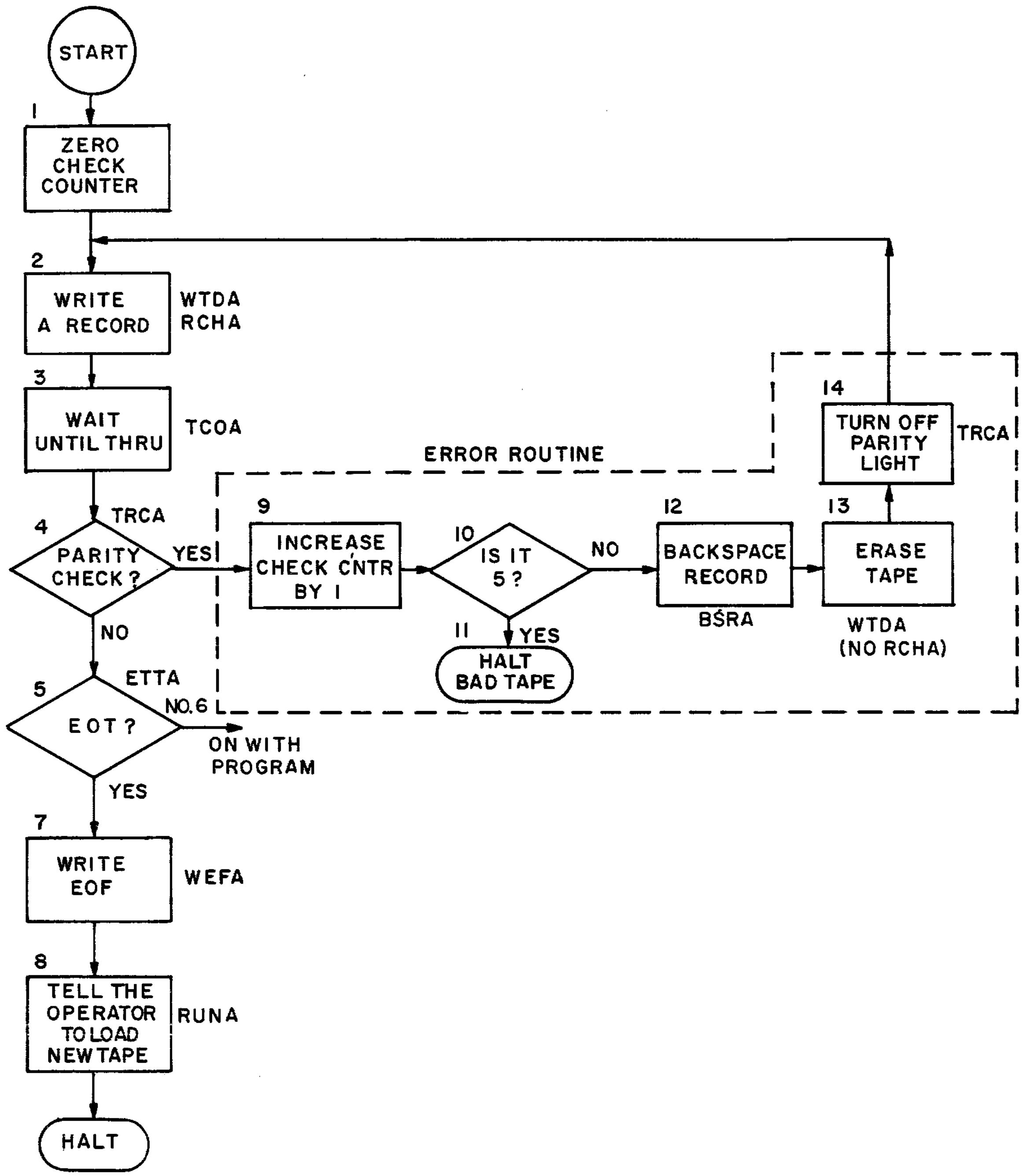
Block 10: If it is 10, halt the program. The record cannot be read by the computer.

Block 11: If it is not 10, backspace the record and go back to Block 2, to try to read the same record again.

Note the Input/Output Instructions associated with the various blocks. These are shown in greater detail for both Read and Write operations on pages 126 and 127. Channel A was arbitrarily chosen for the example.

No instructions are designated for blocks 1, 5, 7, 8, 9, and 10 since these are not specifically input/output instructions.

WRITE TAPE ROUTINE



EXPLANATION

An initial decision is made to try to write five times in the event of a bad piece of tape. The parity check mentioned in the Read Tape Routine, also applies to write tape.

Block 1: A counter is set up at zero to keep track of writing until five "writes" are reached.

Block 2: A record is written on tape by the computer.

Block 3: No further processing until the writing of the record is completed.

Block 4: Test for tape error (parity error).

Block 5: If no tape error, test for end-of-file.

Block 6: If it is not end-of-file, the program continues with its normal processing.

Block 7: If it is end-of-file, write end-of-file.

Block 8: Rewind and unload this tape and if processing is not finished, have the operator load a new tape.

Block 9: In Block 4, if there is a tape error, increase the check counter by one.

Block 10: Test the Check Counter for 5.

Block 11: If it is 5, write has been attempted five times without success. Stop the program.

Block 12: If it is not 5, backspace the record.

Blocks 13 and 14: Erase the tape, turn off the tape error (parity) light and try to write the record again.

Lesson 10, (cont'd)

EXAMPLE: Read tape unit 4 on Channel A. Process the data and write out on Channel C, tape unit 1. Stop when end of file (EOF) is reached. (See note at bottom of the page.)

	LOC	OP	VARIABLE FIELD	REMARKS
(See Note)	X	TAPENO	A4B	Defines X as Chan. A, unit 4, Binary
	Y	TAPENO	CIB	Defines Y as Chan. C, unit 1, Binary
	READ	STZ	CTX	Store zeros in Read counter
		TCOX	*	Wait
		RTDX		Read Chan. A, unit 4, Binary
		RCHX	IOIN	Reset and load Chan. A
		TCOX	*	Wait until record is read
(End of Read Routine)		TEFX	EOF	If it is End-of-File, go to EOF
		TRCX	PEX	If there is Parity Error, go to PEX
	Process record and place output into AREA 1			
	WRITE	STZ	CTY	Store zeros in Write Counter
		TCOY	*	Wait
		WTDY		Write a record
		RCHY	IOOUT	from Area 1
		TCOY	*	Wait until through writing
		TRCY	PEY	If there is parity error, go to PEY
		ETTY		Is it End-of-Tape?
		TRA	EOF	If End-of-Tape, go to EOF
(End of Write Routine)		TRA	READ	If not End-of-Tape, go back to read next record

Note that at the beginning of the program, the Op. Code TAPENO, with a one character location code was used to define the Channel, Tape Unit, and type of notation (Binary). This is much simpler than using the actual channels (A through H) on each succeeding instruction. Also notice how easily the counter is increased and checked with the use of literals in the error routines on the next page.

EXAMPLE--continued

LOC	OP	VARIABLE FIELD	REMARKS
CTX	PZE		Define CTX (X counter)
CTY	PZE		Define CTY (Y counter)
IOIN	IORT	AREA, , 100	Channel Command for input. Decrement of 100 (chosen arbitrarily)
AREA	BSS	100	Allocate 100 positions for AREA
EOF	HTR	*	Halt - end-of-job
		Pick up loose ends	
PEX	CLA	CTX	
(Error rou-	ADD	= 1	Increase counter by 1
tine for	STO	CTX	
Read)			
	SUB	= 10	To check if counter equals 10
	TZE	EOF	If 10 tries, go to EOF to halt program. Unreadable tape
	BSRX		If not 10 tries, backspace record
	TRA	READ + 1	Go back to READ + 1 and try again
IOOUT	IOCD	AREA 1, , 45	Outputs number of words specified in Decrement
AREA 1	BSS	45	Allocate 45 storage positions to AREA 1 (again arbitrarily chosen)
PEY	CLA	CTY	
(Error	ADD	= 1	Increase counter by 1
routine for	STO	CTY	
Write)	SUB	= 5	To check if counter equals 5
	TZE	EOF	Unwriteable tape, go to EOF
	BSRY		Back up and erase tape
	WTDY		
	TCOY	*	Wait
	TRCY	*	Turn off parity light
	TRA	WRITE + 1	Go back to WRITE + 1, and try again.

BUFFERING: A buffer is not a separate piece of equipment. It is an area of storage, assigned by the programmer, specifically to accept Input/Output information.

The Read and Write routines shown on pages 122 and 124, do not show how this is accomplished with Buffering. In some instances, using the buffering technique speeds up the procedure considerably since one record may be processed at the same time that another is being read.

This technique is not shown here because most installations now have ready-made Input/Output Packages which do the job of reading and writing in the most optimum manner. Where the Package is available, it should be used in preference to writing individual Input/Output routines.

INPUT/OUTPUT PACKAGE: Most organizations have prepared Input/Output programs which may be utilized in conjunction with nearly all normal programs. This saves considerable time in programming because usually a great deal of the programming effort deals with Input and Output processing.

The new programmer must familiarize himself with the Input/Output Package of his organization and merely tie it in to his own program.

The preceding pages, dealing with Input and Output routines, were important primarily so that the new programmer would have a working understanding of what occurs during Read and Write operations. Also, there are occasions when Input/Output Packages are not available and therefore Input and Output must be programmed along with the basic problem.

WORK AREA

PROBLEM:

109. Read tape unit 8 on Channel E. Place the first word of the record into storage at loc. HOLD, go back and read another record, placing the first word into HOLD + 1. Halt when end-of-file is reached.

CORRECT ANSWER

PROBLEM 109:

LOC	OP	VARIABLE FIELD	REMARKS
Z	TAPENO	E8B	Defining tape unit 8, Chan. E, Binary
READ	STZ	COUNT	Store zeros into counter
	RTDZ		Read first word of
	RCHZ	IOC	one record.
	TCOZ	*	Wait until record is read
	TEFZ	EOF	If we have reached end of file, go to EOF.
	TRCZ	PE	If tape error, go to PE
	CLA	IOC	Increase location by
	ADD	= 1	one to store the one
	STO	IOC	word for the next record to come in.
	TRA	READ	Go back to beginning to read next record.
PE	CLA	COUNT	Move counter into AC
	ADD	= 1	Add 1
	STO	COUNT	Place back into storage
	SUB	= 10	Check to see if counter has gone to 10 (if so, indicates bad tape).
	TZE	BT	If tried to read 10 times, bad tape. Go to BT (which is equivalent to EOF)
	BSRZ		If not yet 10 tries, backspace the record.
	TRA	READ + 1	Go back to try reading the record again.
COUNT	BSS	1	Allocate one storage position to counter.
EOF	HTR	*	End of file. Halt prog.
BT	EQU	EOF	Define that BT is equivalent to EOF
IOC	IORT	HOLD, , 1	I/O command to read first word of each record.
HOLD	BSS	1000	

WORK AREA

PROBLEM:

110. Take the data from loc. HOLD, HOLD + 1, HOLD + 2, etc., and write it out on Channel H, tape unit 3. When HOLD + 999 is reached, write EOF and stop the program.

LOC	OP	VARIABLE FIELD	REMARKS

CORRECT ANSWER

PROBLEM 110:

LOC	OP	VARIABLE FIELD	REMARKS
X	TAPENO	H3B	Defining tape 3, Chan. H, Binary
	STZ	CT	Stores zeros into counter
LOOP	WTDX		Write a record
	RCHX	IO	
	TCOX	*	Wait until through writing
	TRCX	PE	If there is parity error, go to PE
	STZ	CT	If no parity error, zero counter
LOOP 1	WEFX		Write end-of-file
(End-of-file	TCOX	*	Wait until write is finished
routine)	TRCX	PE1	If parity error for EOF, go to PE1
BT	HTR	*	Bad tape - Halt
PE	CLA	CT	Move counter into AC
	ADD	= 1	Add 1
	STO	CT	Put back into storage
	SUB	= 5	Have we tried 5 times?
(parity	TZE	BT	If yes, go to BT to Halt
error	BSRX		If no, backspace record
routine)	WTDX		Erase tape
	TCOX	*	Wait until through
	TRCX	*	Turn off parity light
	TRA	LOOP	Go back to try to write again
PE1	CLA	CT	
	ADD	= 1	
(parity	STO	CT	
error	SUB	= 5	
routine	TZE	BT	
for EOF)	BSRX		
	WTDX		
	TCOX	*	
	TRA	LOOP 1	
IO	IOCD	HOLD, , 100	Outputs no. of words specified in Decr. (100)
CT	PZE		

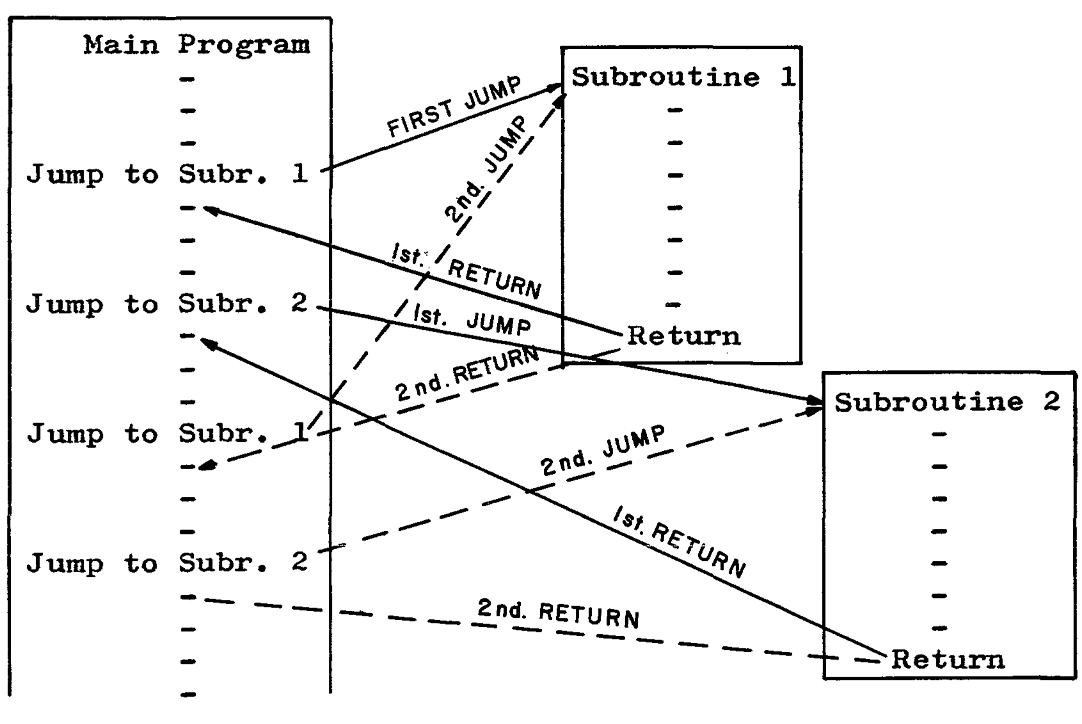
LESSON 11

SUBROUTINES: In nearly all program writing it becomes necessary to repeat certain program steps. It is usually not desirable to write these steps over and over as the need arises. It is much more practical to write the steps once and then arrange to jump to this group of steps when necessary. A subroutine is essentially just this--a group of program steps which may be used repeatedly as required.

There are two types of subroutines: Open and Closed. The Open subroutine is inserted into the main program and the Closed subroutine is separate and apart from the main program. The Closed subroutine is the most economical and the most commonly used, but it is difficult to instruct the subroutine as to where in the main program it should return when it is finished processing. The process used is subroutine linkage.

SUBROUTINE LINKAGE: There are several ways of linking a subroutine to the main program. One of the most simple and economical is to use Index Registers to provide a path to and from the main program. This has the added advantage that the programmer need not be aware of the actual address of the return jump and may continue to write his program in symbolic. Some of the other linkage methods require the knowledge of the actual address for the return jump to the main program. An example of subroutine linkage may be found on the following page.

Symbolically represented, subroutine linkage would look like this:



Lesson 11, (cont'd)

EXAMPLE 1: Suppose that it was necessary to sum three variables and leave the sum in a fourth variable and it was necessary to do this for many different sets of variables. A portion of the program could be:

```
        TSX     SUM, 4   (see Lesson 8 for explanation of TSX)
        PZE     A        (1st variable)
        PZE     B        (2nd variable)
        PZE     C        (3rd variable)
        PZE     D        (answer)
```

If the program were as above, the subroutine could be:

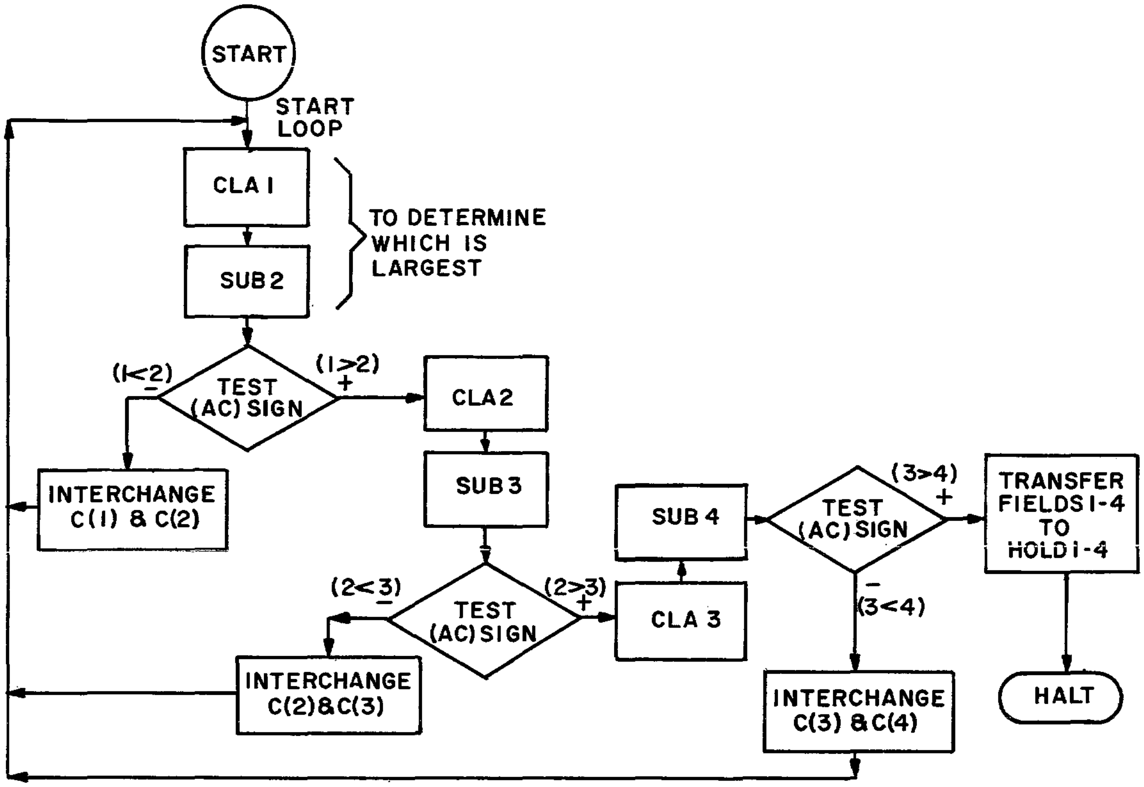
```
SUM     CLA*    1, 4
        FAD*    2, 4
        FAD*    3, 4
        STO*    4, 4
        TRA     5, 4
```

The asterisk (*) after the Op. Code means that the instruction is indirectly addressed. Detailed explanation of this technique and additional examples may be found in Lesson 12. It may be worth while delaying the detailed study of this example until Indirect Addressing has been covered in Lesson 12.

EXAMPLE 2: Let us suppose that there is a long program, with a number of parts, each going to a particular subroutine, and from there back to the beginning of the loop. The flow chart below shows such a program. (This is the flow chart for the program on the following page. It is not truly a closed subroutine, but it does show how a program can be manipulated with Index Registers.)

Notice that on each test for transfer, if the condition is minus, the program goes to an interchange routine and from there back to the beginning of the loop. This may be graphically represented as follows:

```
        PROGRAM
          -
LOOP    CLA
          -                 First part of loop. Transf. on + to
          -                 next part. If not +, go to subr. INTER
        TPL     NO2
        TRA     INTER
NO2       -                 Second part. TPL to next part or go to
          -                 subr.
        TPL     NO3
        TRA     INTER
NO3       -                 Third part. TPL to next part or go to
          -                 subr.
        TPL     NO4
        TRA     INTER
NO4       -                 Fourth part.  Finish program.
        HTR     *
INTER     -                 Subroutine. Always goes back to start
        TRA     LOOP        of LOOP
```



WORK AREA

PROBLEM 111. Given fixed point integers, located sequentially in Field 1, Field 2, Field 3 and Field 4. Sort so that the largest value will go into location HOLD, next largest in HOLD + 1, etc.

LOC	OP	VARIABLE	LOC	OP	VARIABLE

CORRECT ANSWER

PROBLEM 111.

```
LOC        OP      VARIABLE          REMARKS

           COUNT   32
LOOP       AXT     0, 1
           CLA     FIELD, 1          Start of LOOP
           TXI     * + 1, 1, -1      Compare 1 and 2
           SUB     FIELD, 1          (XR1 = -1)
           TPL     2VS3              If 1 > 2, go to 2VS3
           TRA     INTER             If 1 < 2, go to subr.
------------------------------------------------------------
2VS3       CLA     FIELD, 1          (XR1 = -1)
           TXI     * + 1, 1, -1      Compare 2 and 3
           SUB     FIELD, 1          (XR1 = -2)
           TPL     3VS4              If 2 > 3, go to 3VS4
           TRA     INTER             If 2 < 3, go to subr.
------------------------------------------------------------
3VS4       CLA     FIELD, 1          (XR1 = -2)
           TXI     * + 1, 1, -1      Set XR for 4
           SUB     FIELD, 1          (XR1 = -3)
           TPL     MOVE              If 3 > 4, go to MOVE
           TRA     INTER             If 3 < 4, go to subr.
------------------------------------------------------------
MOVE       AXT     4, 2             }
           CLA     FIELD + 4, 2     }Move to HOLD area
           STO     HOLD +  4, 2     }
           TIX     * -2, 2, 1        Small loop back to CLA
                                     until all 4 numbers
                                     are moved.
           HTR     *                 Halt - end of job.
------------------------------------------------------------
INTER      CLA     FIELD, 1         }
(Sub-      TXI     * + 1, 1, 1      }Exchange - last cell
routine)   LDQ     FIELD, 1         }defined by XR1 with
           STO     FIELD, 1         }previous cell (word)
           TXI     * + 1, 1, -1     }
           STQ     FIELD, 1
           TRA     LOOP              Back to start of LOOP
------------------------------------------------------------

FIELD      BSS     4
HOLD       BSS     4
           END
```

LOGICAL OPERATIONS: Logical operations have a special way of operating on a 36 bit word. They are used primarily for masking operations, which are discussed on page 140. The sign position is simply another bit and is not considered separately from the other 35 bits in the word.

Special rules apply when two numbers are combined by logical instructions. These rules are as follows:

1. Logical AND operations: Ones in both numbers equal one. Otherwise zero.

Example:
```
  001011
  001101
= 001001
```

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

2. Logical OR operations: A one in either number causes a one in result. Otherwise zero.

Example:
```
  001011
  001101
= 001111
```

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

3. Exclusive OR operations: A one in only one of the numbers equals one. Otherwise zero.

Example:
```
  001011
  001101
= 000110
```

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

In logical operations, when two numbers are combined, they are matched bit for bit as shown in the examples above. Notice the differences in the resultant numbers. Converting the Binary numbers above to Octal:

AND op.	OR op.	Excl. OR op.
13	13	13
15	15	15
$= 11_8$	$= 17_8$	$= 6_8$

Do not confuse logical operations with the normal arithmetic operations.

INSTRUCTION: CAL (Clear and Add Logical Word) Octal code: -0500
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

DESCRIPTION: This is identical to the CLA (Clear and Add) instruction except that the sign goes into the P position of the AC.

INSTRUCTION: SLW (Store Logical Word) Octal code: +0602
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

DESCRIPTION: This is identical to the STO (Store) instruction except that the bit in position P of the AC goes into the sign position of the word.

INSTRUCTION: ANA (AND to Accumulator) Octal code: -0320
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

DESCRIPTION: Each bit of the c(Y) is matched with the corresponding bit in the c(AC) (positions P, 1-35). The result of the matching (using the rules laid down in page 137) will be in the AC. AC positions S, Q are set to zero.

INSTRUCTION: ANS (AND to STORAGE) Octal code: +0320
FORMAT: (Type B)

OP CODE	IA		TAG	Y
S,1	11 12-13		18-20 21	35

DESCRIPTION: Each bit of the c(AC) (positions P, 1-35) is matched with the corresponding bit in the c(Y). The result will be in storage at location Y.

EXAMPLES:

BEFORE	INSTRUCTION	AFTER	
AC: 1 0 0 1 1	ANA	1 0 0 1 0	in AC
Y: 1 1 0 1 0	ANS	1 0 0 1 0	in Y

<u>INSTRUCTION</u>: ORA (OR to Accumulator) Octal code: -0501
<u>FORMAT</u>: (Type B)

OP CODE	IA	//////	TAG	Y
S,1 — 11	12-13		18-20	21 — 35

<u>DESCRIPTION</u>: Each bit of the c(Y) is matched with the corresponding bit in the c(AC) (P, 1-35). The result (using the rules on page 137) will be in the AC. The c(Y) and the S and Q positions of the AC remain unchanged. The sign of Y will be in the P position in the AC.

<u>INSTRUCTION</u>: ORS (OR to STORAGE) Octal code: -0602
<u>FORMAT</u>: (Type B)

OP CODE	IA	//////	TAG	Y
S,1 — 11	12-13		18-20	21 — 35

<u>DESCRIPTION</u>: As above, except that the result will be in the c(Y) and the bit in position P of the AC will be in the sign position of Y.

<u>INSTRUCTION</u>: ERA (Exclusive OR to Accumulator) Octal code: +0322
<u>FORMAT</u>: (Type B)

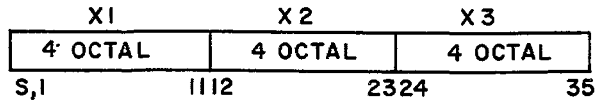
OP CODE	IA	//////	TAG	Y
S,1 — 11	12-13		18-20	21 — 35

<u>DESCRIPTION</u>: Exactly the same as the ORA instructions above, except that the rules for ERA apply (as shown on page 137).

<u>EXAMPLES</u>:

<u>BEFORE</u>	<u>INSTRUCTION</u>	<u>AFTER</u>	
AC: 1 0 0 1 1	ORA	1 1 0 1 1	in AC
	ORS	1 1 0 1 1	in Y
Y: 1 1 0 1 0	ERA	0 1 0 0 1	in AC

<u>MASKING - PACKING AND UNPACKING</u>: Quite often, the items to be used in a computer operation are small enough that more than one could fit into a machine word. This process is called <u>packing</u>. For example, if the numbers are no larger than three Decimal digits, they would convert to no larger than four Octal digits and three such numbers (complete with sign) could be placed into one machine word.



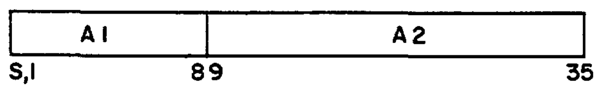
In this example, the signs would be in positions S, 12 and 24.

Packing a word in this manner, not only saves storage space, but also speeds up machine operating time since it takes less time for the computer to read or write the data.

If it is necessary to operate on one of the numbers packed into a word, it is necessary to <u>mask</u> out the other numbers. The <u>mask</u> may be set up by using the OCT pseudo op. code.

<u>EXAMPLE</u>: Two numbers are packed into a word as follows:

location HOLD:

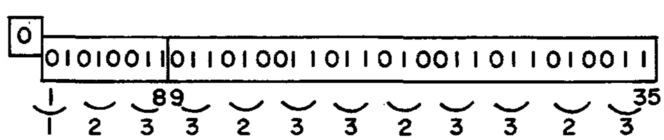


A2 is needed for other work. Mask out A1 (this is unpacking)

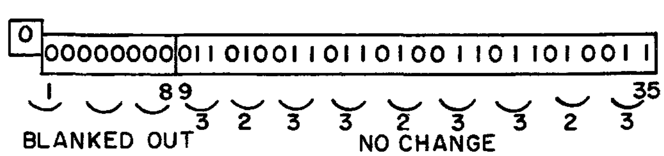
LOC	OP	VARIABLE FIELD	REMARKS
	CAL	HOLD	Move word into AC
	ANA	MASK	Add logical-MASK (defined below)
	ALS	9	Left shift to bring A2 into proper place in AC
	SLW	A2	Store from AC into loc. A2
MASK	OCT	000777777777	This will put 9 zeros into S-8 and 27 ones into 9-35 of the Mask word (Using AND rules- zeros will blank out the word while ones will have no effect).

Assume that A1 = 123_8 and A2 = 323323323_8

AC before MASK:

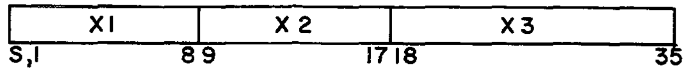


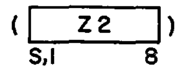
AC after MASK:
(using AND rules)



EXAMPLES continued:

Three numbers are packed into a word as follows:

location STAND: 

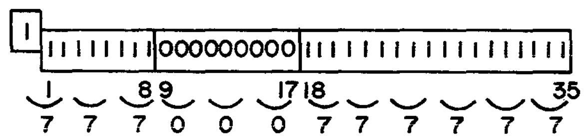
Mask out X2 and move Z2 (

) into the vacated X2 position.

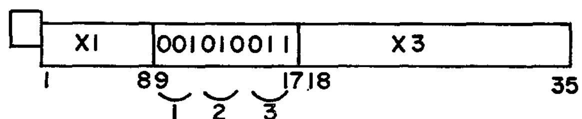
LOC	OP	VARIABLE FIELD	REMARKS
	CAL	MASK	Place MASK into AC
	ANS	STAND	Match MASK, bit for bit with c(STAND). This blanks out X2 in storage.
	CAL	Z2	Move Z2 into AC
	ARS	9	Shift right 9 positions to line up with positions 9-17
	ORS	STAND	OR to storage loc. STAND
MASK	OCT	777000777777	Sign and 1-8 will be ones, 9-17 will be zeros and 18-35 will be ones (Using OR rules - zeros will have no affect on the word while ones will blank out the word).

Assume that X2 = 123_8 and Z2 = 456_8

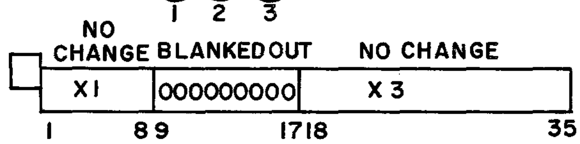
Before execution of ANS instr. (Mask in AC)



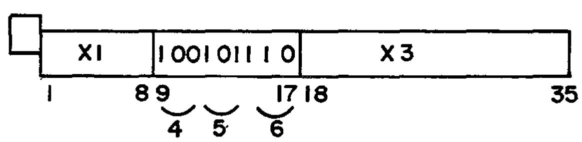
X1, X2, X3 in loc. STAND:



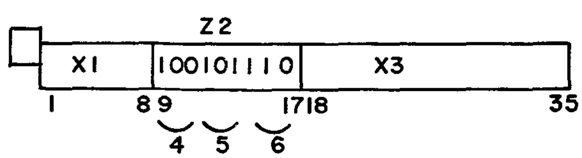
After execution of ANS instr. (contents of loc. STAND)



Before execution of ORS instr. (Z2 now in the AC)



After execution of ORS instr. (Z2 moves intact into loc. STAND)

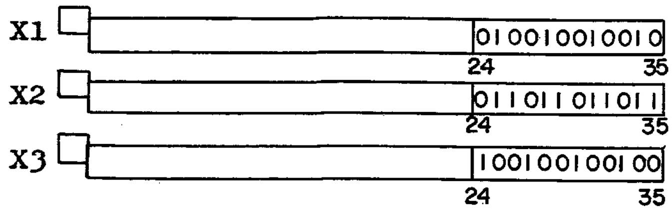


The zeros in loc. STAND are compared to 456_8 in the AC. Using the rules for OR, the 456_8 in the AC moves intact into loc. STAND and the job is finished.

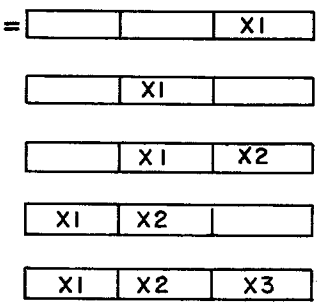
EXAMPLES continued:

X1, X2, and X3 are to be packed into location HOLD.
X1 = 2222_8 X2 = 3333_8 X3 = 4444_8

Before packing:



LOC	OP	VARIABLE FIELD	REMARKS
	CAL	X1	AC =
	ALS	12	
	ORA	X2	
	ALS	12	
	ORA	X3	
	SLW	HOLD	Store from AC into loc. HOLD
	HTR	*	Halt - end of job



INSTRUCTION: LGR (Logical Right Shift) Octal code: -0765
FORMAT: (Type B)

DESCRIPTION: The contents of the AC and MQ are treated as one long register (this includes the S,Q,P in the AC, and the S in the MQ). The contents are shifted to the right the number of places specified in positions 28-35 of (Y) the address portion of the instruction. Thesign of the AC will remain unchanged.

INSTRUCTION: LGL (Logical Left Shift) Octal code: -0763
FORMAT: (Type B)

DESCRIPTION: Identical to the LGR instruction, except that the shift is to the left. In both of the above instructions, vacated positions are filled with zeros. Any bits shifted left of position Q in the AC, will be lost.

EXAMPLES:

1. The example shown on page 142 may also be accomplished with the LGR instruction.

X1 = [| X1] (X1 in positions 24–35)

X2 = [| X2] (X2 in positions 24–35)

X3 = [| X3] (X3 in positions 24–35)

Store in loc. HOLD.

LOC	OP	VARIABLE FIELD	REMARKS: AC	REMARKS: MQ
	CAL	X3	[| X3]	[ ]
	LGR	12	[ ]	[X3 |]
	CAL	X2	[| X2]	[X3 |]
	LGR	12	[ ]	[X2 | X3 |]
	CAL	X1	[| X1]	[X2 | X3 |]
	LGR	12	[ ]	[X1 | X2 | X3]
	STQ	HOLD	Store from MQ into loc. HOLD	
	HTR	*	Halt - end of job	

2.

X1 = [X1 |] (X1 in positions S,1–11)

X2 = [| X2 |] (X2 in positions 12–23)

X3 = [X3 |] (X3 in positions S,1–11)

Store in loc. HOLD.

LOC	OP	VARIABLE FIELD	REMARKS: AC	REMARKS: MQ
	CAL	X3	[X3 |]	[ ]
	ARS	24	[| X3]	[ ]
	LGR	12	[ ]	[X3 |]
	CAL	X2	[| X2 |]	[X3 |]
	ARS	12	[| X2]	[X3 |]
	LGR	12	[ ]	[X2 | X3 |]
	CAL	X1	[X1 |]	[X2 | X3 |]
	ARS	24	[| X1]	[X2 | X3 |]
	LGR	12	[ ]	[X1 | X2 | X3]
	STQ	Hold	Store from MQ into loc. HOLD	

A Mask may be used very effectively to change instructions. This process is a little tricky at first, because the bits must be shuffled to do the required job.

EXAMPLE:

1. We wish to use a Mask to change the following instruction:

LOC	OP	VAR.		LOC	OP	VAR.
ABC	STO	HOLD	to	ABC	SLW	HOLD

In this case, only the op. code is to be changed:

From: STO = +0601
To: SLW = +0602

A mask may be set up with the pseudo op. OCT or with a literal (see page 83). The following instructions will do the job.

	LOC	OP	VARIABLE FIELD	REMARKS
		CAL	=Ø000200000000	into AC 000 000 000 010 → 0 0 0 2
STORAGE CONTAINS 000 110 000 001 → 0 6 0 1		ORS	ABC	into stor. 000 110 000 011 → 0 6 0 3
		CAL	=Ø777677777777	into AC 111 111 111 110 → 7 7 7 6
		ANS	ABC	into stor. 000 110 000 010 → 0 6 0 2

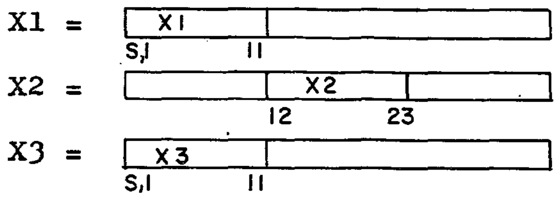
2. Change FAD (+0300), located in AA, to ANS (+0320)

	LOC	OP	VARIABLE FIELD	REMARKS
		CAL	=Ø002000000000	into AC 000 000 010 000 → 0 0 2 0
STORAGE CONTAINS 000 011 000 000 → 0 3 0 0		ORS	AA	into storage 000 011 010 000 → 0 3 2 0

The Octal 2 in the mask simply drops into place to replace the Octal zero. Many people find it easier to think in Octal rather than in Binary when preparing a mask.

PROBLEMS:

112. Pack X1, X2 and X3 into loc. HOLD.



This is the same as example 2 on page 143, but in this problem, work from the MQ into the AC, using left shifts.

LOC	OP	VARIABLE FIELD	REMARKS

113. Use a mask to change the following instructions:

ZZ CLA XYZ to ZZ CAL XYZ
(+0500) (-0500)

LOC	OP	VARIABLE FIELD	REMARKS

114. At the completion of problem 112, X1, X2 and X3 are packed in loc. HOLD. Unpack X2 and place it into loc. STAND, in orig. position.

LOC	OP	VARIABLE FIELD	REMARKS

CORRECT ANSWERS

PROBLEM 112.

LOC	OP	VARIABLE FIELD	REMARKS AC	REMARKS MQ
	LDQ	X1	[ ]	[X1 |]
	LGL	12	[| X1]	[ ]
	LDQ	X2	[| X1]	[| X2 |]
	SLW	TEMP	Save AC in temporary location	
	LGL	12	[ ]	[X2 |]
	CAL	TEMP	[| X1]	[X2 |]
	LGL	12	[| X1 | X2]	[ ]
	LDQ	X3	[| X1 | X2]	[X3 |]
	LGL	12	[X1 | X2 | X3]	[ ]
	SLW	HOLD	Store into loc. HOLD	
	HTR	*	Halt - end of job	
TEMP	BSS	1	Allocate storage posit. to TEMP	

PROBLEM 113.

LOC	OP	VARIABLE FIELD	REMARKS
	CAL	=Ø400000000000	
	ORS	ZZ	

The Octal 4 will put a one bit into the sign position, changing the + to a -.

PROBLEM 114.

LOC	OP	VARIABLE FIELD	REMARKS
	CAL	HOLD	Move HOLD into AC
	ANA	=Ø000077770000	Literal - the 7's will all be ones. This will leave X2, while masking out X1 and X3.
	SLW	STAND	Store into loc. STAND

LESSON 12

SENSE INDICATOR OPERATIONS: Before going into this area, turn back to page 17 and review subparagraph 2 on Sense Indicator Registers.

There are two types of switches on the 7090: (1) Sense Switches, which are located on the computer console, and are manipulated by the operators and (2) Sense Indicators, which are internal to the machine and are manipulated by the program.

There are six Sense Switches. The pseudo op. SWT (page 148) tests the setting of any switch. A group of sense indicator instructions are used to manipulate and test the sense indicators.

Each of the 36 bits in the Sense Indicator Register (SI Register) may be used as a switch or bits may be used in groups. They are turned on when the bits are set to "one" and off when set to "zero." The bits are manipulated by the programmer by the use of a Mask (see page 140).

Sense switches may be compared to switches on a railroad. In a railroad operation, it is known that at certain points along the track, the train must switch to either one of two branches depending on certain conditions that occur at the time the train reaches the switch or at some previous point in time. In the same manner, at the time a program is being written, it may be known that conditions will arise which will require that the program proceed along one of two branches at a later point.

At the point where the decision is to be made as to which branch to be taken, an instruction is used to test the sense switch. This is also true of sense lights (covered on page 152).

A few of the most valuable Sense Indicator instructions are defined on the following two pages. The instructions on page 148 are concerned with the movement of the full 36 bit word between the SI Register and either the AC or storage. The instructions on page 149, are used to test the SI Register. There are a number of other instructions used to test or to modify the SI Register. These may be found in the 7090 Reference Manual and will be self-explanatory when the following instructions are thoroughly understood.

PSEUDO OP. CODE: SWT (Sense Switch Test)
DESCRIPTION: This is a pseudo op. code which tests whether the sense switch (Y) is on or off. (Where Y = 1, 2, 3, 4, 5, or 6). If the sense switch is on, the computer skips one instruction. If the sense switch is off, the computer takes the next instruction in sequence.

INSTRUCTION: PAI (Place AC in Indicators) Octal code: +0044
FORMAT: (Type D)

DESCRIPTION: The c(AC), positions P and 1-35, replace the contents of the Sense Indicator Register. The c(AC) remain unchanged.

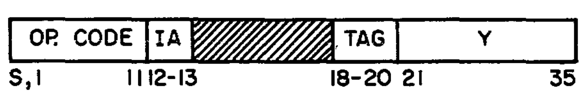
INSTRUCTION: PIA (Place Indicators in AC) Octal code: -0046
FORMAT: (Type D)

DESCRIPTION: This is the reverse of the PAI instruction above. Contents of the Indicator Register move into the AC (positions P, 1-35). Positions S and Q of the AC are cleared and the SI remains unchanged.

INSTRUCTION: LDI (Load Indicators) Octal code: +0441
FORMAT: (Type B)

DESCRIPTION: The c(Y) replace the contents of Sense Indicator Register. The c(Y) remain unchanged.

INSTRUCTION: STI (Store Indicators) Octal code: +0604
FORMAT: (Type B)



DESCRIPTION: The c(SI Register) replace the c(Y) in storage. The c(SIR) remain unchanged.

INSTRUCTION: ONT (On Test for Indicators) Octal code: +0446
FORMAT: (Type B)

OP. CODE	IA		TAG	Y
S,1	11 12-13		18- 20	21 35

DESCRIPTION: For each bit in the c(Y) that is a one, the corresponding bit of the Sense Indicator Register is examined. If all the positions examined in the SI Register are ones, the computer skips one instruction. If any of the positions examined in the SI Register do not contain a one, the computer takes the next instruction in sequence.

INSTRUCTION: OFT (Off Test for Indicators) Octal code:+0444
FORMAT: (Type B)

OP. CODE	IA		TAG	Y
S,1	11 12-13		18-20	21 35

DESCRIPTION: This is identical to the ONT instruction except that the SI Register is examined for zeros to compare with the ones in the c(Y). All zeros, skip one instruction. Any non-zeros, take the next instruction.

INSTRUCTION: TIO (Transfer when Indicators On) Octal code: +0042
FORMAT: (Type B)

OP. CODE	IA		TAG	Y
S,1	11 12-13		18 -20	21 35

DESCRIPTION: For each bit in the c(AC) that is a one, the corresponding bit of the SI Register is examined. If all the positions examined in the SI Register are ones, the computer takes its next instruction from location Y. Otherwise the computer takes the next instruction in sequence.

INSTRUCTION: TIF (Transfer when Indicators Off) Octal code: +0046
FORMAT: (Type B)

OP.	IA		TAG	Y
S,1	11 12-13		18-20	21 35

DESCRIPTION: The ones in the AC are compared with corresponding zeros in the SI Register. If all ones match zeros, the computer takes its next instruction from location Y. Otherwise the computer takes the next instruction in sequence. In all of the above instructions, the contents of both registers being examined remain unchanged.

EXAMPLES:

1. Pick up a location called TOTAL. If bit 35 is "one," go to SUBR1 and if bit 35 is zero, go to SUBR2. Place the word in location TOTAL into the AC before going to the subroutine.

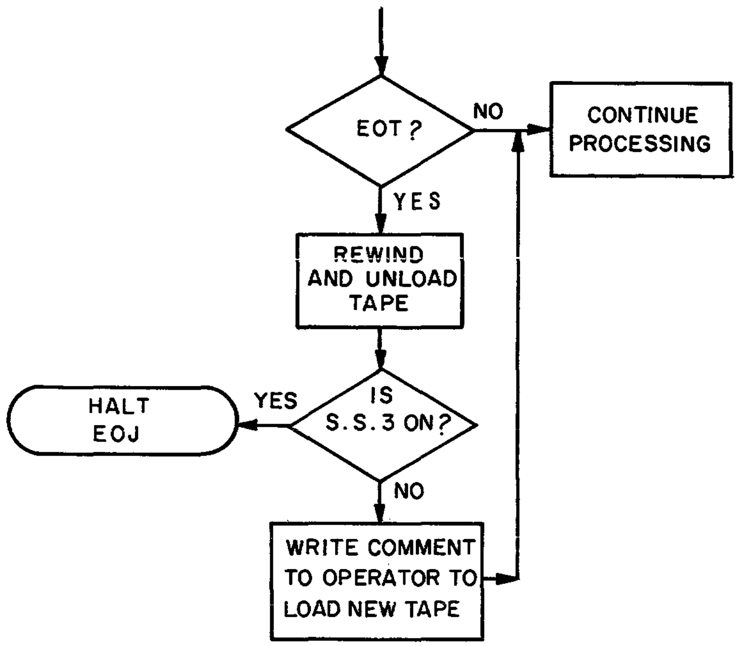
OP	VARIABLE FIELD	REMARKS
LDI	TOTAL	Place TOTAL into Indicator
PIA		Also place it into AC
ONT	=1B35	If bit 35 of Indicator is on, skip one instruction
TRA	SUBR2	If bit 35 is off (zero) go to SUBR2
TRA	SUBR1	Bit 35 was on (one), go to SUBR1

2. Assume that there is a "flag" word already in AC. If bit 31 is on, go to HOLD; if bit 1 is on, go to STAND and if bit P is off, go to STOP.

OP	VARIABLE FIELD	REMARKS
PAI		Place c(AC) into Indicator
CAL	=1B31	Pick up proper bit for compare
TIO	HOLD	If bit 31 is on, go to HOLD
CAL	=1B1	Bit 31 was off, pick up next bit for compare purposes
TIO	STAND	If bit 1 is on, go to STAND
CAL	= -0	This puts a minus sign in position P (see page 138)
TIF	STOP	If P bit is off, go to STOP

EXAMPLES

3. Assume that an end-of-tape has been reached in writing output on Channel A, tape 7 (A7). Check Sense Switch 3. If it is on (in down position), this is the end-of-job (EOJ). If it is off (in up position), go to another tape to continue writing.



LOC	OP	VARIABLE FIELD	REMARKS
	ETTA		Is it end-of-tape?
	TRA	ET	Yes, go to ET routine
	TRA	CONT	No, go on processing
ET	RUNA	7	Rewind and unload tape 7
	SWT	3	Test S.S.3 for end-of-file
	TRA	COMMNT	No (switch is up) go to write COMMNT
	TRA	EOJ	Yes (switch is down), go to EOJ

Since this is only a tiny portion of a program to show use of SWT, CONT, COMMNT, and EOJ are not defined.

SENSE LIGHTS: There are four Sense Lights, designated by (Y), where Y represents lights 1, 2, 3 or 4 correspondingly defined by positions 97, 98, 99 and 100. In the use of Sense Lights, a particular condition will not automatically turn on a sense light. When the programmer has determined that a particular condition exists, he must use an instruction to turn a sense light on or off to be used as an indication of the existence of that condition.

PSEUDO OP: SLN (Sense Light On) (Y)
DESCRIPTION: This pseudo instruction turns on the Sense Light designated by (Y).

PSEUDO OP: SLF (Sense Lights Off)
DESCRIPTION: This pseudo instruction turns off all Sense Lights.

PSEUDO OP: SLT (Test Sense Light) (Y)
DESCRIPTION: This pseudo instruction tests whether Sense Light (Y) is on or off. If Sense Light (Y) is on, it is turned off and the computer skips one instruction. If the light is off, the computer takes the next instruction in sequence.

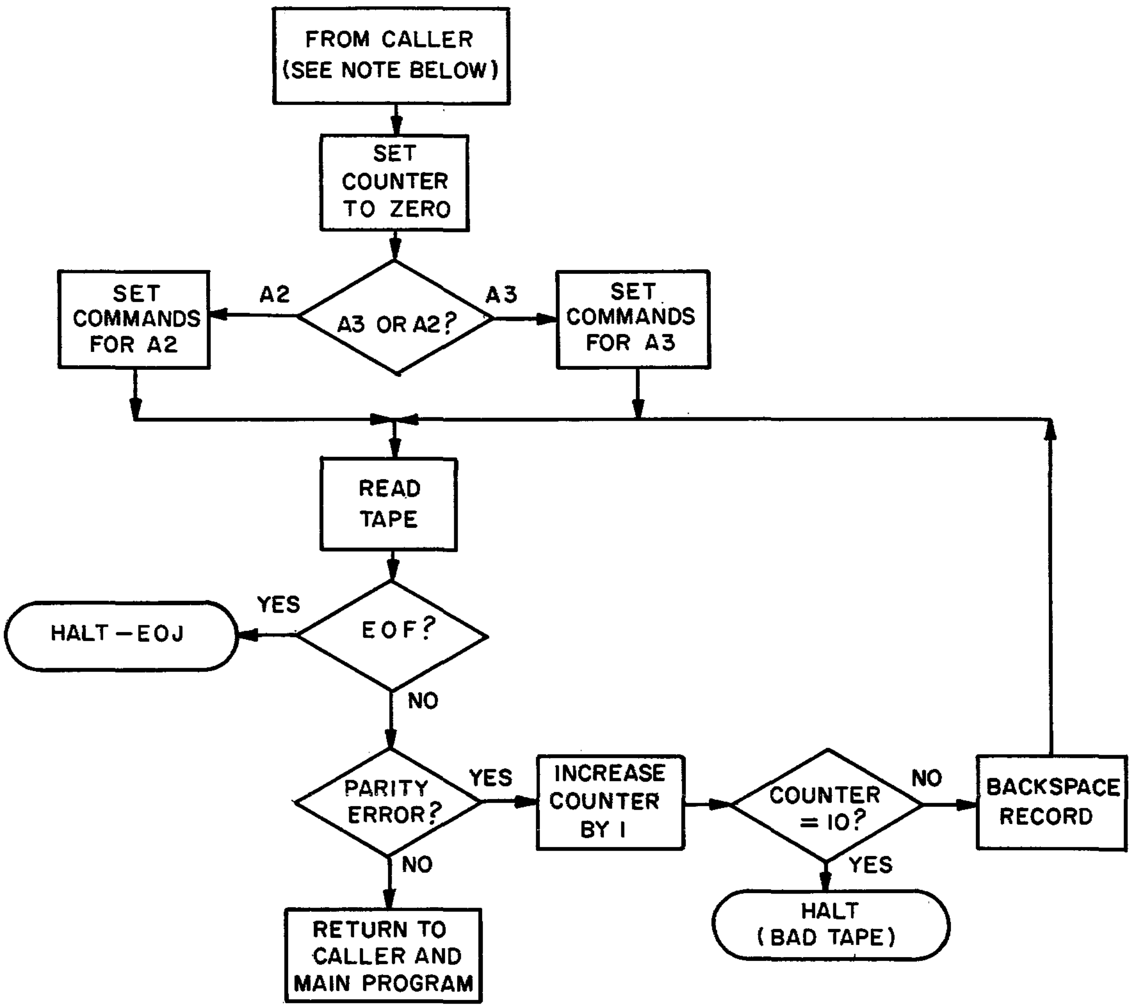
EXAMPLE:

To understand the use of Sense Lights, let us assume that we have a program with the following peculiarities: During certain computations, a number will be in the AC whose sign is to indicate whether SUBR1 or SUBR2 is to be followed at a later time in the program. If the sign is plus, SUBR1 is to be entered and if the sign is minus, SUBR2 is to be entered. Unfortunately, between the time that the indicator appears in the AC and the time the decision is to be made, other operations occur using the AC, so that the indicator is destroyed. This type of problem may be solved with the use of Sense Lights as shown below:

LOC	OP	VARIABLE FIELD	REMARKS
	SLF		Sense Lights turned off at beginning of program
	___		
	___		
	___		
	___		
	SLT	1	Turn off Sense Light 1
	NOP		Skip one instruc.
	___		
	___		
	___		
	___		
	___		
	___		
	TPL	JUMP	If sign +, go to JUMP, skipping one instruction
	SLN	1	Sign was -, turn on Sense Light 1 to indicate SUBR2 will be entered
JUMP			Further computations
	___		
	___		
	___		
	___		
	___		
	___		
	SLT	1	Test Sense Light 1
	TRA	SUBR1	If light is off, go to SUBR1
SUBR2			If light on, go to SUBR2
	___		
	___		
	___		
	___		
SUBR1			
	___		
	___		
	___		
	___		

EXAMPLE:

Write a subroutine to read from A3. However, if Sense Light 4 is on, reading will be from A2. Records are 22 words in length from both tapes and will be placed into location HOLD, HOLD + 1, etc. The Sense Light will be turned on, if necessary, each time a record is processed and the subroutine is called again. When EOF is reached, the job is finished.



Note: The instruction in the Main Program that leads to a subroutine is named the caller. When the subroutine is finished, control is returned to the Main Program. Each time the caller is encountered in the Program, the subroutine is entered (see page 133 for subroutine linkage).

EXAMPLE--continued

LOC	OP	VARIABLE FIELD	REMARKS
X	TAPENO	A3B	Define X as Chan. A, tape 3, Binary
Y	TAPENO	A2B	Define Y as Chan. A, tape 2, Binary
READ	STZ	CNT	Set counter to zero
	CLA	A2	Move A2 (defined below) into AC
	SLT	4	Test Sense Light 4
	CLA	A3	Light off, move A3 (defined below) into AC
	STA	RDS	Preset to read either A3 or A2
	STA	BSR	Preset to backspace either A3 or A2 (Note Octal codes under A2 and A3 below)
	TCOX	*	Wait until through
RDS (Read Routine)	RTDX		Preset to A2 or A3
	RCHX	IO	
	TCOX	*	
	TEFX	EOJ	
	TRCX	PEX	
	TRA	1, 4	Return to caller
EOJ	HTR	*	Halt - end of job.
PEX (parity error routine)	CLA	CNT	
	ADD	= 1	
	STO	CNT	
	SUB	= 10	
	TZE	EOJ	Unreadable record (if zero)
BSR	BSRX		Preset to A2 or A3
	TRA	RDS	
IO	IORT	HOLD, , 22	Read 22 words
HOLD	BSS	22	Assign 22 locations to HOLD
A2	RTDY		In Octal: 076600001222
A3	RTDX		In Octal: 076600001223 (RDS = 076200001222) (BSR = 076400001223)
CNT	PZE		

Note: The last 4 Octal characters of location RDS and BSR specify the channel, mode and tape unit. Thus the STORE address (STA) does not change the operation, only the Input/Output device.

INDIRECT ADDRESSING: (IA) A brief definition of Indirect Addressing was given on page 21. Please review it before continuing on this page. Any instruction that is Indirectly Addressable may be used to set the IA flag (one bits in positions 11-12). A list of these codes may be found in Appendix E of the IBM 7090 Reference Manual.

An instruction that is indirectly addressed calculates the presumptive location, goes to this location and gets its actual location from the effective location of the new instructions (effective location defined on page 93). This statement is rather difficult to follow. It will be further explained by a number of examples.

An asterisk (*) placed directly after the Op. Code indicates that this instruction is indirectly addressed. This is a very powerful and useful programming tool and should be studied very carefully.

EXAMPLES:

1.

LOC	OP	VARIABLE	REMARKS
	CLA *	HOLD	Presumptive loc. of CLA is HOLD. Program goes to HOLD and finds the effective loc. to be AREA. This is then moved into the AC and the STO operation is ignored.
	___		
	___		
HOLD	STO	AREA	

Without the asterisk (*), the contents of location HOLD would be placed into the AC. Since it is indirectly addressed, the contents of location AREA will be placed into the AC.

2.

LOC	OP	VARIABLE	REMARKS
	AXT	1, 1	This moves 1 into XR1
	STO *	HOLD, 1	Indirectly addressed, modified by XR1 takes us to AREA, since it says in effect, HOLD-1.
	CLA	AREA	If XR1 is less than 1 (zero) the effective
HOLD	CLA	BLOCK	address would be BLOCK.

Using an Index Register complicates the problem, but makes IA more powerful as a tool. In this case, the indirectly addressed STO instruction will place the contents of the AC into location AREA, unless XR1 is zero, in which case the contents of the AC would be placed into location BLOCK.

EXAMPLE:

3.

LOC	OP	VARIABLE	REMARKS
	AXT	1, 2	Place 1 into XR2
	AXT	-1, 1	Place -1 into XR1
	STZ *	SAVE, 1	(go to save + 1, and store zeros in LAP -1 (AB PZE)
SAVE	CLA*	MAP, 2	
	CLA*	LAP, 2	
AB	PZE		
LAP	PZE		
CD	PZE		
MAP	PZE		

The STZ (indirectly addressed) will store zeros into LAP -1, unless XR1 is zero, in which case it will store zeros into MAP -1. If XR2 is zero, then zeros would be stored in LAP.

The indirect addressing on SAVE and SAVE + 1 has no effect on the STZ instruction.

The lines in the example above show the steps taken by the computer to determine just where zeros are to be stored in the case where XR1 is -1. If XR1 and XR2 are both zero, it would be as follows:

	AXT	0, 2	
	AXT	0, 1	
	STZ *	SAVE, 1	zeros are stored in loc. MAP
SAVE	CLA*	MAP, 2	
	CLA*	LAP, 2	
AB	PZE		
MAP	PZE		

EXAMPLES:

4. Assume that we want a subroutine that will calculate 3* X + 4 (floating point). The address of X is in the AC (positions 21-35) upon entrance to the subroutine. The answer will be left in the AC upon exit from the subroutine.

	LOC	OP	VARIABLE	REMARKS
(caller will be:)		TSX	CALC, 4	This is in the Main Program

Subroutine

LOC	OP	VARIABLE	REMARKS
CALC	STA	CLA	Address of X into CLA instruction
	CLA	**	Pick up X into AC. Preset to loc. X
	FAD*	CLA	Calculate 2X
	FAD*	CLA	Calculate 3X
	FAD	= 4.	Calculate 3X + 4
	TRA	1, 4	Return to "caller," end of subroutine

5. In the problem above, if the "caller" was changed to:

TSX	CALC, 4
PZE	X

Subroutine

LOC	OP	VARIABLE	REMARKS
CALC	CLA*	1, 4	Bring X into AC
	FAD*	1, 4	Calculate 2X
	FAD*	1, 4	Calculate 3X
	FAD	= 4.	Calculate 3X + 4
	TRA	2, 4	Return to caller

PROBLEM:

115. Write a subroutine with arguments A, B, and C respectively. In the subroutine, calculate (in floating point) (2 * A + B) /C and return to Main Program with the answer in the AC (solve with Indirect Addressing).

Assume the caller looks like this:

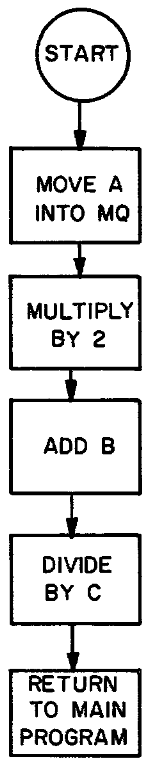
LOC	OP	VARIABLE FIELD	REMARKS
	TSX	CALC, 4	
	TSX	A	
	TSX	B	
	TSX	C	
		return	

SUBROUTINE:

LOC	OP	VARIABLE FIELD	REMARKS
CALC			

CORRECT ANSWER

PROBLEM 115:



LOC	OP	VARIABLE FIELD	REMARKS
CALC	LDQ*	1, 4	Pick up A in MQ
	FMP	= 2.	Multiply by literal of 2
	FAD*	2, 4	2 * A + B
	FDH*	3, 4	(2 * A + B) /C
	XCA		Move quotient to AC
	TRA	4, 4	Return to program

LESSON 13

GENERAL CONSIDERATIONS:

1. Always start the program by rewinding the tapes that are to be used.

2. If temporary storage areas are to be used, always clear them out at the beginning of the program with STZ instructions to make certain that the areas contain nothing but zeros. Never assume that anything is zero initially.

3. Consider a large program as a series of subroutines. This gives you the advantage of being able to check out one routine at a time.

4. Always flow chart the problem before attempting to code it. This is the easiest way to catch logic errors and it simplifies the problems of coding, debugging and modifying programs.

5. Take maximum advantage of Input/Output. Use prepared programs if available.

6. Always check for End-of-File (when reading) and End-of-Tape (when writing).

7. If buffer areas are to be used, be sure that they are large enough.

8. If the program is long enough to run over 10 minutes on the 7090, it should have restart capability. In this way, if there is trouble in running the program, it isn't necessary to go back to the beginning and start over. Can go to the nearest restart point.

9. Do only what is essential on-line. All possible outputs should be off-line. Stick to tape input and tape output on-line.

10. If Sense Lights are to be used in the program, turn them off at the beginning.

11. Use as many system checks as possible:

    a. Keep record count of number of records in storage
    b. Keep control total if possible
    c. Keep limit checks (compare to a limit which is not to be exceeded)
    d. Keep tape labeling checks if tapes are to be mounted in sequence.

12. Use messages to the operator where it will help to make things clear to him in running your program.

TRAPPING: Floating point traps were discussed briefly on page 51. Another form of trapping is called Transfer trapping. There are special instructions to enter and to leave the Transfer Trapping Mode of operation. These instructions are shown on page 163.

When the computer is operating in the trapping mode, control is transferred to location 0001, whenever the conditions for transfer have been met.

EXAMPLE: TZE (Transfer on Zero)

Normally, if the AC = zero, transfer to instruction contained in loc. Y. Otherwise computer takes the next instruction in sequence.

In the Trapping Mode, if the AC = zero, the computer transfers to location 0001 for its next instruction.

EXAMPLE 2: TRA (Transfer)

This is an unconditional transfer, therefore the condition for transfer is always met and control is always transferred to location 0001 in the Trapping Mode.

Whenever the condition for transfer is not met, the instruction is executed in the normal manner.

The major use of the Transfer Trap Mode is in checking out a program. When operating in this Mode, the location of every transfer instruction (with the exception of trap transfer instructions) replaces the address part of location 0000. This occurs whether the condition for transfer is met or not.

A special trap trace program may be written, starting in location 0001, which will write out on a special tape, all transfer instructions for subsequent off-line printing. At the end of the trace program, control is returned to the main program which will continue until another transfer instruction returns it to the trace program.

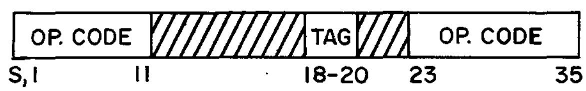
When the information accumulated by the trap trace program is printed out, it will give the programmer a record of the contents of various registers at each transfer instruction, providing the conditions for transfer were met. This can be extremely useful information to a programmer in checking a program which is not functioning properly.

When the program has been debugged (corrected), the Enter Trapping Mode instruction may be replaced by a NOP instruction, cutting off the entire trace program.

<u>EXAMPLE</u>: The trap trace program could store the following information, beginning with a location designated TRAP.

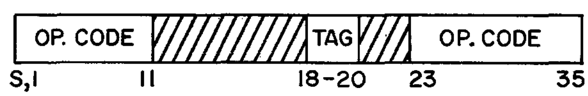
TRAP	c(AC) positions S, 1-35
TRAP + 1	c(AC) positions P and Q in bit positions 34 and 35
TRAP + 2	c(MQ)
TRAP + 3	c(XR1) in the decrement part of the word
TRAP + 4	c(XR2) in the decrement part of the word
TRAP + 5	c(XR4) in the decrement part of the word

<u>INSTRUCTION</u>: ETM (Enter Trapping Mode) Octal code: +0760 0007
<u>FORMAT</u>: (Type E)



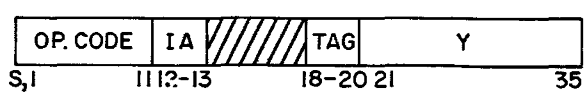
<u>DESCRIPTION</u>: This instruction causes the computer to enter the transfer trapping mode. It turns on the trapping indicator and the trap light on the operators console. The computer will continue to operate in the trapping mode until a "leave trapping mode" instruction is executed or until the "clear" or "reset" keys are pressed on the operators console.

<u>INSTRUCTION</u>: LTM (Leave Trapping Mode) Octal code: -0760 0007
<u>FORMAT</u>: (Type E)



<u>DESCRIPTION</u>: This instruction turns off the trap indicator and the trap light on the operators console. Another ETM instruction would be required to put the program back into the trapping mode.

<u>INSTRUCTION</u>: TTR (Trap Transfer) Octal code: +0021
<u>FORMAT</u>: (Type B)



<u>DESCRIPTION</u>: This instruction causes the computer to take its next instruction from location Y. This makes it possible to have an ordinary transfer even when in the trapping mode. This is the only transfer instruction that will not cause control to be transferred to location 0001, when the conditions for transfer have been met and the machine is in the trapping mode.

<u>SORTING</u>: This term refers to the procedure of arranging data according to certain specified characteristics. For example; a group of numbers may be sorted in such a way that the smallest number comes first, followed sequentially by the next largest number, until all numbers are in order from smallest to largest.

Sorting on the 7090 is quite difficult. Fortunately, most organizations have "Sort Routines" already developed and the programmer merely has to use the applicable routine if he desires to do a sort in the program. Sorting is a slow process, taking up considerable machine time.

On the following pages, an example of sorting is shown. This is not the fastest or best way, but it is fairly simple to understand.

The problem is stated on page 165. What the program does is:

1. Finds the smallest number by searching through the entire 100 numbers and places it into the first position of a temporary location.

2. Although it has been moved to a temporary location, it is still present in the original 100 numbers, so the numbers are shifted so that all "words" above it move down one, covering up the one we have moved.

3. This process continues until all of the numbers have been moved to the Temporary area. Now they are in numerical sequence.

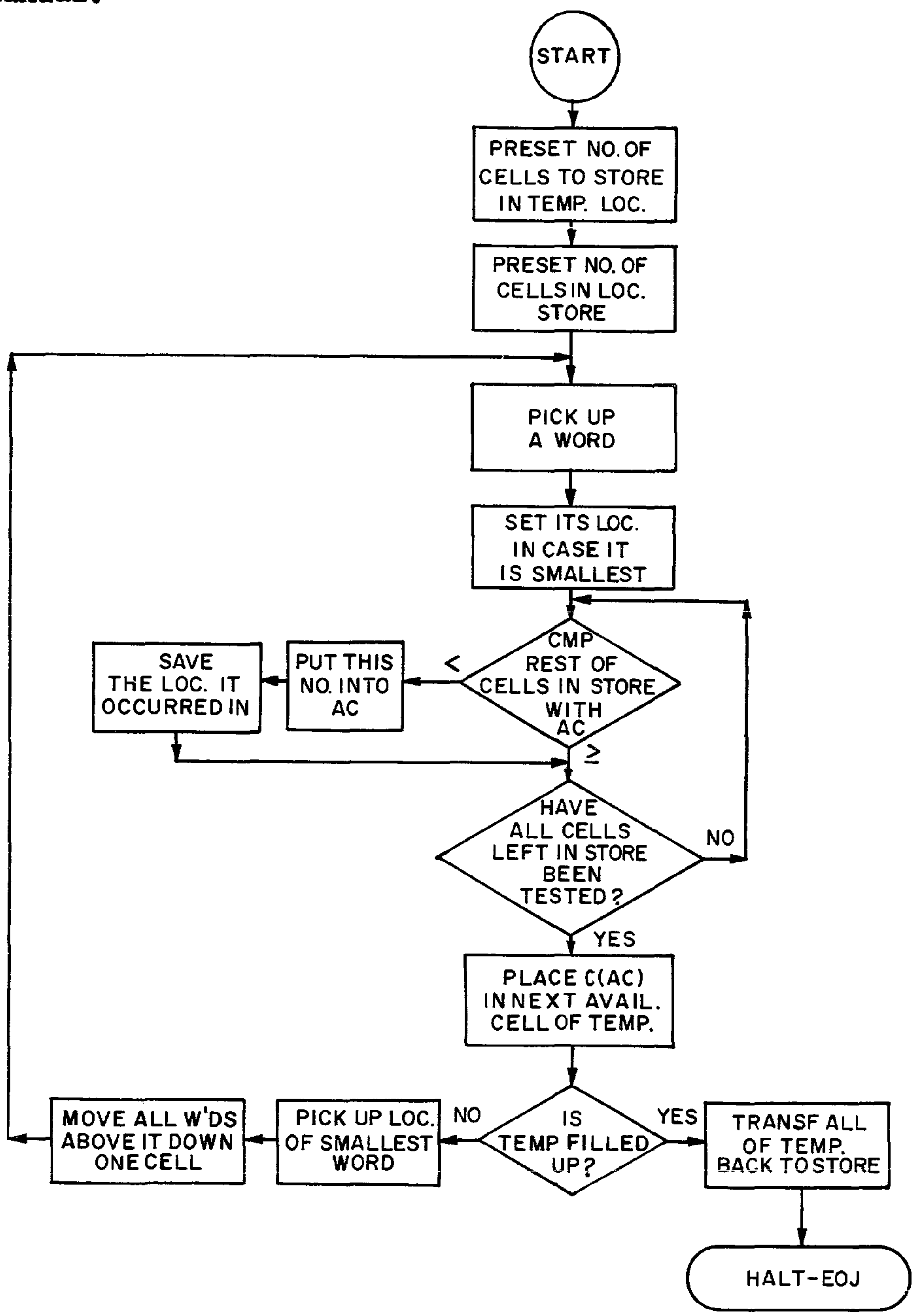
4. The entire 100 numbers are then moved back to the original area as required by the problem.

5. The job is done.

Note: The term <u>cells</u> is used both in the flow chart and the program on the next two pages. This term is used to indicate a machine <u>word</u> and it is more commonly used among programmers than the term <u>machine word</u>.

EXAMPLE:

PROBLEM: One hundred numbers are stored consecutively beginning in location STORE. Sort the numbers (put them in sequential order) from smallest to largest, leaving them in the original block of locations. There are two instructions in the following program that have not been previously defined (SXA and TNX). Look them up in the 7090 reference manual.



LOC	OP	VARIABLE FIELD	REMARKS
START	AXT	100, 1	Preset no. of cells to store into TEMP
	AXT	100, 2	Preset the no. of words in STORE
LOOP	CLA	STORE + 100, 2	Pick up first word left in STORE
	SXA	SX2, 2	Save loc. in case 1st word is smallest
	SXA	NINS, 2	Save the no. left in STORE
CAS	CAS	STORE + 100, 2	Is this STORE no. less?
	TRA	SWITCH	Yes (<), set new 'compare
	TRA	* + 1	No (=), move down to next instruction
TEST	TIX	* - 3, 2, 1	No (>), are we thru? No, compare rest of STORE
	STO	TEMP + 100, 1	Yes, save in TEMP area
	TNX	THRU, 1, 1	Is TEMP area full, yes - go to THRU.
SX2	AXT	**, 2	No, pick up loc. of smallest no.
	CLA	STORE + 99, 2	Move all numbers
	STO	STORE + 100, 2	above, down one
	TXI	* + 1, 2, 1	
	TXL	* - 3, 2, 99	Is XR 2 $\leqslant$ 99? Yes
	LXA	NINS, 2	No, all have been shifted
	TXI	LOOP, 2, -1	Repeat for next smallest no.
SWITCH	SXA	SX2, 2	Save location of smallest no.
	CLA	STORE + 100, 2	
	TRA	TEST	
THRU	AXT	100, 2	
	CLA	TEMP + 100, 2	Transfer back
	STO	STORE + 100, 2	to store
	TIX	* -2, 2, 1	
	HTR*	*	
STORE	BSS	100	
TEMP	BSS	100	
NINS	BSS	1	

PROGRAM TESTING:

On page x, a very brief summary of computer-programmer interaction was given. Now we will touch lightly on each step of the process from coding to final output product. To prepare a program for operational use, the following steps must be observed after the problem has been analyzed, flow charted and coded on the appropriate coding sheets.

1. The coding sheets must be sent to the keypunch organization. It is important to request that the cards be interpreted (this means that whatever is punched in a card will be printed across the top of it). Each line of the coding sheet will become a punched card.

2. When the cards come back from Keypunch, they must be compared with the coding sheets. The comparison must be extremely careful and detailed, digit for digit. Any card containing errors must be destroyed and replaced with a corrected card.

3. The deck of cards you now have is called the source program. The source program is sent to Machine Operations organization for assembly. The Fortran Assembly Program operates on the source program, changing the symbolic source program into language that is understandable to the computer. This is accomplished automatically by the computer. You request an Assembly Print-Out when submitting the source program for assembly. This allows you to make a final check of the program and the print-out will show the locations in storage of constants and assigned work areas. The assembled program is called the object program.

4. Before the object program may be run against live data, it must be debugged (freed of all possible errors). The best and least expensive way of doing this is by running the program against Test Data. Test data is written by the programmer to attempt to simulate operational data and to attempt to cover each different action taken by the program. Since the programmer is writing the test data, he can easily determine what the results should be after the data is run through the machine. In this manner, he can check out his program before it is allowed to work on operational data. The test data must also be key punched and desk checked.

5. The object program card deck and test card deck are sent to Machine Operations for a test run. Again, a print-out of the result is requested. The two card decks are transferred (off-line-not on the main computer) to tape, loaded into the 7090 and the program execution is begun.

6. If the program processes the test deck all the way through, it is still necessary to check the print-out to make sure that the results obtained are as expected.

7. If the computer hangs up (stops before processing is finished), a memory print will automatically be furnished by the operator to give the programmer an idea of where the trouble occurred so that he may try to find and correct the error.

8. If correction is to be made, the corrected card (or cards) must be put into the original card deck of the source program, replacing the cards that were in error. The program must then be reassembled before attempting to run again. It is possible to patch a program in such a way that reassembly is avoided, but patching will not be elaborated upon here.

9. After corrections have been made, another test run is attempted and this process is continued until the program is clean (no more errors apparent). A new program almost never runs through without errors. A programmer always expects a few ineffective runs before he can clean up his program, but the important thing is to work as carefully as possible to avoid foolish clerical errors.

10. It is also extremely important to avoid errors in basic logic when the program is in the planning stage. Careful flow charting and anticipating all contingencies in advance help to make for better programs. A good flow chart also helps others to understand the workings of your program and greatly assists you if modifications or corrections are required.

11. When the output product is to be in printed form, it must be remembered that Binary words written on tape by the 7090, are not intelligible. Each installation has several good subroutines for converting Binary numbers to Decimal characters. These subroutines should be used and the process should be accomplished off line whenever possible.

PROBLEM:

116. Write a routine to compute the following expression:

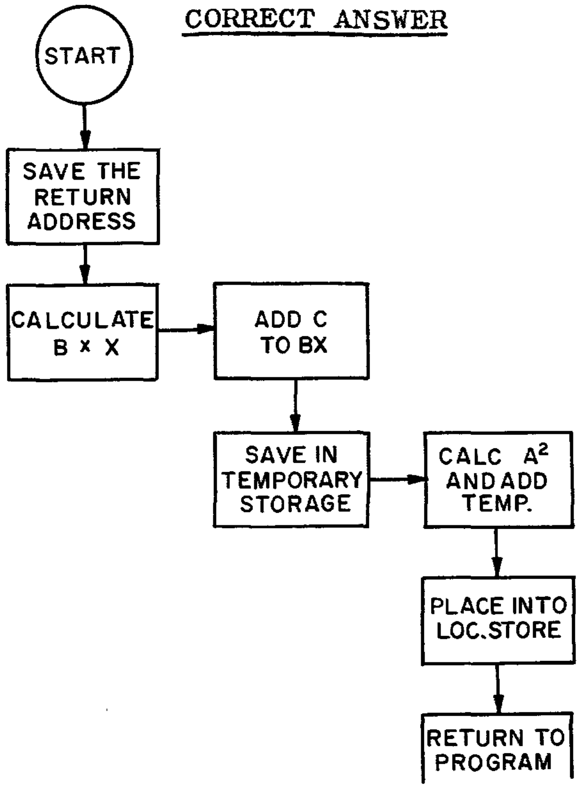
$A^2 + BX + C$ (floating point numbers)

Where A, B, and C are stored consecutively beginning in location HOLD and X is in the MQ. The Decrement part of the AC contains the address to which the routine will transfer upon completion of the problem.

LOC	OP	VARIABLE	REMARKS

CORRECT ANSWER

PROBLEM 116:



LOC	OP	VARIABLE	REMARKS
ENTRY	ARS	18	Save return address by
	STA	RTN	storing it in RTN
	FMP	B	Calculate BX
	FAD	C	Calculate BX + C
	STO	TEMP	Save in temporary storage
	LDQ	A	Calculate A^2
	FMP	A	
	ADD	TEMP	Calc. A^2 + BX + C
	STO	STORE	Place into loc. STORE
RTN	TRA	**	Return to Main program
A	DEC	1.95	
B	DEC	3.84	Constants (values chosen
C	DEC	.98E-4	at random since none were
			given in problem)
TEMP	BSS	1	Allocate storage positions
STORE	BSS	1	to Temp. storage and the
			answer

PROBLEM:

117. Read a 5 word record from Channel A, unit 6, in Binary. Place in location HOLD, HOLD + 1, etc. Solve the following equation using HOLD as A, HOLD + 1 as B and HOLD + 2 as C.

$$R = 2\left(1 - \frac{A^2 + B^2}{C^2}\right) \text{ floating point numbers}$$

Assume that there will be no overflow or underflow and that the result of each arithmetic operation may be contained within one register. The answer (R) will be placed into loc. COMP. Write the result on Channel C, unit 5, Binary, as a 5 word record:

1st word = A, 2nd word = B, 3rd word = C, 4th = R, 5th = C^2. Stop at EOF or EOT.

Remember, that the Read and Write routines must be as complete as they are in lesson 10. Flow chart on this page and code on the next two pages.

FLOW CHART

PROGRAM

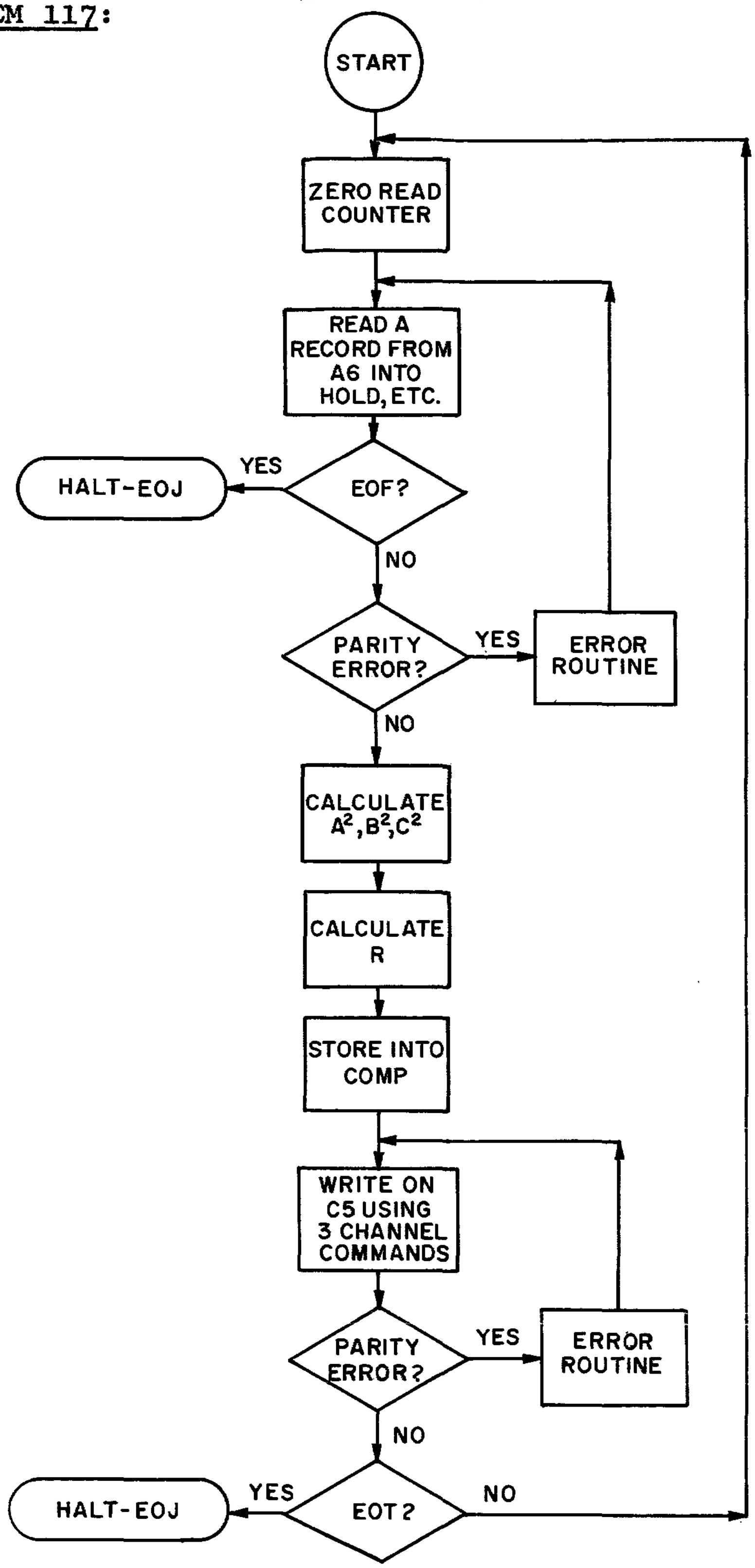
LOC	OP	VARIABLE	REMARKS

PROGRAM--continued

LOC	OP	VARIABLE	REMARKS

PROBLEM 117:

CORRECT ANSWER



CORRECT ANSWER

PROGRAM

LOC	OP	VARIABLE	REMARKS
X	TAPENO	A6B	
Y	TAPENO	C5B	
START	STZ	CNT	Preset counter for Read
RX	RTBX		
(Read	RCHX	IOIN	Read tape
Routine)	TCOX	*	
	TEFX	EOJ	End of file?
	TRCX	PEX	Parity error?
(calcu-	LDQ	HOLD	
lation)	FMP	HOLD	Calculate A^2
	STO	A2	
	LDQ	HOLD + 1	Calculate B^2
	FMP	HOLD + 1	
	STO	B2	
	LDQ	HOLD + 2	Calculate C^2
	FMP	HOLD + 2	
	STO	C2	
	CLS	A2	
	FSB	B2	Calculate $-A^2 - B^2$
	FDH	C2	Calculate $-(A^2 + B^2) / C^2$
	XCA		
	FAD	=1.	
	XCA		
	FMP	=2.	Cal. $2\ (1 - \frac{(A^2 + B^2)}{C^2})$
	STO	COMP	
(Write	STZ	CNT	Preset counter for Write
Routine)	TCOY	*	
AGAIN	WTBY		
	RCHY	IOOUT	Write a record
	TCOY	*	
	TRCY	PEY	Parity error?
	ETTY		End of tape?
	TRA	STOP	
	TRA	START	

PROGRAM--continued:

```
                LOC     OP      VARIABLE        REMARKS
                ------------------------------------------------------------
                EOJ     HTR     *
                ------------------------------------------------------------
                STOP    EQU     EOJ
                ------------------------------------------------------------
                PEX     CLA     CNT
(Error                  ADD     = 1
Routine                 STO     CNT
for Read)               SUB     =10
                        TZE     EOJ             Bad tape, if yes-go to EOJ
                        BSRX                    If no, backspace
                        TRA     RX              Back to try to read again
                ------------------------------------------------------------
                PEY     CLA     CNT
                        ADD     = 1
(Error                  STO     CNT
Routine                 SUB     = 5
for                     TZE     STOP            Bad tape, if yes - go to
Write)                                          STOP
                        BSRY                    If no, backspace
                        WTBY
                        TCOY    *
                        TRCY    *
                        TRA     AGAIN           Back to try writing again
                ------------------------------------------------------------
                IOOUT   IOCP    HOLD,  , 3      Output A, B and C
                        IOCP    COMP, , 1       Output R in same record
                        IOCD    C2,   ,  1      Output C2 in same record
                                                and stop
                IOIN    IORT    HOLD,  , 5      Input Command
                ------------------------------------------------------------
                CNT     BSS     1
                HOLD    BSS     5
                A2      BSS     1               Allocate storage locations.
                B2      BSS     1
                C2      BSS     1
                        END
                ------------------------------------------------------------
```

LESSON 14

QUICK REFERENCE

INSTRUCTIONS AND THEIR MEANINGS

Refer to Page			
	1. MISCELLANEOUS INSTRUCTIONS:		
47	XCA	(+0131)	Exchange AC and MQ - Reverses the two fields
30	HTR	(+0000)	Halt and Transfer - Halts Program, if restart, goes to Y
77	NOP	(+0761)	No operation - Program continues to next instruction
	2. FIXED POINT ARITHMETIC INSTRUCTIONS:		
27	ADD	(+0400)	ADD - Add Y to AC
29	SUB	(+0402)	SUBTRACT - Subtract Y from AC
29	MPY	(+0200)	MULTIPLY - Multiply Y by MQ, Product in AC (and MQ if needed)
69	RND	(+0760 0010)	ROUND - Increase AC by Binary 1 if posit. 1 of MQ contains 1.
29	DVH	(+0220)	DIVIDE OR HALT - AC and MQ are Dividend, Y is Divisor, Quotient in MQ, Remainder in AC. If can't divide, HALT.
69	DVP	(+0221)	DIVIDE OR PROCEED - as above, except that if can't divide, continue with program with Div. Check Light on.
69	DCT	(+0760 0012)	DIVIDE CHECK TEST - if indicator on, takes next instruction. If indicator off, skips one instruction.
	3. FLOATING POINT ARITHMETIC INSTRUCTIONS:		
46	FAD	(+0300)	FLOATING ADD - Add Y to AC
46	FSB	(+0302)	FLOATING SUBTRACT - Subtract Y from AC
46	FMP	(+0260)	FLOATING MULTIPLY - Multiply Y by MQ
46	FDH	(+0240)	FLOATING DIVIDE OR HALT - AC divided by Y. Quotient in MQ, remainder in AC. If can't divide HALT.
	4. SHIFTING INSTRUCTIONS:		
47	ALS	(+0767)	AC LEFT SHIFT - The AC shift left No. posit. in Y 28-35
47	ARS	(+0071)	AC RIGHT SHIFT - As above, only shift to the right.
73	LLS	(+0763)	LONG LEFT SHIFT - AC and MQ as one register. Shifted left no. places specified in Y 28-35.
73	LRS	(+0765)	LONG RIGHT SHIFT - As above, only shift to the right.

Refer to Page	5. <u>STORE AND LOAD INSTRUCTIONS</u>:	
27	CLA (+0500)	CLEAR AND ADD - Move Y into AC
29	STO (+0601)	STORE - Move AC into Y
30	LDQ (+0560)	LOAD MQ REGISTER - Move Y into MQ
30	STQ (-0600)	STORE FROM MQ REGISTER - Move MQ to Y
73	STZ (+0600)	STORE ZEROS - Move zeros into Y, Sign to +
89	STA (+0621)	STORE ADDRESS - from AC_{21-25} to Y_{21-35}
89	STD (+0622)	STORE DECREMENT - from AC_{3-17} to Y_{3-17}
89	STT (+0625)	STORE TAG - from AC_{18-20} to Y_{18-20}
89	STP (+0630)	STORE PREFIX - from $AC_{S,\ 1,\ 2}$ to $Y_{S,\ 1,\ 2}$

6. <u>TRANSFER INSTRUCTIONS (No Index)</u>:

73	TRA (+0020)	TRANSFER - Transfer to instruction specified by Y
31	TZE (+0100)	TRANSFER ON ZERO - If AC = Zero transf. to Y, otherwise to next instr.
31	TOV (+0140)	TRANSFER ON OVERFLOW - If AC overflow indicator on, transfer to Y, otherwise on to next instruction
47	TPL (+0120)	TRANSFER ON PLUS - If sign of AC +, transf. to Y, otherwise to next instr.
47	TMI (-0120)	TRANSFER ON MINUS - If sign of AC -, transf. to Y, otherwise to next instr.
77	CAS (+0340)	COMPARE AC WITH Y - if c(AC) > c(Y) go to next instr. If = skip one instr, if <, skip two instructions
52	NZT (-0520)	STORAGE NOT ZERO TEST - If c(Y) are not 0, skip 1 instr. If c(Y) are 0, on to next instruction
53	ZET (+0520)	STORAGE ZERO TEST - This is the opposite of NZT instruction

7. <u>TRANSFER INSTRUCTIONS (INDEX)</u>

95	TIX (+2000)	TRANSFER ON INDEX - If c(XR) > Decr. XR reduced by Decr. and on to Y. Otherwise on to next instruction
95	TXI (+1000)	TRANSFER WITH INDEX INCREMENTED-Adds Decr. to XR and on to Y
95	TXL (-3000)	TRANS. ON INDEX LOW OR EQUAL-If c(XR) < or = Decr. go to Y. Otherwise on to next instruction
95	TXH (+3000)	TRANS. ON INDEX HIGH-If c(XR) > Decr., go to Y. Otherwise on to next instr.
95	TSX (+0074)	TRANS. AND SET INDEX-Place 2's Compl. of Instr. CTR into XR. Next instr. from loc. Y.

Page		
	8. INDEXING INSTRUCTIONS:	
94	LXA (+0534)	LOAD INDEX FROM ADDRESS - $c(Y)_{21-35}$ Moves into specified XR
94	LXD (-0534)	LOAD INDEX FROM DECREMENT - $c(Y)_{3-17}$ Moves into specified XR
94	AXT (+0774)	ADDRESS TO INDEX TRUE - Positions $_{21-35}$ of this instruction moves into specified XR.
	9. INPUT/OUTPUT INSTRUCTION AND COMMANDS:	
120	RTD (+0762)	READ TAPE DECIMAL - Will select tape to be read from if followed by RCH
120	WTD (+0766)	WRITE TAPE DECIMAL - With the RCH will select tape to write, otherwise writes blank
120	BSR (+0764)	BACKSPACE RECORD - Backspace 1 record
120	WEF (+0770)	WRITE END-OF-FILE - Writes EOF gap and tape mark
120	REW (+0772)	REWIND - Tape rewinds to load point. Ready to run again.
120	RUN (-0772)	REWIND AND UNLOAD - Rewinds tape to load point and unloads it
120	TCO $(+^{CH.A}_{0060})$	TRANS. IF CHAN. IN OP. - If Channel in operation, takes next instruction from loc. Y
121	TRC $(+^{CH.A}_{0022})$	TRANS. ON REDUNDANCY - If parity indicator on, turned off and next instruction from loc. Y
121	TEF $(+^{CH.A}_{0030})$	TRANS. ON END-OF-FILE - If EOF indicator on, turned off and next instruction from loc. Y
121	BTT (+0760)	BEGINNING-OF-TAPE TEST - If indicator on, turned off and takes next instr. in sequence. If off, jumps one instr.
121	ETT (-0760)	END-OF-TAPE TEST - As for BTT, except tests EOT indicator
121	RCH $(+^{CH.A}_{0540})$	RESET AND LOAD CHAN. - Used with the Read or Write instr. to specify First Data Channel Command
121	IOCD-I/O UNDER COUNT CNTRL, DISCON.	- Reads or Writes the number of words specified in Decrement
122	IORT - I/O OF RECORD, TRANS.	Reads to end of record or until word count to zero. Writes number of words specified in Decrement. Next command from LCH or else disconnects.
	10. LOGICAL INSTRUCTIONS:	
138	CAL (-0500)	CLEAR AND ADD LOGICAL WD. - As the CLA except sign goes into posit. P
138	SLW (+0602)	STORE LOGICAL WORD - As the STO except that bit from P goes into Sign position.

Page 10. LOGICAL INSTRUCTIONS--continued

138 ANA (-0320) AND TO ACCUMULATOR - Bits are matched, using AND rules, result into AC

138 ANS (+0320) AND TO STORAGE - As for ANA, except result into storage loc. Y

139 ORA (-0501) OR TO ACCUMULATOR - Bits are matched, using OR rules, result into AC

139 ORS (-0602) OR TO STORAGE - As for ORA, except result into storage loc. Y

139 ERA (+0322) EXCLUSIVE OR TO AC - As for ORA, except rules for EXCLUSIVE OR apply

142 LGL (-0763) LOGICAL LEFT SHIFT - AC and MQ treated as one. Shifted left no. places in 28-35. Sign of AC no chg.

142 LGR (-0765) LOGICAL RIGHT SHIFT - As for LGL, only shift is to right.

11. SENSE INDICATOR INSTRUCTIONS:

148 PAI (+0044) PLACE AC IN INDICATORS - c(AC)P, 1-35 into Indicator

148 PIA (-0046) PLACE INDICATOR IN AC -Reverse of PAI

148 LDI (+0441) LOAD INDICATORS - c(Y) into Indicator

148 STI (+0604) STORE INDICATORS - Reverse of LDI

149 ONT (+0446) ON TEST FOR INDICATORS - One bits in Y and SI are compared. If =, skip one instr. If not =, takes next instr.

149 OFT (+0444) OFF TEST FOR INDICATORS - As ONT, except SI checked for zeros.

149 TIO (+0042) TRANS. IF INDICATORS ON - One bits in AC and SI are compared. If =, next instruction from loc. Y

149 TIF (+0046) TRANS. IF INDICATORS OFF - Ones in AC compared with zeros in SI. If =, next instruction from loc. Y

12. TRAPPING INSTRUCTIONS:

163 ETM (+0760 0007) ENTER TRAPPING MODE - Causes computer to enter trans. trap. mode

163 LTM (-0760 0007) LEAVE TRAPPING MODE - Turns off trap indicator and trap light. Takes the computer out of trap. mode

163 TTR (+0021) TRAP TRANSFER - Next instruction from loc. Y. Only normal transfer allowed when in trapping mode.

13. PSEUDO OPERATION CODES:

60 COUNT - COUNT - First card of symbolic deck. Gives number of cards in program.

60 END - END - Last card of symbolic deck.

60 BSS - BLOCK STARTED BY SYMBOL - Allocates block of storage. First loc. of block tagged by a symbol.

Page	13. PSEUDO OPERATION CODES--continued
81	PZE - PLUS ZERO - Assigns one word and puts zeros into S, 1, 2. Can specify Address, TAG, Decrement
81	EQU - EQUIVALENT - Used to define a symbol
81	OCT - OCTAL DATA - Data generating, series of variables
82	DEC - DECIMAL DATA - Data generating, decimal integers, fixed PT or floating PT.
148	SWT - SENSE SWITCH TEST - If Sense Switch (1-6) is on, skip one instr. Otherwise takes next
152	SLN - SENSE LIGHT ON - Turns on Sense Light designated by Y.
152	SLF - SENSE LIGHT OFF - Turns off all Sense Lights.
152	SLT - SENSE LIGHT TEST - Tests Sense Light designated by Y. If on, turns off and skips one instruction.

REVIEW AND SELF-TEST

The following pages contain another review and self-test. Again, page references will be given with the correct answers and it is suggested that the reference be checked on each question answered incorrectly.

Consider this test to be closed book. Use only the quick reference to instructions at the beginning of this lesson. Do not refer to any other part of the book while you are working the problems.

There will be 25 questions covering the high lights of the entire course and a problem to be flow charted and coded. Answer all questions and complete the coding before checking the correct answers. The answers to the 25 questions may be found on pages 190, 191 and the flow chart and correct solution to the problem on pages 192, 193 and 194.

Score this test as you did the previous one in Lesson 9. Your total score on the two parts of the test should be 70 or over and you should not take over two hours in completing the entire quiz.

Lesson 14, (cont'd)

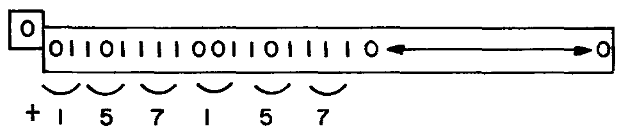
PROBLEMS:

118. Flow chart a typical Read Tape error routine:

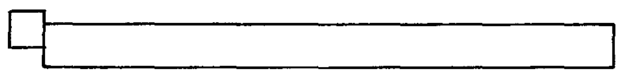
119. Flow chart a typical Write Tape error routine:

120. Using AND logical rules with MASK OCT 000777777777

Before:

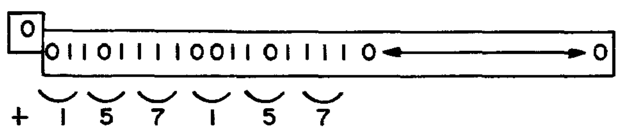


After:

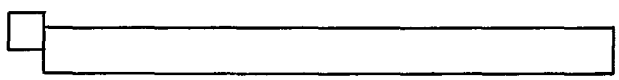


121. Using OR logical rules with MASK OCT 007777000000

Before:

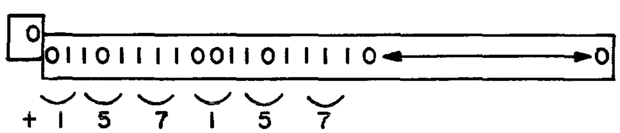


After:

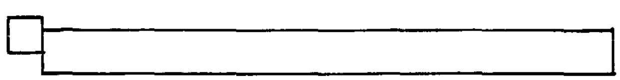


122. Using EXCLUSIVE OR logical rules with MASK as in problem 120

Before:



After:



123. a. How many sense switches are there? ________

b. How many sense lights are there? ________

124. Assume that a "flag" word is already in the AC. If bit 15 is on, go to AREA and if bit 27 is off, go to STOP. Write a partial program to accomplish this action.

LOC	OP	VARIABLE	REMARKS

125.

LOC	OP	VARIABLE	REMARKS
	AXT	1, 1	
	STO*	Block, 1	
	STO	HOLD	
BLOCK	CLA	AREA	
	STO	FIELD	

In the little program above, the STO* will place c(AC) into location ________________.

126. In the program above, if the AXT looked like this:

AXT -1, 1

The STO* will place c(AC) into location____________.

127. When operating in the trapping mode, control is transferred to what location when the conditions for transfer have been met?

Location____________.

128. The program, before assembly, is called the

______________________ ________________________

Lesson 14, (cont'd)

129.

LOC	OP	VARIABLE
	LXD	HOLD, 1

HOLD	PZE	6, 4, 2

What is loaded into XR1? ___________

130. XR4 contains the number 7.

```
Instr.        TIX   HOLD, 4, 5
          AA  CLA   AREA
```

a. After execution, what is in XR4? _____________

b. Program moves to location _______________

131. In problem 130, if the number in XR4 was 3:

a. After execution, what is in XR4? ___________

b. Program moves to location _______________

132. XR1 contains the number 2:

```
Instr.       TXI   HOLD, 1, 5
          AA CLA   AREA
```

a. After execution, what is in XR1? ___________

b. Program moves to location _________________

133. XR2 contains the number 13_8

```
Instr.        TXL   HOLD, 2, 12
          AA  CLA   AREA
```

a. Program moves to ________

b. After execution, what is in XR2? _______________

Op.	Variable

134. In storage, location HOLD looks like this?

```
CLA    50, 2
(+0500)
```

Show the instructions that will move the Op. Code into loc. AA, TAG into loc. BB and Address into loc. CC.

135. Show storage locations AA, BB and CC after problem 134 has been executed.

AA

BB

CC

136. Show the Octal representation of the following constants.

a. DEC 17B14

b. DEC 12

c. OCT 17563

d. DEC 4E-4

137. Problem: If A > B, go to loc. HOLD. If A = B, go to loc. HOLD + 1. If A < B, go to loc. HOLD + 2. Show a partial program to accomplish this action.

LOC	OP	VARIABLE	REMARKS

138. Show the floating point word for the following number:

276_{10} Show the word in Octal.

Char. Mantissa

139. Show the following numbers in normalized form.

a. 12476_{10}

b. $.00035_{10}$

c. $101\ 001_2$

d. 117.25_{10}

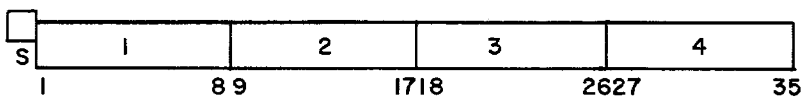
140. Show the Characteristic (in Octal) for the following floating point numbers:

CHAR.

a. -25_{10}

b. 163_{10}

141. Four numbers are packed into a word at loc. HOLD as follows:



Unpack number 3 and place into location HOLD 1 in positions S, 1-8. Show a partial program to accomplish this action:

LOC	OP	VARIABLE	REMARKS

142. Add two fixed point numbers; A(B0) and B(B0). Move in the AC so that the Binary point will be at B16 and store in HOLD.

LOC	OP	VARIABLE	REMARKS

PROBLEM:

143. On Tape 1, Channel A, are 100 values of X. On Tape 2, Channel A, are 100 values of Y (both sets in floating point). For each pair of X and Y, calculate X^2, Y^2 and XY. Write a record on Tape 5, Channel C, containing X, Y, X^2, Y^2 and XY (in the order given and also in floating point). Stop at end-of-tape.

Use Storage loc. HOLD for X, HOLD + 1 for Y, HOLD + 2 for X^2, HOLD + 3 for Y^2 and HOLD + 4 for XY.

FLOW CHART

PROGRAM

LOC	OP	VARIABLE	REMARKS

PROGRAM--continued

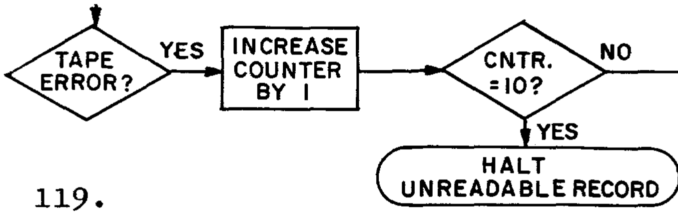
LOC	OP	VARIABLE	REMARKS

Lesson 14, (cont'd)

PROBLEM

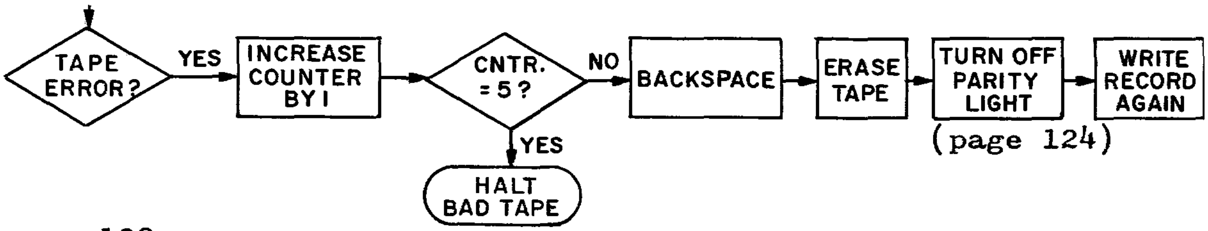
CORRECT ANSWERS

118.



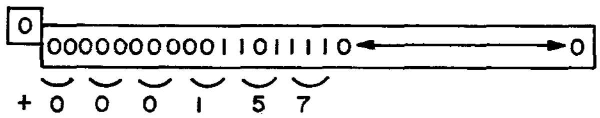
(page 122)

119.



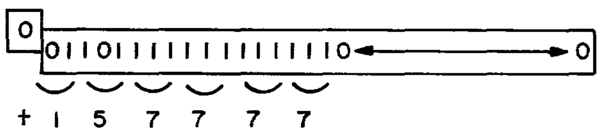
(page 124)

120.

After: 

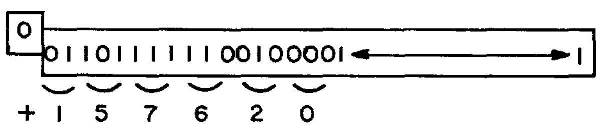
(page 137)

121.

After: 

(page 137)

122.

After: 

(page 137)

123.

a. 6 (page 147)

b. 4 (page 152)

124.

LOC	OP	VARIABLE	REMARKS
	PAI		Place c(AC) into Indicator
	CAL	=1B15	Pick up bit 15 to compare
	TIO	AREA	If 15 on, go to AREA
	CAL	=1B27	If off, pick up bit 27
	TIF	STOP	If off, go to STOP

(pages 148-149)

125. HOLD (page 156)

126. FIELD (page 156)

127. LOC. 0001 (page 162)

128. Source program (pg.167)

129. 2 (page 94)

130. a. 2
(page 95)
b. HOLD

131. a. 3
b. AA (page 95)

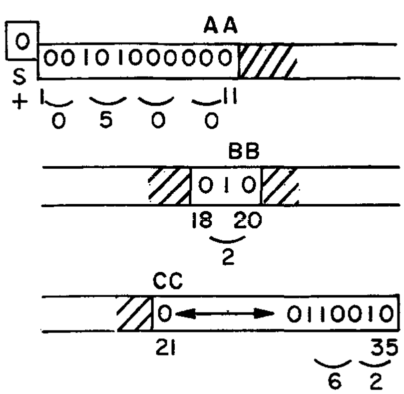
132. a. 7
(page 95)
b. HOLD

133. a. HOLD $12_{10} = 14_8$
(page 95)
b. 13_8

134.

LDQ	HOLD	
LLS	8	To move
ALS	27	Op. Code
STO	AA	
CLA	HOLD	To move
STT	BB	TAG
STA	CC	To move Address

135.



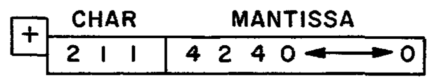
136. a. $+00021_\Delta0000000$
b. +204600000000
c. +000000017563
d. +165644000000
(page 83)

137.

LOC	OP	VAR.	REM.
	CLA	A	
	CAS	B	
	TRA	HOLD	A >
	TRA	HOLD 1	A =
	TRA	HOLD 2	A <

(page 77)

138.

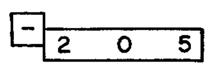


(page 41)

139. a. $.12476 \times 10^5$
b. $.35 \times 10^{-3}$
c. $.101001 \times 2^3$
d. $.11725 \times 10^3$
(page 41)

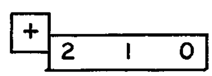
140. $-25_{10}=-31_8=-011001._2 =$
$.11001x2^5$ $(200+5 = 205_8)$

a.



$163_{10}=243_8 =$
$010100011._2 =$
$.10100011x2^{10}$
$(200 + 10 = 210_8)$

b.



(pg. 41)

141.

LOC	OP	VAR.	REMARKS
	CAL	HOLD	Move to AC
	ANA	MASK	Blank all but 3
	ALS	18	Shift Left
	SLW	HOLD 1	Store
MASK	OCT	000000777000	

(page 140)

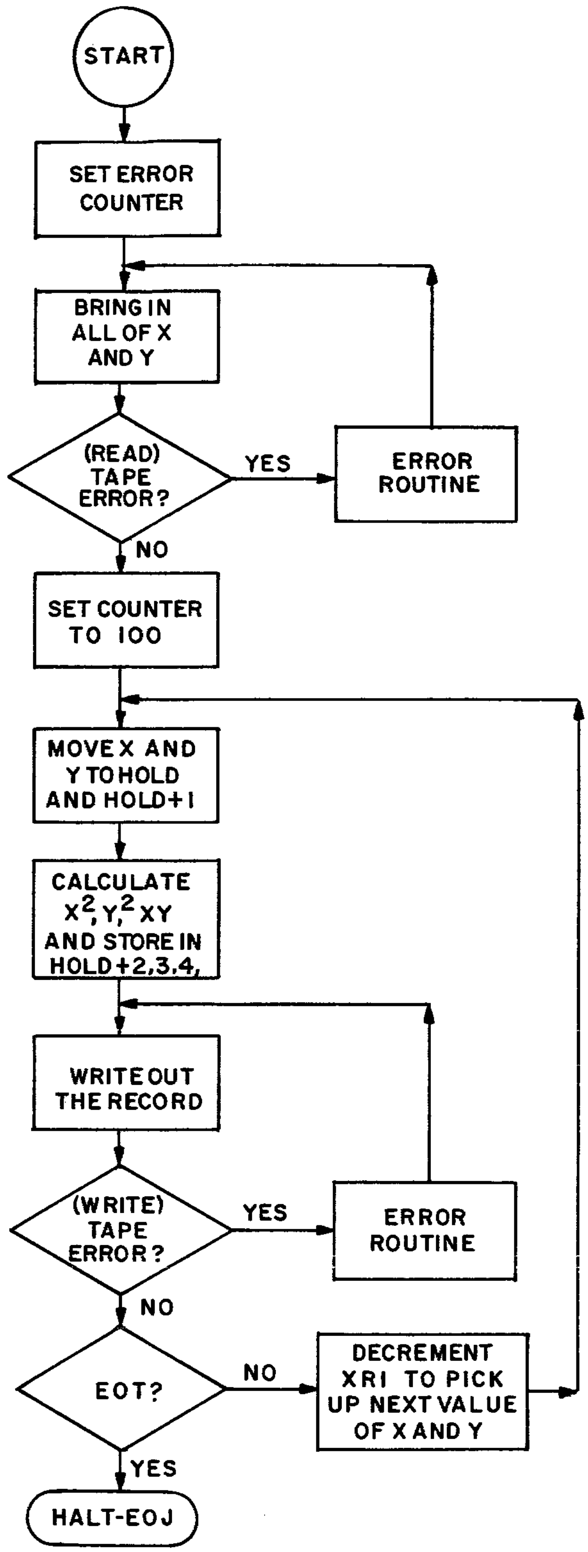
142.

LOC	OP	VAR	REM
	CLA	A	
	ADD	B	
	LRS	16	
	STO	HOLD	

(pg. 74)

PROBLEM 143:

FLOW CHART



PROGRAM

LOC	OP	VARIABLE	REMARKS
X	TAPENO	A1B	
Y	TAPENO	A2B	
Z	TAPENO	C5B	
START	AXT	10, 1	Set counter to try again
	TCOX	*	
RDX	RTBX		Read a record
	RCHX	IOX	
	TCOX	*	
	TEFX	STOP	
	TRCX	PEX	
	AXT	10, 1	Set counter to try again,
	TCOY	*	reading Y
RDY	RTBY		
	RCHY	IOY	
	TEFY	STOP	
	TRCY	PEY	
	AXT	100, 2	Set counter to 100
LOOP	CLA	X + 100, 2	
	STO	HOLD	Move X and Y to
	CLA	Y + 100, 2	output area
	STO	HOLD + 1	
	LDQ	X + 100, 2	Calculate X^2
	FMP	X + 100, 2	
	STO	HOLD + 2	
	LDQ	Y + 100, 2	Calculate Y^2
	FMP	Y + 100, 2	
	STO	HOLD + 3	
	LDQ	X + 100, 2	Calculate XY
	FMP	Y + 100, 2	
	STO	HOLD + 4	

PROGRAM--continued

PROBLEM 143:

LOC	OP	VARIABLE	REMARKS
	TCOZ	*	
	AXT	5, 1	Set number of times to try
WDZ	WTBZ		Write a record on tape
	RCHZ	100	
	TCOZ	*	
	TRCZ	PEZ	
	ETTZ		End of Tape?
	TRA	STOP	Yes, go to STOP
	TIX	LOOP, 2, 1	No, go back to loop
STOP	HTR	*	End of job
PEX	BSRZ		
	TIX	RDX, 1, 1	Try 10 times
	TRA	STOP	Bad tape
PEY	BSRY		
	TIX	RDY, 1, 1	
	TRA	STOP	
PEZ	BSRZ		Backspace
	WTBZ		Erase tape
	TCOZ	*	
	TRCZ	*	
	TIX	WDZ, 1, 1	Try again? Yes, to WDZ
	TRA	STOP	No, bad tape
IOO	IORT	HOLD, , 5	Output Command
IOY	IORT	Y, , 100	Input Y Command
IOX	IORT	X, , 100	Input X Command
X	BSS	100	Allocate input areas for
Y	BSS	100	X and Y
HOLD	BSS	5	Allocate output area
	END		

LESSON 15

SAMPLE PROGRAM: Most of the examples and problems throughout this book showed only partial programs, enough to solve the particular problem being presented. The sample program that follows attempts to show a complete source program, starting with the statement of the problem to be solved and followed by the programmers' flow chart and the coding required to execute the problem.

The Source Program on pages 198 and 199 shows the actual print-out the programmer will receive. Each line represents one punched card of the Source Card Deck (refer to page 167).

The following five pages show the cards of the Object Program after assembly. These contain all the information contained in the Source Program, ready to be used by the computer to operate on live data.

PROBLEM

Given a block of no more than 1000 floating point numbers, located at AREA, AREA + 1, etc. The last word in the block contains all binary ones.

Find the number of words in the block (excluding the word containing all ones) and place the number into location NIB, in floating point.

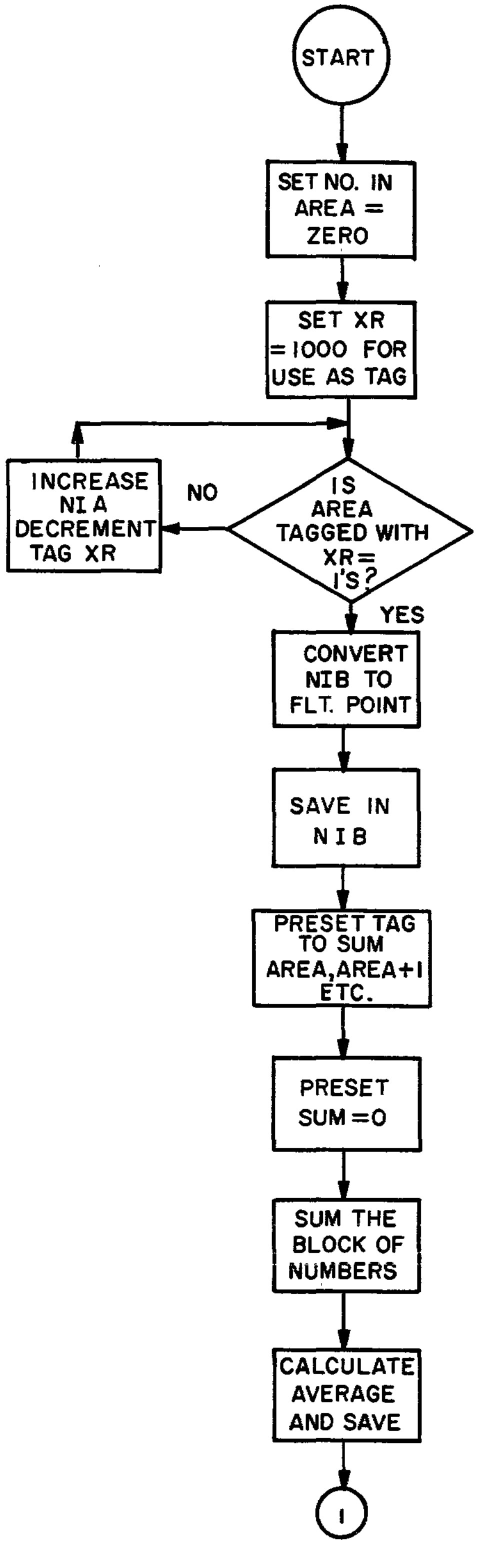
Find the average of all of the words and place it into location TAVE.

Find the average of all plus words and place into location PAVE.

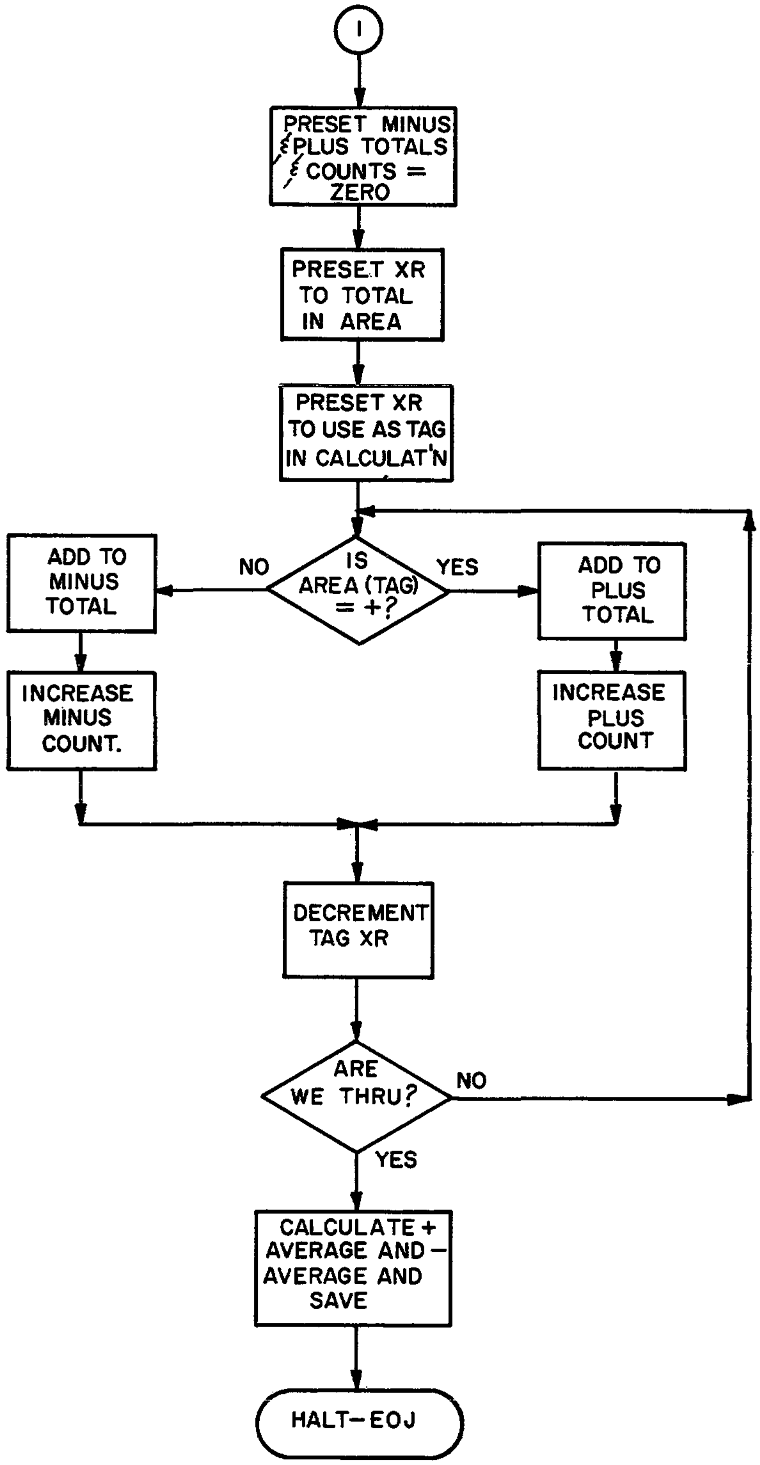
Find the average of all the minus words and place into location MAVE.

When all averages have been found and put into the proper locations, the job is done.

FLOW CHART


START
SET NO. IN AREA = ZERO
SET XR = 1000 FOR USE AS TAG
INCREASE NIA DECREMENT TAG XR
NO
IS AREA TAGGED WITH XR = 1'S?
YES
CONVERT NIB TO FLT. POINT
SAVE IN NIB
PRESET TAG TO SUM AREA, AREA+1 ETC.
PRESET SUM = 0
SUM THE BLOCK OF NUMBERS
CALCULATE AVERAGE AND SAVE
1

FLOW CHART--continued

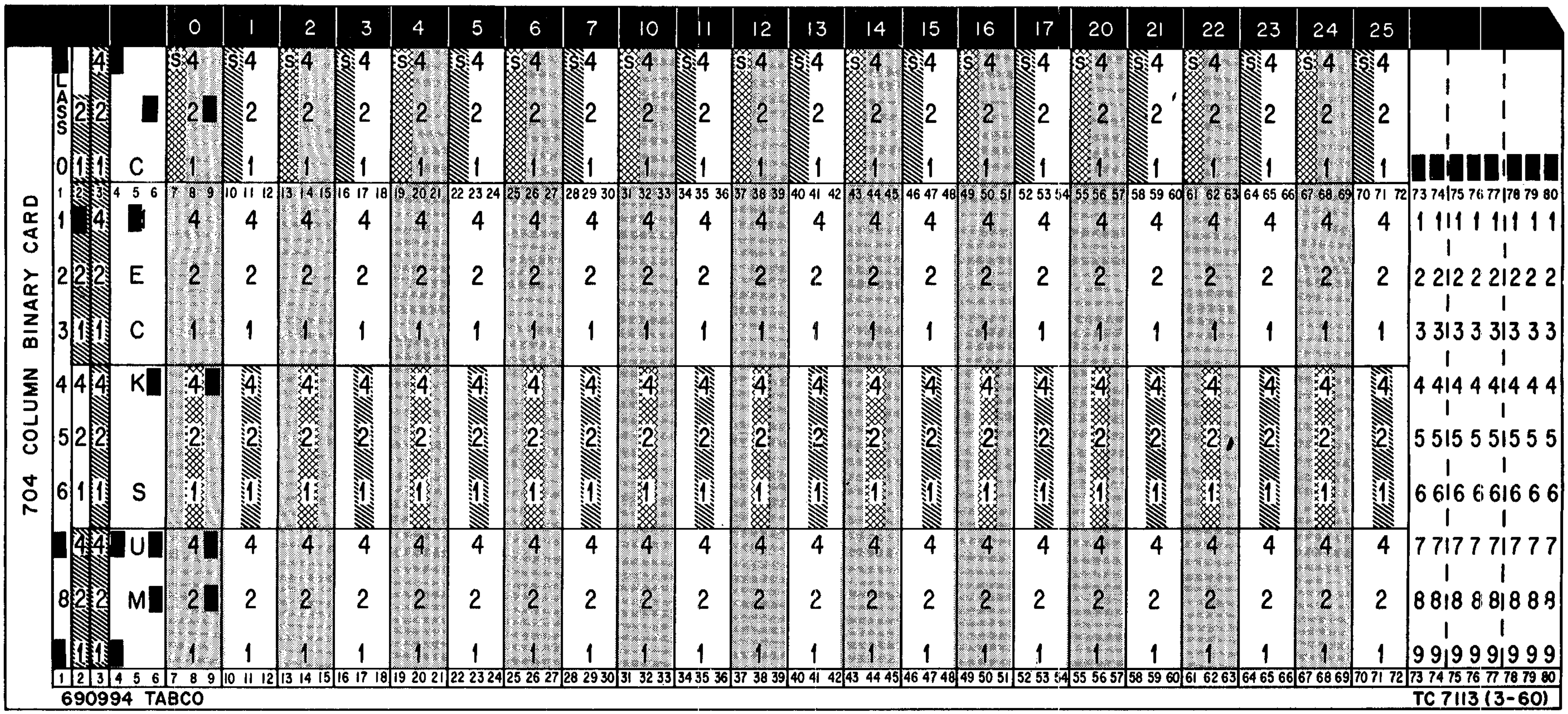


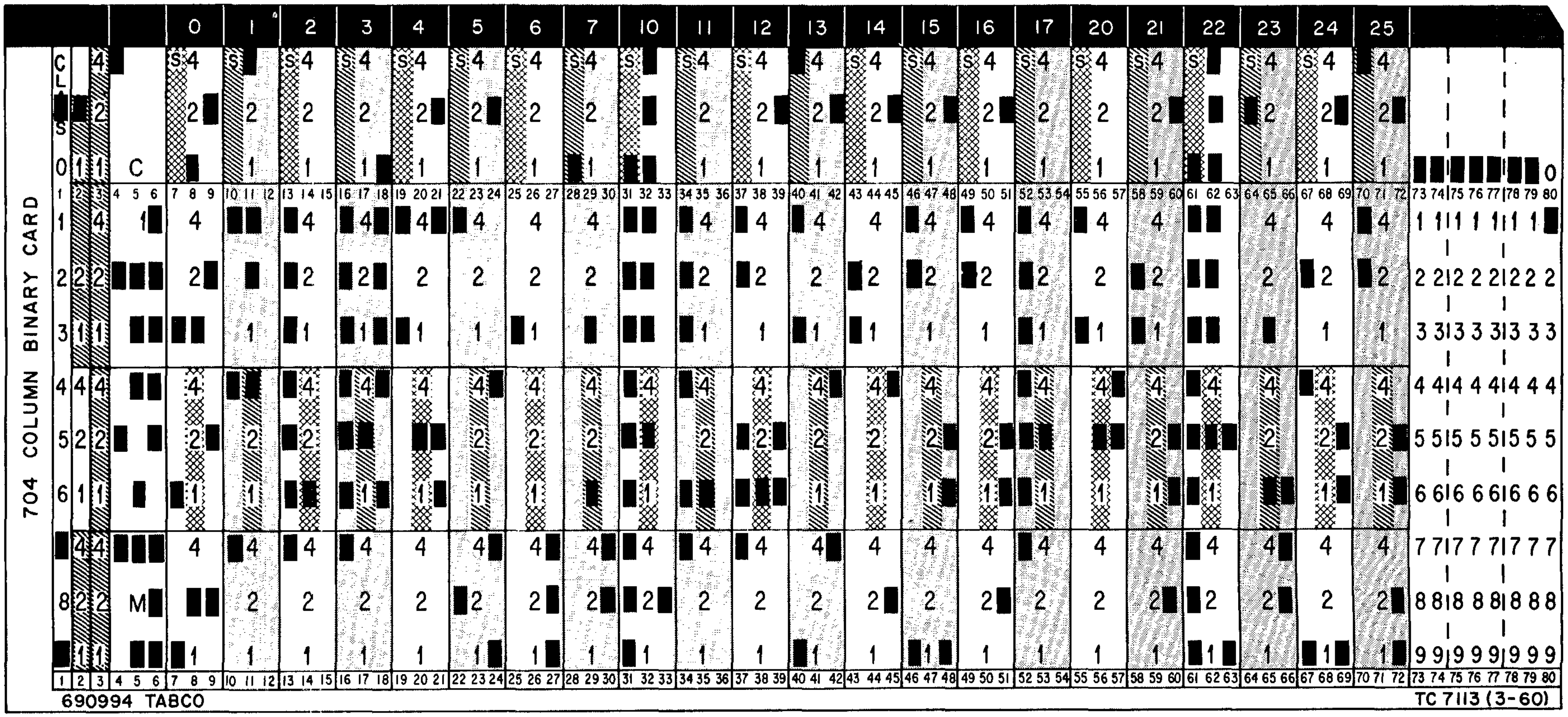
```
00000  0774 00 1 00000   START AXT    0,1
00001  0774 00 2 01750         AXT    1000,2
00002  0500 00 2 02030   LOOP  CLA    AREA+1000,2    IS IT THE END
00003  0402 00 0 02045         SUB    =0777777777777 WORD.
00004  0100 00 0 00007         TZE    TNIB           YES
00005  1 00001 1 00006         TXI    *+1,1,1        NO, BUMP THE COUNT OF NIB
00006  1 77777 2 00002         TXI    LOOP,2,-1      TRY NEXT WORD
00007  0754 00 1 00000   TNIB  PXA    ,1             PUT COUNT IN AC
00010  0634 00 1 02030         SXA    FXNIB,1        SAVE FIXED POINT COUNT
00011 -0501 00 0 02044         ORA    =0233000000000 CONVERT FIXED POINT TO
00012  0300 00 0 02042         FAD    =0             FLOATING POINT
00013  0601 00 0 02031         STO    NIB            SAVE FLOATING POINT NIB
00014  0600 00 0 02032         STZ    TOTAL          PRESET SUM TØ ZERØ
00015  0774 00 2 00000         AXT    0,2            PRESET TAG FOR ADDING AREA, AREA+1, ETC.
00016  0500 00 2 00060   LØØP1 CLA    AREA,2         SUM THE
00017  0300 00 0 02032         FAD    TOTAL          BLOCK OF NUMBERS
00020  1 77777 2 00021         TXI    *+1,2,-1       BUMP TAG FOR AREA
00021  2 00001 1 00016         TIX    LOOP1,1,1      ALL THRU SUMMING, NO
00022  0241 00 0 02031         FDP    NIB            YES, CALCULATE
00023 -0600 00 0 02033         STO    TAVE           AVERAGE AND SAVE
00024  0600 00 0 02034         STZ    PTOT           PRESET TOTALS
00025  0600 00 0 02035         STZ    MTØT         AND COUNTS
00026  0600 00 0 02036         STZ    PCNT
00027  0600 00 0 02037         STZ    MCNT
00030  0534 00 1 02030         LXA    FXNIB,1        PICK UP NUMBER IN BLOCK
00031  0774 00 2 00000         AXT    0,2            PRESET TAG FOR AREA
00032  0500 00 2 00060   LØØP2 CLA    AREA,2         PICK UP NUMBER
00033  0120 00 0 00052         TPL    PLUS           IS IT PLUS, YES
00034  0300 00 0 02035         FAD    MTØT           NØ
00035  0601 00 0 02035         STO    MTØT
00036  0500 00 0 02037         CLA    MCNT           BUMP COUNT OF
00037  0300 00 0 02043         FAD    =1.0           MINUS NUMBERS
00040  0601 00 0 02037         STO    MCNT
```

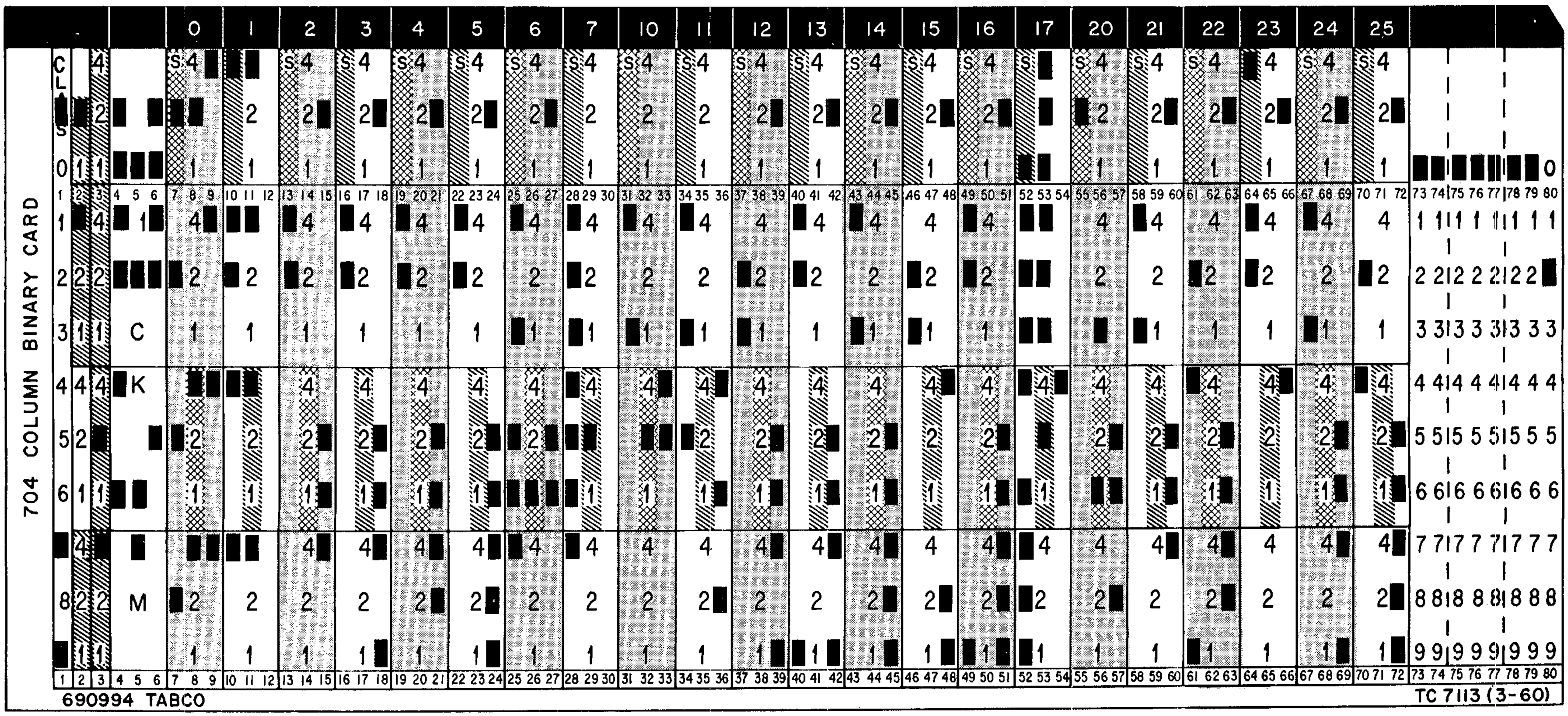
```
00043   0500 00 0 02034
00044   0241 00 0 02036           FDP     PCNT           PLUS AVERAGE
00045  -0600 00 0 02040           STQ     PAVE
00046   0500 00 0 02035           CLA     MTOT           CALCULATE MINUS
00047   0241 00 0 02037           FDP     MCNT           AVERAGE
00050   0601 00 0 02041           STO     MAVE
00051   0000 00 0 00051    STOP   HTR     *              ALL THRU STOP
00052   0300 00 0 02034    PLUS   FAD     PTOT           SUM PLUS NUMBERS
00053   0601 00 0 02034           STO     PTOT
00054   0500 00 0 02036           CLA     PCNT           BUMP PLUS
00055   0300 00 0 02043           FAD     =1.0           COUNT
00056   0601 00 0 02036           STO     PCNT
00057   0020 00 0 00041           TRA     TEST
00060                      AREA   BSS     1000
02030                     FXNIB   BSS     1              B35 COUNT OF NUMBER IN AREA
02031                      NIB    BSS     1              FLOATING POINT COUNT OF NUMBER IN AREA
02032                     TOTAL   BSS     1              TOTAL OF ALL NUMBERS
02033                      TAVE   BSS     1              AVERAGE OF ALL NUMBERS
02034                      PTOT   BSS     1              PLUS NUMBERS TOTAL
02035                      MTOT   BSS     1              MINUS NUMBERS TOTAL
02036                      PCNT   BSS     1              PLUS NUMBERS COUNT
```

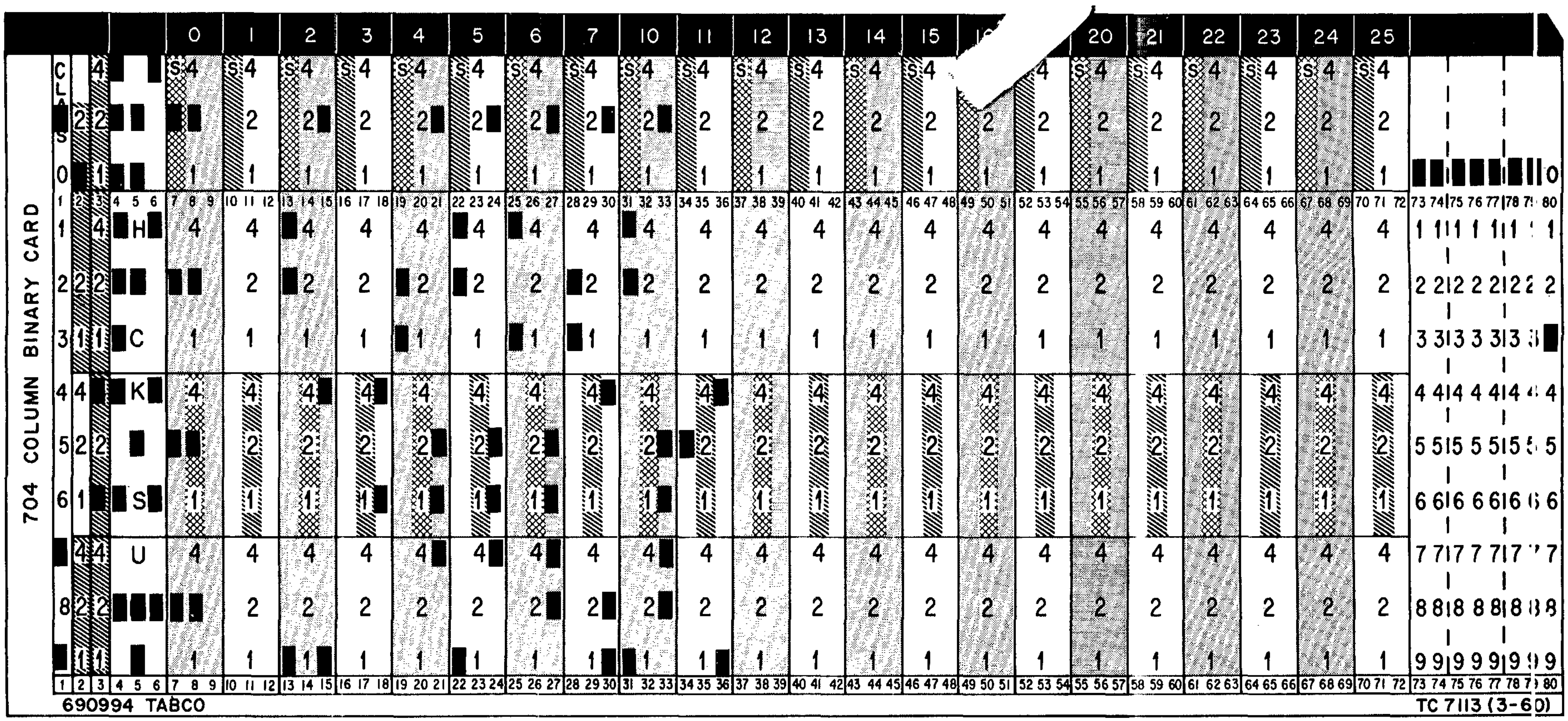
```
                                                          9/05/62     PAGE 2

02037                      MCNT   BSS     1              MINUS NUMBERS COUNT
02040                      PAVE   BSS     1              PLUS NUMBERS AVERAGE
02041                      MAVE   BSS     1              MINUS NUMBERS AVERAGE
                                  END

  LITERALS
02042   000000000000
02043   201400000000
02044   233000000000
02045   777777777777
```

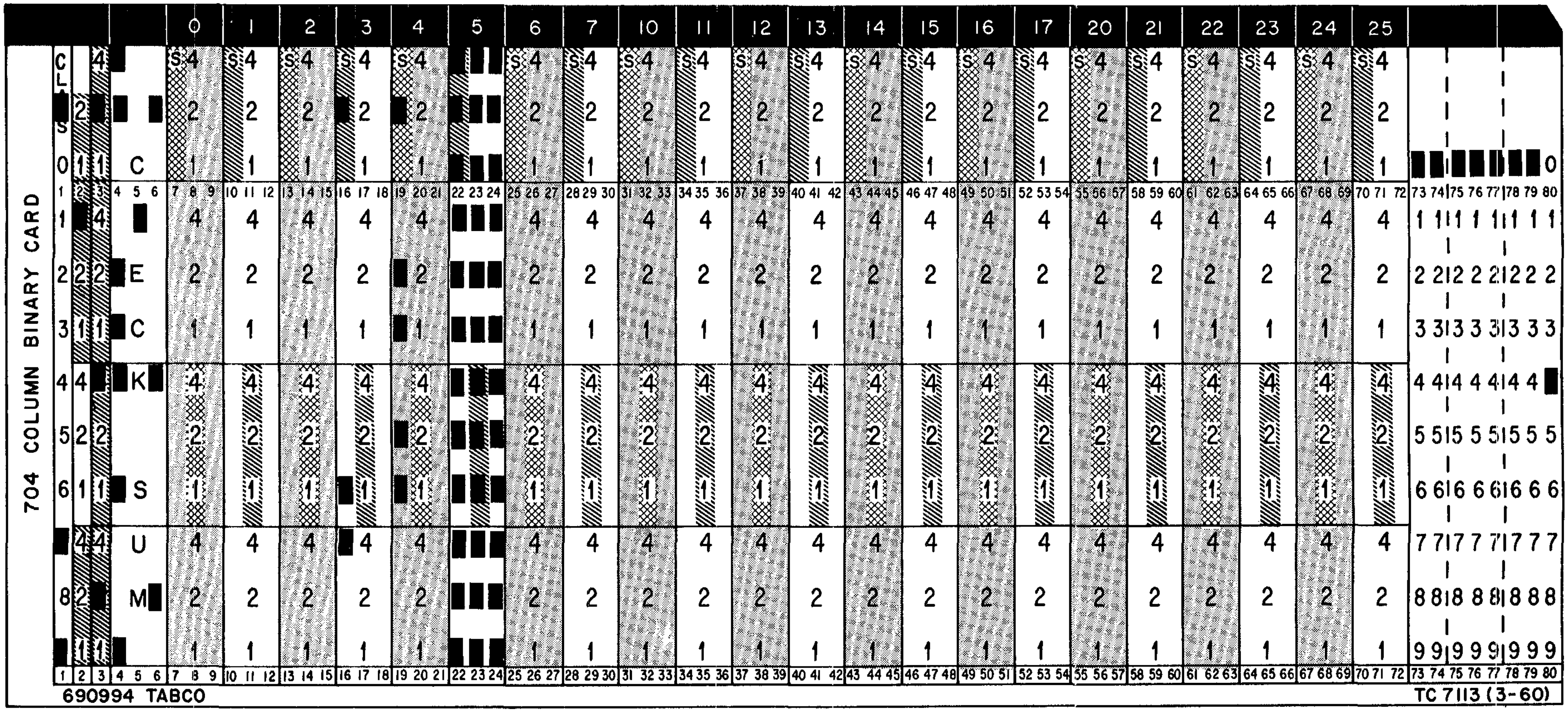
```
                         POST PROCESSOR ASSEMBLY DATA

         2046 IS THE FIRST LOCATION NOT USED BY THIS PROGRAM

   REFERENCES TO DEFINED SYMBOLS
         2031    NIB     13,    22
           60   AREA      2,    16,    32
            2   LOOP      6
         2041   MAVE     50
         2037   MCNT     27,    36,    40,    47
         2035   MTOT     25,    34,    35,    46
         2040   PAVE     45
         2036   PCNT     26,    44,    54,    56
           52   PLUS     33
         2034   PTOT     24,    43,    52,    53
           51   STOP
         2033   TAVE     23
           41   TEST     57
            7   TNIB      4
         2030  FXNIB     10,    30
           16  LOOP1     21
           32  LOOP2     42
            0  START
         2032  TOTAL     14,    17

    NO ERROR IN ABOVE ASSEMBLY.

   TIME TO COMPILE/ASSEMBLE WAS      3 MILLIHOURS.
```


704 COLUMN BINARY CARD
690994 TABCO
TC 7113 (3-60)


704 COLUMN BINARY CARD
690994 TABCO
TC 7113 (3-60)


704 COLUMN BINARY CARD
CLASS
C
K
M
690994 TABCO
TC 7113 (3-60)


704 COLUMN BINARY CARD
690994 TABCO
TC 7113 (3-60)


704 COLUMN BINARY CARD
0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20 21 22 23 24 25
690994 TABCO
TC 7113 (3-60)

CONCLUDING REMARKS: As a final step in this course of instruction, turn to the Index at the back of the book and read each term and phrase. If there are any that you do not understand thoroughly, please turn to the indicated page (or pages) and review the topic.

It must be understood that what you have learned is only the beginning of the learning process. To become an accomplished programmer, you must work with the machine and with the problems to be solved by the machine. Nothing can be substituted for experience.

Many of the areas covered in the book only give you a basic idea that such a method exists. No more is possible in a book of this nature (or in a short lecture course, for that matter). Constant use of the concepts and instructions will do more than anything else to implant them firmly in your mind.

You will find this book to be helpful as a source of review and reference as you learn more about programming. The only way to learn more about programming is to work as a programmer.

You now have enough knowledge of the terminology, techniques, and operating instructions of the 7090 computer, that you should feel confident in being able to pull your own weight as a fledgeling programmer. Working under the supervision of an experienced programmer will complete your education.

If there are any areas in the book that you feel are inadequately covered, feel free to write to the author with your comments and remarks. They will be evaluated and, if acceptable, will be used for future revisions of the book.

INDEX