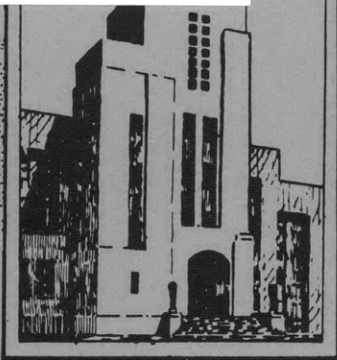


V393
.R46

MIT LIBRARIES

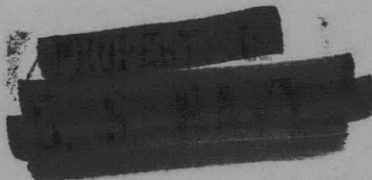


DEPARTMENT OF THE NAVY
DAVID TAYLOR MODEL BASIN

HYDROMECHANICS

TRAINING MANUAL ON PROGRAMMING FOR THE IBM 704

AERODYNAMICS



by

Carl L. Tibery

STRUCTURAL
MECHANICS

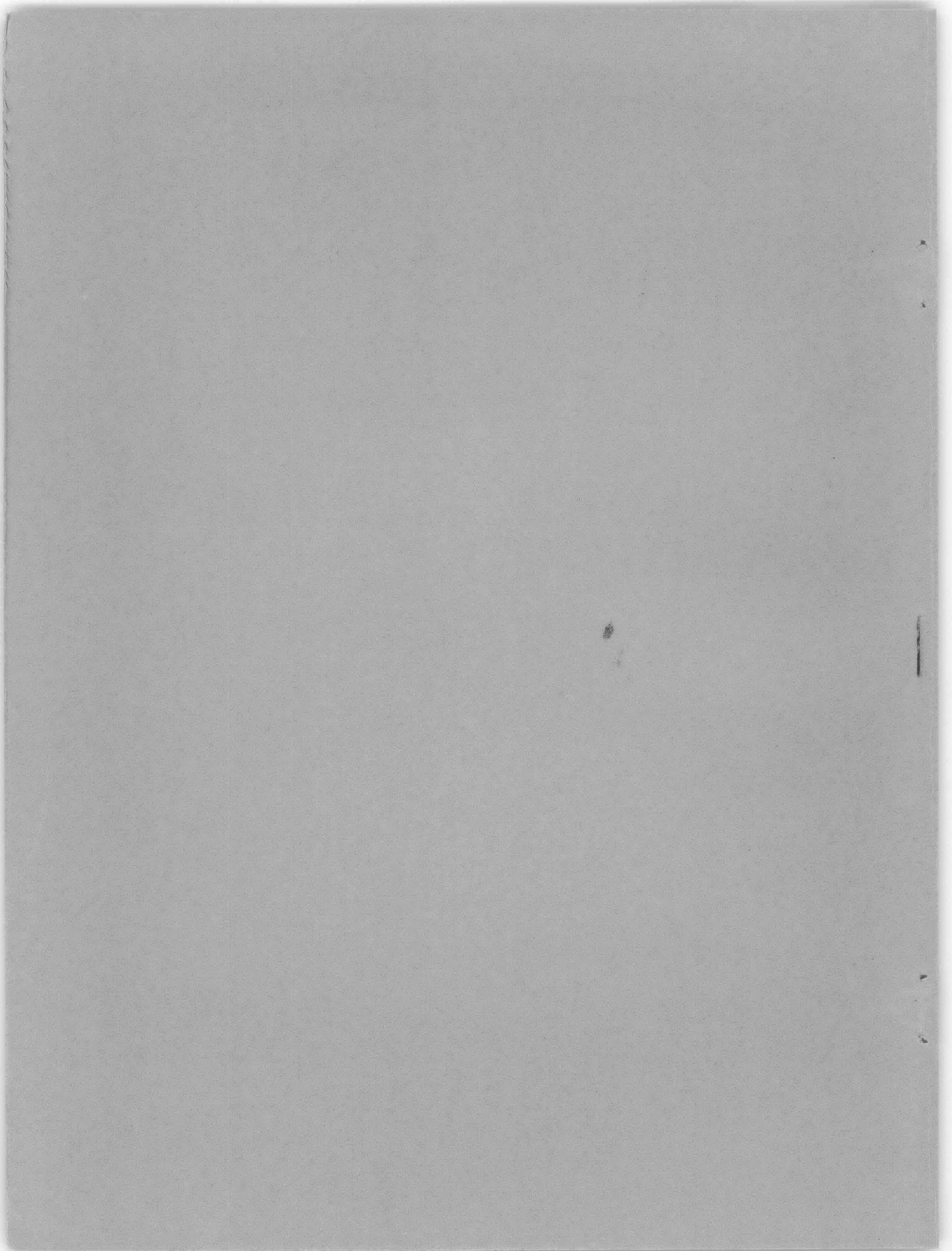


APPLIED
MATHEMATICS

APPLIED MATHEMATICS LABORATORY
RESEARCH AND DEVELOPMENT REPORT

April 1960

Report 1368



TRAINING MANUAL ON PROGRAMMING FOR THE IBM 704

by

Carl L. Tibery

April 1960

Report 1368

TABLE OF CONTENTS

	Page
ABSTRACT	1
INTRODUCTION	1
USE OF THE CONTROL CARDS	2
CHAPTER I – ADDITION AND SUBTRACTION	3
CHAPTER II – MULTIPLICATION AND DIVISION	10
CHAPTER III – INDEX REGISTERS AND THEIR USE	15
CHAPTER IV – FLOW CHARTS	20
CHAPTER V – AUTOMATIC PROGRAMMING	26
CHAPTER VI – BELL INPUT-OUTPUT SYSTEM	32
CHAPTER VII – SUBROUTINES	36
CHAPTER VIII – NUMBERS IN MACHINE LANGUAGE	46
CHAPTER IX – INSTRUCTIONS IN MACHINE LANGUAGE	54
CHAPTER X – LOGICAL OPERATIONS	60
CHAPTER XI – PROGRAM CHECKING	71
CHAPTER XII – READING AND WRITING ON TAPE	80
CHAPTER XIII – FORTRAN, AN AUTOMATIC CODING SYSTEM	90
ACKNOWLEDGMENT	94
APPENDIX A – OPERATIONS BY ALPHABETIC CODE	95
APPENDIX B – INSTRUCTIONS TO THE SAP AND BELL SYSTEMS	96
APPENDIX C – DECIMAL, BINARY, AND OCTAL NUMBER SYSTEMS	97
REFERENCES	105
BIBLIOGRAPHY	105
INDEX	107

LIST OF FIGURES

	Page
Figure 1 – Contents of the Core Units	4
Figure 2 – Sample Coding Form	4
Figure 3 – An IBM Card	5
Figure 4 – An IBM Card with the Instruction ADD 121	6
Figure 5 – Program to Add Two Numbers	7
Figure 6 – Representation of the Product	10
Figure 7 – Product of 6×4	10
Figure 8 – Program to Evaluate the Product of Two Numbers	11
Figure 9 – Representation of the Dividend	11
Figure 10 – Program to Evaluate the Quotient of Two Numbers	12
Figure 11 – Program to Find the Sum of Ten Numbers Using an Index Register	17
Figure 12 – Program to Place Sum of Each Pair of Numbers in Consecutive Locations	21
Figure 13 – A Logical Choice	21
Figure 14 – An Evaluation of a Formula	21
Figure 15 – Counting Symbol	22
Figure 16 – A Connector	22
Figure 17 – An Assertion or Note	22
Figure 18 – Flow Chart to Evaluate $y_i = (x_i^2 + 3x_i - 5)$, $i = 1, 2, \dots, 100$	22
Figure 19 – Program Associated with the Flow Chart of Figure 18	23
Figure 20 – Symbolic Program to Find the Sum of Ten Numbers	27
Figure 21 – Symbolic Program to Evaluate $y_i = (x_i^2 + 3x_i - 5)$, $i = 1, 2, \dots, 100$	28
Figure 22 – Order of Instructions to Run a Symbolic Program Using SAP 3–7	34
Figure 23 – Flow Chart to Evaluate the Sum of Two Numbers	37
Figure 24 – Symbolic Program Associated with Flow Chart of Figure 23	37

	Page
Figure 25 – SHARE Subroutine for Evaluating TAN X	39
Figure 26 – Flow Chart to Evaluate the TAN X Using a SHARE Subroutine	40
Figure 27 – Symbolic Program Associated with Flow Chart of Figure 26	40
Figure 28 – Flow Chart Demonstrating Variable Connector	41
Figure 29 – Program Associated with Flow Chart of Figure 28	42
Figure 30 – SHARE Subroutine for Evaluating the Square Root of the Absolute Value of X	45
Figure 31 – Bits of a Core Unit	46
Figure 32 – Floating-Point Number Representation	47
Figure 33 – Decimal 5 in Normalized Floating-Point Binary Form	47
Figure 34 – Bits of the Accumulator	47
Figure 35 – Program to Check for Overflow in Addition	48
Figure 36 – Program to Check Whether or Not Division Takes Place	50
Figure 37 – Type A Instruction	54
Figure 38 – Type B Instruction	54
Figure 39 – SAP Printout of TIX 101,1,1	55
Figure 40 – Machine Representation of TIX 101,1,1	55
Figure 41 – Octal Code for FISCAL YEAR 1959	56
Figure 42 – Octal Representation of FISCAL YEAR 1959 in Core Storage	56
Figure 43 – Type A Instruction	58
Figure 44 – Type B Instruction	58
Figure 45 – Program to Modify the Address of an Instruction	60
Figure 46 – Exchange of Bits as Result of CAL A	61
Figure 47 – Operation of ACL Y	61
Figure 48 – A Packed Word	62
Figure 49 – Extractor Pattern 1	62

	Page
Figure 50 – N_1 in the $c(AC)_P, 1-11$	62
Figure 51 – N_1 in $c(MQ)_S, 24-35$	63
Figure 52 – P_1 in $c(MQ)_S, 1-11$	63
Figure 53 – Extractor Pattern	64
Figure 54 – Location DATA Now Contains N_1 and N_3	64
Figure 55 – Location DATA Now Contains $P_1, N_2,$ and N_3	64
Figure 56a – Extractor Pattern 1 and its Octal Code	65
Figure 56b – Extractor Pattern 2 and its Octal Code	65
Figure 57 – Flow Chart for Extraction of N_1 from Packed Word and Insertion of P_1 into Packed Word	65
Figure 58 – Program Associated with Flow Chart of Figure 57	66
Figure 59 – Operation of ANA Instruction	67
Figure 60 – A Binary Card with Twenty-two Instructions	71
Figure 61 – Sequence of Cards to Run with Binary Deck	72
Figure 62 – Instruction for Snapshot Dump	73
Figure 63 – Sequence of Cards when Making Corrections and/or Dumps with a Binary or Symbolic Deck	74
Figure 64 – Program to Obtain a Dump at Location 165 and a Post-Mortem Dump	74
Figure 65 – Program to Multiply Two 10×10 Matrices, A and B	75
Figure 66 – Programs to Run a Symbolic Deck with Dumps or a Binary Deck with Corrections and/or Dumps	78
Figure 67 – One Frame on Tape	80
Figure 68 – BCD Mode of Tape	80
Figure 69 – Longitudinal Check	81
Figure 70 – Flow Chart to Copy 1000 10-Word Records from Tape to Memory	82
Figure 71 – Program Associated with Chart of Figure 70	83

	Page
Figure 72 – Flow Chart to Copy 1000 Words from Core Storage in Units of 10-Word Records onto Tape	84
Figure 73 – Program Associated with Flow Chart of Figure 72	85
Figure 74 – A FORTRAN Program to Evaluate the NORM of a Matrix	91
Figure 75 – Data Card with Format 5E14.8	91
Figure 76 – A Few Binary Integers and Their Decimal Equivalentents	97
Figure 77 – Addition and Multiplication Tables for Binary Number System	97
Figure 78 – Multiplication of 6 by 7	98
Figure 79 – A Few Binary Integers with Their Octal and Decimal Equivalentents	98
Figure 80 – Addition and Multiplication Tables for Octal Number System	99

ABSTRACT

This training manual consolidates the essential information from the IBM 704 Reference Manual, the Bell Telephone Laboratories IBM 704 Input-Output System – BE SYS 2, and the United Aircraft SAP 3-7 Programmer's Notes into one presentation, from which the mathematician can learn to write a program for the IBM 704 using those systems.

Among the topics covered are flow charting, machine language, symbolic programming, subroutines, input-output operations, and FORTRAN, an IBM automatic coding system. Each chapter includes simple examples and exercises.

INTRODUCTION

The purpose of this manual is to introduce the techniques of programming mathematical problems for the IBM 704 to the mathematician who is unacquainted with high-speed computers.

Programming for the IBM 704 in the Applied Mathematics Laboratory of the David Taylor Model Basin requires a knowledge of several systems, as described in: IBM 704 Reference Manual,¹ Bell 704 Input-Output System - BE SYS 2,² Programmer's Notes for SAP 3-7,³ FORTRAN Primer,⁴ FORTRAN Reference Manual,⁵ FORTRAN II Reference Manual,⁶ and 704 Snapshots.⁷

This manual coordinates the essential points of References 1-7 into one presentation, from which the mathematician should be able to learn to prepare a complete program for the solution of a problem on the IBM 704.

Since one learns best by applying what he is learning, this manual is written so that, from the very first chapter, the mathematician can run programs on the computer. A set of control cards is provided with each manual (in a pocket inside the back cover), and to perform the input-output operations the cards are simply placed before and after the symbolic program.

The instructions necessary to write a program for a fairly sophisticated mathematical problem are presented first. Then the meaning of the control cards is explained, and the method for preparing the control cards is given. In the chapters that follow, more instructions, subroutines, symbolic codes, and other input-output devices are presented, but the responsibility for preparing the control cards is left to the user of this manual.

Thus, upon completion of the exercises given in this manual, the mathematician should be able to program, independently, problems for the IBM 704, and should have no difficulty in using the references to find any new instructions he may need.

Appendixes A and B contain tables of the IBM 704 operations, SAP codes, and Bell System codes.

¹References are listed on page 105.

USE OF THE CONTROL CARDS

You will find a set of control cards in the pocket inside the back cover of this manual. They must be in the following order, with the label INT (initial) or FIN (final) appearing in columns 73-80 of the card:

INT 2
FIN 1
FIN 2
FIN 3
FIN 4
FIN 5
FIN 6
FIN 7
FIN 8
FIN 9
FIN 10
FIN 11

These control cards, except for the one labeled FIN 5, are to be used for all the exercises through Chapter VII. The card labeled FIN 5 is to be replaced in each chapter by a new card which has the chapter number preceding FIN 5.

The first initial control card, INT 1, is not included in this deck. It is called the JOB card and contains the programmer's problem number. It can be obtained from the supervisor of computer operations.

CHAPTER I

ADDITION AND SUBTRACTION

This is an introduction to the solving of mathematical problems on the IBM 704 computing machine.

One of the most important functions of a computing machine is to perform the four arithmetic operations, and its great utility comes from the fact that it can perform these operations many times faster than a human being can using a desk calculator.

Similar to a human brain, the computer must first "find" the number it wants to operate on and then perform the operations. How is this done?

This is done by instructions which the programmer gives to the machine. The 704 has 32,768 storage units, or cells, which can contain either instructions or data. Suppose you have stored in location 120 the number +5 and in location 121 the number +2, and you want to find the sum of these two numbers.

The instruction "Clear and Add"

CLA 120

will bring the contents of cell 120 (namely, the number +5) to a working part of the computer called the *accumulator*. There the contents of cell 120 will be ready for comparison. Then the instruction "Add"

ADD 121

will tell the machine to add the contents of cell 121 to the contents of the accumulator. The result, +7, will replace the number, +2, in the accumulator. But next, you will want to store this sum in a convenient location. This is done by the instruction "Store"

STO Y

Y being the location where you desire the result to be. For example, if you want this sum stored in location 122, you would write the instruction thus:

STO 122

The internal memory or core storage, assuming the instructions to begin in location 100, now contains the following information; see Figure 1. (Note that the first cell is numbered 0.)

But now a question arises. How do these instructions and data get into the internal memory (core storage) of the machine, and how do you get the result out of the memory?

One way of doing this is by punching the instructions and data on cards and then letting the machine read the cards into the memory. Instructions are normally written on special coding paper which specifies the card columns to be punched. An example is given in Figure 2. The computer will read the data and instructions into the memory by means of control cards placed before the instruction cards. Then by a control card placed at the end of the instructions and some additional ones placed at the end of the data, the computer will write the

Location Number	Contents
0	
.	
.	
.	
100	CLA 120
101	ADD 121
102	STO 122
103	
.	
.	
.	
120	+5
121	+2
122	+7
123	
.	
.	
32,767	

Figure 1 – Contents of the Core Units

IBM Data Processing Division

SHARE 704 Symbolic Coding Form

Problem																	
Coder John Smith								Date September 15, 1959		Page 1 Of 1							
H	Location					Op	Address, Tag Decrement					Comments		Identif-ication			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
						ORG					100						
						CLA					120						
						ADD					121						
						STO					122						
						ORG					120						
						DEC					5						
						DEC					2						

Figure 2 – Sample Coding Form

answer or answers on a magnetic tape. A printer will print the information from tape onto paper. (The control cards will be explained in Chapters VI and VII.)

To begin our instructions at location 100, place immediately after the initial control cards the "Origin" card

ORG 100

This instructs the machine to place the instructions on the cards that follow in core storage, starting at location 100.

Thus to perform your sample problem you would place the sets of cards in the order indicated in the hopper of the machine:

Problem												
Coder John Smith						Date September 15, 1959			Page 1 Of 1			
H	Location		Op		Address, Tag		Decrement		Comments		Identif- cation	
1	2	6	7	8	10	11	12			72	73	80
(Initial Control Cards)												
(Final Control Card #1)												
(Final Control Cards #2-11)												

100M P50 2/59
Form 120-6809-3

Figure 5 – Program to Add Two Numbers

SUMMARY – Chapter I

To run a program with cards in the 704, place them as follows in the card hopper:

Initial Control Cards Nos. 1–2

Your Own Instructions

Final Control Card No. 1

Data

Final Control Cards Nos. 2–11

Instructions covered in this chapter are:

ORG Y	Origin	Specifies that the information on the cards that follow is to be stored in the core storage beginning at location Y.
CLA Y	Clear and Add	Places the contents of Y in accumulator and leaves Y unchanged.
ADD Y	Add	Adds contents of Y to contents of accumulator, places the sum in the accumulator, and leaves Y unchanged.
STO Y	Store	Stores contents of accumulator in location Y, and leaves the accumulator unchanged.
DEC N	Decimal Data	Specifies that N is a decimal integer.
SUB Y	Subtract	Subtracts contents of Y from contents of accumulator, places the difference in the accumulator, and leaves Y unchanged. (See Exercise 2.)

EXERCISES – Chapter I

1. Write a program to find the sum of five different numbers. Begin the first instruction in location 100 and the first data item in location 200. Store the result in 300.*

2. Given that the instruction “Subtract”

SUB Y

will subtract the contents of Y from the contents of the accumulator and leave the result in the accumulator, write a program to add 5 and 7; then subtract 4. Assume same ORG cards as in Exercise 1. Store the result in 300.

3. Punch your instruction and data cards and place them with the control cards in the order given in this chapter; then run Exercises 1 and 2 on the 704. Check your answer with the machine answer. Notice that the first of the Final Control Cards is to be placed after your last instruction card.

*Note that on your printout the core storages are numbered in the octal system and not the decimal system. See Appendix A.

CHAPTER II

MULTIPLICATION AND DIVISION

In Chapter I you learned how to subtract and add on the 704. Now you will learn how to divide and multiply.

MULTIPLICATION

To multiply, you must use two registers: the accumulator or AC register, and the multiplier-quotient or MQ register. First, place the multiplicand with sign in the MQ register by the instruction "Load MQ"

LDQ Y

This will replace the contents of the MQ with the contents of Y, leaving the contents of Y unchanged.

Then the instruction "Multiply"

MPY Y

will multiply the contents of MQ by the contents of Y.

The most significant half (msh) and the sign of the product will appear in the AC, and the least significant half (lsh) will appear in the MQ. Read your answer as if the contents of the AC were placed to the left of the contents of the MQ. (See Figure 6.)

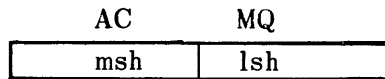


Figure 6 - Representation of the Product

If you multiply 4 by 6, the answer 24 will appear, as in Figure 7. The sign of the MQ is the same as the sign of the AC. The number 24 appears entirely in the MQ because each register is capable of holding an integer as large as 34,359,738,367 plus a sign.

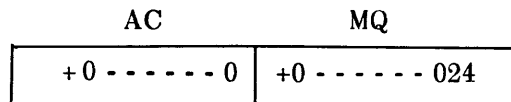


Figure 7 - Product of 6 x 4

Now you must take the contents of the AC and the MQ and place them somewhere in storage. As before, use

STO Y

to place the contents of the AC in location Y.

To place the contents of the MQ in some storage location, use the instruction "Store MQ"

STQ Y

This will place the contents of the MQ in location Y.

Example: Assume that +972 is in location 200 and +852 is in location 201. Find the product of these two numbers, and place the *msh* of the product in location 300 and the *lsh* in location 301. You would do this as shown in Figure 8.

IBM Data Processing Division

SHARE 704 Symbolic Coding Form

Problem															
Coder John Smith										Date September 15, 1959		Page 1 Of 1			
H	Location					Op	Address, Tag, Decrement					Comments		Identification	
1	2	6	7	8	10	11	12	13	14	15	16	17	18	19	20
(Initial Control Cards)															
						ORG	100						Specifies 100 as the location of the first instruction.		
						LDQ	200						Places c(200), contents of 200, in the MQ.		
						MPY	201						Multiplies c(MQ) by c(201).		
						STO	300						Stores msh product in 300.		
						STQ	301						Stores lsh product in 301.		
(Final Control Card #1)															
						ORG	200						Specifies 200 as the location of the first data item.		
						DEC	972						Places 972 in location 200.		
						DEC	852						Places 852 in location 201.		
(Final Control Cards #2-11)															

Figure 8 – Program to Evaluate the Product of Two Numbers

After the operations are performed, location 300 will contain all zeros and location 301 will contain +828144.

DIVISION

To divide, use both the AC and the MQ registers. The instruction for division is

DVH Y

This instruction treats the contents of the AC and the MQ as the dividend, the MQ containing the lsh, and the AC containing the msh and the sign (Figure 9).

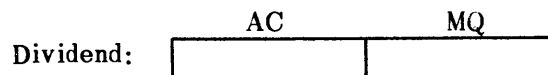


Figure 9 – Representation of the Dividend

Then if the absolute value of the contents of Y is greater than the absolute value of the contents of the AC, i.e., $|c(Y)| > |c(AC)|$, division takes place. The quotient and its sign replace the contents of the MQ, and the remainder replaces the contents of the AC.

Suppose you have 0 in location 200, 55 in location 201, and 11 in location 202, and you want to divide 55 by 11 and then store the quotient in location 300 and the remainder in location 301. If you begin the instructions at location 100, your program would look thus. (See Figure 10.)

Problem														
Coder John Smith						Date September 15, 1959			Page 1 Of 1					
H	Location					Op	Address, Tag, Decrement					Comments	Identification	
1	2	6	7	8	10	11	12					72	73	80
(Initial Control Cards)														
						ORG	100					Specifies location of first instruction.		
						CLA	200					Places 0 in the AC.		
						LDQ	201					Places 55 in the MQ.		
						DVH	202					Places quotient in the MQ and remainder in the AC.		
						STQ	300					Stores quotient of 5 in 300.		
						STO	301					Stores remainder 0 in 301.		
(Final Control Card #1)														
						ORG	200							
						DEC	0							
						DEC	55							
						DEC	11							
(Final Control Cards #2-11)														

Figure 10 – Program to Evaluate the Quotient of Two Numbers

SUMMARY – Chapter II

The new orders learned in this chapter are:

- LDQ Y** **Load MQ**
 Replaces the contents of the MQ by the contents of Y, leaving Y unchanged.
- STQ Q** **Store MQ**
 Places contents of the MQ in Y, and leaves MQ unchanged.
- MPY Y** **Multiply**
 Multiplies the contents of the MQ by the contents of Y and puts the msh and the sign of the product in AC and the lsh of the product in MQ, and leaves Y unchanged.
- DVH Y** **Divide or Halt**
 Treats the AC and MQ as the dividend and then divides this by the contents of Y and puts the signed quotient in MQ and the signed remainder in AC. The sign of the remainder always agrees with the sign of the quotient. Division takes place only if $|c(Y)| > |c(AC)|$. If $|c(Y)| \leq |c(AC)|$ the computer stops.

EXERCISES – Chapter II

1. Find the average of five integers, each less than 1,000. Begin your instructions at location 100 and your data at 200. Place quotient in 300 and remainder in 301. Assume that the dividend will fit entirely in the MQ.

2. Evaluate the following:

$$\sum_{i=1}^5 \frac{x_i^2}{y_i}, \text{ where } \begin{array}{l} x_i = 2, 3, 4, 5, 6; i = 1, 2, \dots, 5 \\ y_i = 2, -3, 1, 5, 4; i = 1, 2, \dots, 5. \end{array}$$

Begin your instructions at location 100 and data at 200. Place result in 300.

3. Punch cards for Exercises 1 and 2 and then run the problems on the computer. Check your answers.

CHAPTER III

INDEX REGISTERS AND THEIR USE

Suppose you have 100 different numbers beginning at location 200, and you want to find the sum of these numbers. It certainly would be tedious to write 99 ADD orders. To facilitate operations such as this, the 704 has some special instructions.

All 704 instructions with memory references are classed into two groups: *indexable*, and *nonindexable*. The nonindexable instructions contain the letter X in their operation code whereas the indexable instructions do not. To understand how these instructions work, it is necessary to know how the 704 picks up and interprets an instruction.

The 704 has an instruction location counter which tells the computer from which location to pick up the next instruction. The computer then places this instruction in the storage register (SR) and restores the instruction unchanged into its original storage location. Then the computer places the operation part of the instruction in the instruction register and leaves the rest of the instruction in the SR. If the instruction register contains an instruction which is not indexed, such as CLA 100, the computer will execute this instruction without further delay. In this case, it will clear and add $c(100)$ to the accumulator. But if, on the other hand, the instruction register contains an indexed instruction, the computer will modify the address of the instruction before execution. This is explained as follows.

If you associate an index register (of which the 704 has three, designated 1, 2, and 4) with an indexable instruction, the instruction is *tagged* with the index register named. For instance, if you *tag* the instruction CLA 100 with the index register 1 (IR-1), write it as follows:

CLA 100,1

Thus, you associate with this instruction the index register 1. Suppose now that the index register 1 contains the number 10.

When the computer picks up the instruction CLA 100,1, it will, as explained above, place the operation part in the instruction register and the address and tag in the storage register. Since this instruction is tagged with index register 1, the computer will first subtract the contents of the index register (namely, 10) from the address in the storage register and then execute the instruction with the new address. Thus the computer will execute

CLA 90

That is, the contents of location 90 (instead of location 100) will now replace the contents of the accumulator. This is called *effective address modification*.

The next two instructions explain how to load an index register with a number, and how to change this number once it is in the index register.

To load an index register with a number, you must first have the number stored somewhere in the core storage. Then the instruction "Load Index from Address"

LXA Y,N

will load the contents of location Y into index register N, where N is 1, 2, or 4.

For example, if you have the number 10 in location 200, the following instruction

LXA 200,1

will load the number 10 into index register 1.

To change the contents of the index registers, the 704 uses control instructions, which have, besides an address and a tag, a third part called a *decrement*. One of these instructions is "Transfer on Index"

TIX, Y, N, D

where Y represents the address, N the tag, and D the decrement.

This instruction operates as follows. If the contents of the index register N are greater than the decrement D, the computer will reduce the contents of the index register by the amount of the decrement and take the next instruction from location Y. Otherwise, the computer proceeds to the next instruction in sequence.

These instructions are illustrated by the following example.

Find the sum of ten numbers stored in consecutive locations beginning at location 200, and store the result in 210. Begin the instructions at location 100.

The program for this problem is in Figure 11. In the column for Comments, an arrow (→) indicates "replaces" and c(u) indicates contents of unit u.

Problem														
Coder John Smith						Date September 15, 1959			Page 1 Of 1					
H	Location					Op	Address, Tag, Decrement					Comments	Identifi- cation	
1	2	6	7	8	10	11	12	1st time around loop			2nd time around loop		----- 9th and last time around loop	
(Initial Control Cards)														
						ORG	100	Says place instructions in core beginning at location 100.						
						LXA	200, 1	9 → c(IR-1).						
						CLA	200	c(200)=9 → c(AC).						
						ADD	210, 1	c(AC)+c(201) → c(AC)			c(AC)+c(202) → c(AC)		----- c(AC)+c(209)	
								=9+2 → c(AC).			=9+2+4 → c(AC).		→ c(AC)=9	
												+2+4+3-4+5		
												+8-7+12-13		
												=19 → c(AC).		
						TIX	102, 1, 1	Reduces c(IR-1) to 8 and proceeds to 102.			Reduces c(IR-1) to 7 and proceeds to 102.		----- Since c(IR-1) is now 1, computer takes next instruction in sequence.	
						STO	210						Sum → c(210).	
(Final Control Card #1)														
						ORG	200	Places the following data in core storage beginning at location 200.						
						DEC	9							
						DEC	2							
						DEC	4							
						DEC	3							
						DEC	-4							
						DEC	5							
						DEC	8							
						DEC	-7							
						DEC	+12							
						DEC	-13							
(Final Control Cards #2-11)														

Figure 11 – Program to Find the Sum of Ten Numbers Using an Index Register

SUMMARY – Chapter III

A nonindexable instruction contains an X in its operation code, but an indexable instruction does not.

If you tag an indexable instruction with an index register, the computer uses effective address modification on this instruction.

The following new instructions were introduced:

Instruction	Address	Tag	Decrement
LXA Y, N			
	Load Index from Address		
	Loads index register N with number stored in location Y, where N = 1, 2, or 4.*		
TIX Y, N, D			
	Transfer on Index		
	If the contents of the index register N are greater than the decrement D, the computer will reduce the contents of the index register by the amount of the decrement and take the next instruction from location Y. Otherwise, the computer proceeds to the next instruction in sequence.		

*A tag number 3, 5, 6, or 7 is also used. For their meaning, see Reference 1, page 8.

EXERCISES – Chapter III

1. Do Exercise 1 of Chapters I and II using an index register.
2. Given that a_i , b_i , and c_i ($i = 1, 2, \dots, 10$) are 30 integers beginning at location 200, find X as defined below and place the value of X in 300. Begin the instruction at 100.

$$X = \left(\sum_{i=1}^{10} (a_i c_i)^2 \right) \left(\sum_{i=1}^{10} (b_i - c_i) \right)$$

3. Assume, in Exercise 2, that the first nine a 's are the elements of a 3×3 matrix A , stored row-wise, and that the first nine b 's are the elements of a 3×3 matrix B , also stored row-wise. Find the elements of the product $AB = C$. Store the c_{ij} columnwise beginning at location 300. Begin the instructions at 100.
4. Prepare data and instruction cards for Exercises 2 and 3, and then run the problems on the machine. Check your answers by hand computation.

CHAPTER IV

FLOW CHARTS

In the first three chapters, you learned what the instruction codes for the arithmetic operations are, how to index an instruction, and what the meaning of one control instruction is. In this chapter, you will study three more control instructions and an aid to programmers—flow charting.

The three new control operations are:

TRA Y	Transfer
	Transfers control to Y. The computer will take its next operation from location Y and continue from there.
TNX, Y, N, D	Transfer on No Index
	If contents of index register N are greater than the decrement D, the computer reduces the contents of index register N by D and takes the next instruction in sequence. If contents of index register N are less than or equal to D, the computer leaves the IR as it is and takes the next instruction from location Y.
TXI, Y, N, D	Transfer with Index Incremented
	Adds D to the contents of index register N and places this sum in index register N. Then the computer takes the next instruction from location Y.

The following example will illustrate how you may use these orders. Note that the pseudoinstruction TRA FINISH signifies the end of your program and prints the output on tape 9, from which you obtain a printed copy via the printer. This card is always placed after your last instruction. It is Final Control Card No. 1. Previously, you have included this instruction in the final control cards. Now this card in the control deck will be omitted and you are to place it with your own instructions.

Assume that you have ten integers stored in consecutive locations beginning at 200. Find the sum of the first two numbers and store this sum in 300; then find the sum of the next two numbers and store this sum in 301. Continue this procedure until you have added all pairs of numbers.

The program in Figure 12 will accomplish this.

When writing a program, you must write the instructions for the machine in a particular order, according to what you want to do. You could write down in sequence what you desire to do and then write the machine instructions that will accomplish each step. However, a diagram is usually easier to use, so resort to this method.

Problem											
Coder		Date				Page		Of		Page	
John Smith		September 15, 1959				1		1			
H	Location	Op	Address, Tag, Decrement	Comments							
1	2	3	4	5	6	7	8	9	10	11	12
				1st time around loop	2nd time around loop	3rd time around loop	4th time around loop	Last time around loop			
	(Initial	Control	Cards)								
		ORG	100	Sets location counter to 100.							
		LXA	108, 2	5 → c(IR-2).							
		LXA	109, 1	10 → c(IR-1).							
		CLA	210, 1	c(200) → c(AC).	c(202) → c(AC).	c(204) → c(AC).	c(206) → c(AC).	c(208) → c(AC).			
		ADD	211, 1	c(200) + c(201)	c(202) + c(203)	c(204) + c(205)	c(206) + c(207)	c(208) + c(209)			
				→ c(AC).	→ c(AC).	→ c(AC).	→ c(AC).	→ c(AC).			
		STO	305, 2	c(200 + c(201)	c(202) + c(203)	c(204) + c(205)	c(206) + c(207)	c(208) + c(209)			
				→ c(300).	→ c(301).	→ c(302).	→ c(303).	→ c(304).			
		TNX	107, 2, 1	Subtracts 1 from IR-2, i.e., 4 → c(IR-1), then calculator goes to next instruction.	Subtracts 1 from IR-2, i.e., 3 → c(IR-2), then calculator goes to next instruction.	Subtracts 1 from IR-2, i.e., 2 → c(IR-2), then calculator proceeds to next instruction.	Subtracts 1 from IR-2, i.e., 1 → c(IR-2), then calculator proceeds to next instruction.	Since IR-2 contains one, the calculator proceeds to end routine.			
		TIX	102, 1, 2	8 → c(IR-1), then calculator proceeds to 102.	6 → c(IR-1), then calculator proceeds to 102.	4 → c(IR-1), then calculator proceeds to 102.	2 → c(IR-1), then calculator proceeds to 102.				
		TRA	FINISH								
		DEC	5								
		DEC	10								
		ORG	200								
		DEC	578								
		DEC	-2								
		DEC	-457								
		DEC	3								
		DEC	812								
		DEC	75236								
		DEC	-851								
		DEC	+752								
		DEC	1256								
		DEC	9234								
	(Final	Control	Cards #2-11)								

Figure 12 – Program to Place Sum of Each Pair of Numbers in Consecutive Locations

1. Path of Computational Flow

To indicate “the path of computational flow,” use a directed arrow. (→).

2. Logical Choice

A logical choice between two directions of a path, depending on the magnitude of two quantities A and B, is indicated by an oval, as shown in Figure 13:

3. Computation, Operation, Etc.

The evaluation of a formula or other expression is indicated by a rectangular box. See Figure 14.

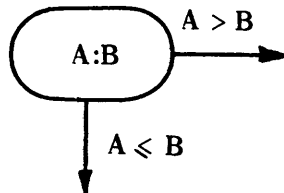


Figure 13 – A Logical Choice

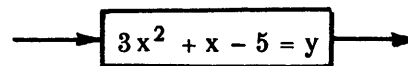


Figure 14 – An Evaluation of a Formula

4. Counting Operations

When you wish to increase the value of a subscripted quantity, use a rectangular box with a double line at the left end. See Figure 15.

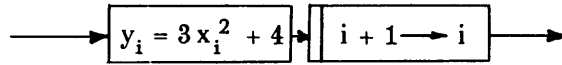


Figure 15 – Counting Symbol

Note that many programmers use the plain rectangular box as the counting symbol.

5. Connectors

To connect one part of a flow chart to another part, you can draw an arrow to that part. But at times it is more convenient to use what is called a connector. This is indicated by a circle with a number or other symbol in it (Figure 16).

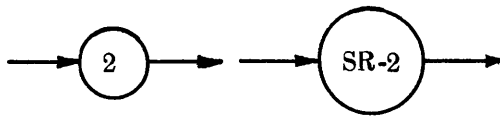


Figure 16 – A Connector

6. Assertions and Notes

To assert that a certain condition is true at a particular time or to make a note for later reference, use a rectangular flag attached to the flow line.

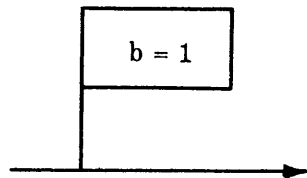


Figure 17 – An Assertion or Note

The flow chart in Figure 18 and program in Figure 19 illustrate the use of these symbols, in solving the problem given on page 23. Notice that each figure in the flow chart is labeled and certain instructions are associated with it in the program.

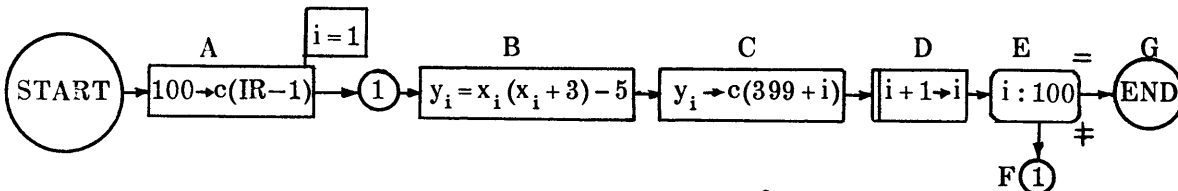


Figure 18 – Flow Chart to Evaluate $y_i = (x_i^2 + 3x_i - 5)$, $i = 1, 2, \dots, 100$

Problem											
Coder John Smith						Date September 15, 1959			Page 1 Of 1		
H	Location	Op	Address, Tag, Decrement			Comments			Identifcation		
1	2	6	7	8	10	11	12	13	14	15	16
						1st time around loop	2nd time around loop	Last time around loop			
(Initial Control Cards)											
		ORG	100								
A		LXA	302,1 100→c(IR-1).								
B		CLA	300,1 $x_1 \rightarrow c(AC)$.			$x_2 \rightarrow c(AC)$.	-----	$x_{100} \rightarrow c(AC)$.			
		ADD	301 $(x_1+3) \rightarrow c(AC)$.			$(x_2+3) \rightarrow c(AC)$.	-----	$(x_{100}+3) \rightarrow c(AC)$.			
		STO	302 $(x_1+3) \rightarrow c(302)$.			$(x_2+3) \rightarrow c(302)$.	-----	$(x_{100}+3) \rightarrow c(302)$.			
		LDQ	302 $(x_1+3) \rightarrow c(MQ)$.			$(x_2+3) \rightarrow c(MQ)$.	-----	$(x_{100}+3) \rightarrow c(MQ)$.			
		CLA	303 0→c(AC).			0→c(AC).	-----	0→c(AC).			
		MPY	300,1 $x_1(x_1+3) \rightarrow c(MQ)$.			$x_2(x_2+3) \rightarrow c(MQ)$.	-----	$x_{100}(x_{100}+3) \rightarrow c(MQ)$.			
		STQ	302 $x_1(x_1+3) \rightarrow c(302)$.			$x_2(x_2+3) \rightarrow c(302)$.	-----	$x_{100}(x_{100}+3) \rightarrow c(302)$.			
		CLA	302 $x_1(x_1+3) \rightarrow c(AC)$.			$x_2(x_1+3) \rightarrow c(AC)$.	-----	$x_{100}(x_{100}+3) \rightarrow c(AC)$.			
		ADD	300 $x_1(x_1+3)-5 \rightarrow c(AC)$.			$x_2(x_2+3)-5 \rightarrow c(AC)$.	-----	$x_{100}(x_{100}+3)-5 \rightarrow c(AC)$.			
C		STO	500,1 $x_1(x_1+3)-5 \rightarrow c(400)$.			$x_2(x_2+3)-5 \rightarrow c(401)$.	-----	$x_{100}(x_{100}+3)-5 \rightarrow c(499)$.			
D, E, F		TIX	101, 1, 1 Reduces c(IR-1) by one; then computer proceeds to 101. c(IR-1) is now 99.			Reduces c(IR-1) by one; then computer proceeds to 101. c(IR-1) is now 98.	-----	Since c(IR-1) = 1, computer goes to next instruction.			
G		TRA	FINISH					Transfers to ending routine.			
		ORG	300					Stores constants beginning at 300.			
		DEC	-5								
		DEC	3								
		DEC	100								
		DEC	0								
(Final Control Cards #2-11)											

Figure 19 – Program Associated with the Flow Chart of Figure 18

Find the value of each y_i in the following expression, and store each y_i beginning at location 400. Assume that all products will not exceed the capacity of the MQ. Store the x_i 's at location 200; place the constant -5 in 300 and the constant 3 in 301. Start the instruction in location 100.



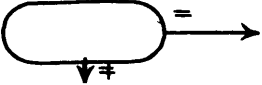


$$y_i = (x_i^2 + 3x_i - 5), i = 1, 2, \dots, 100.$$

SUMMARY – Chapter IV

In this chapter you learned the following instructions and flow charting.

TRA Y	Transfer Transfers control to location Y.
TNX Y, N, D	Transfer on No Index If contents of index register N are greater than the decrement D, the computer reduces the contents of index register N by D and takes the next instruction in sequence. If contents of index register N are less than or equal to D, the computer leaves the IR as it is and takes the next instruction from location Y.
TXI, Y, N, D	Transfer with Index Incremented Adds D to the contents of index register N and places this sum in index register N. Then the computer takes the next instruction from location Y.
TRA FINISH	Transfer to FINISH Signifies the end of your program and prints output on tape 9.

The following symbols are used in flow charts:

1.  path of computational flow
2.  computation box
3.  logical choice
4.  counting operations
5.  connectors

EXERCISES – Chapter IV

In the following exercises, draw a flow chart first and then code each symbol of the flow chart.

1. Fifty numbers (x_i , $i = 1, 2, \dots, 50$) are stored in locations 301, 302, \dots , 350, and fifty others (y_i , $i = 1, 2, \dots, 50$) are stored in 351, 352, \dots , 400. Place $x_i - y_i$, $i = 1, 2, \dots, 50$ in locations 201–250.

2. Given the instruction

TPL Y Transfer on Plus

The computer takes the next instruction from location Y if the sign of the quantity in the accumulator is positive. If the sign of the quantity in the A is negative, the computer proceeds to the next instruction.

do the following problem:

Fifty numbers are stored in locations 201–250. Place the sum of the positive ones in 300 and the sum of the negative ones in 301.

CHAPTER V

AUTOMATIC PROGRAMMING

A machine program, as previously defined, is a list of instructions written in machine language and arranged in proper sequence. Since the machine will do exactly what it is told to do, these instructions must be precise and correct to the last detail. For this reason and because machine language is totally dissimilar to English, machine programming is extremely time-consuming and error-prone.

One method used to help alleviate these problems is *symbolic programming*. In this system, the programmer uses symbolic operation codes and addresses. Then a machine master program, called an *assembly* or *compiler*, translates these symbolic codes into machine language, and either writes them onto a tape or punches them on cards which then can be used in the usual manner.

The symbolic assembly program discussed next is called *SAP 3-7*.

When you refer to an instruction or a piece of data by giving its location, you must know exactly where this instruction or piece of data is located in the core storage. With a large program, this would be difficult and tedious to do, so instead you may use *symbolic addresses*. That is, in place of a numerical address, use a symbol in the address part of an instruction. The symbol may be from one to six consecutive characters, at least one of which is an alphabetic character. The other characters may be numbers or letters. For example, A1, 2C, DATA 1, 34A 2, 12345 X, L, are all acceptable symbols.

To refer to the n^{th} instruction after the current instruction use the following scheme when using symbolic addresses.

First note the meanings of the following 704 characters:

- + denotes addition,
- denotes subtraction,
- * denotes multiplication, and
- / denotes division.

You may use these arithmetic characters in the address, tag, or decrement of an instruction. For example, suppose that you have ten numbers beginning at symbolic location DATA. If you want to place the seventh one into the accumulator, you could use

CLA DATA + 6

The character* is used not only to denote multiplication but also to signify the current location. It means the current location if it is the first character in the symbolic address. For example, if you have a transfer instruction at location 200, which transfers control to location 203, you could write it thus:

TRA *+ 3

The computer will then pick up the next instruction from location 203.

The following example illustrates the use of these symbols. The following pseudo-instruction "Block Reservation"

M BSS N

is introduced here. It will reserve N consecutive core units beginning at location M, so that the programmer may store data or results in consecutive locations.

Find the sum of ten integers which are stored in consecutive memory locations, beginning at location M. Begin your instructions in symbolic location L. Store the sum in symbolic location SUM. The program for this problem is given in Figure 20.

IBM Data Processing Division **SHARE 704 Symbolic Coding Form**

Problem										
Coder John Smith						Date September 15, 1959		Page 1		Of 1
H	Location			Op	Address, Tag, Decrement			Comments		Identification
1	2	6	7	8	10	11	12	72	73	80
	(Initial Control Cards)									
				ORG	100					
	L			LXA	NINE, 1			Loads IR-1 with 9.		
				CLA	M			c(M) → c(AC).		
				ADD	M+10, 1			c[M+10-c(IR#1)] + c(AC) → c(AC).		
				TIK	*-1, 1, 1			If not the last number to be added, transfers to previous instruction. If last number, goes to next instruction.		
				STO	SUM			Stores the sum in SUM.		
				TRA	FINISH			Transfers to END Routine.		
	NINE			DEC	9			Stores the number 9 in location nine.		
	M			DEC	N₁			Begins storing data at location M.		
				DEC	N₂			N_i (i = 1, 2, ..., 10) are integers.		
				.						
				.						
				.						
				.						
				DEC	N₁₀					
	SUM			BSS				Reserves 1 core unit for the sum.		
	(Final Control Cards #2-11)									

Figure 20 – Symbolic Program to Find the Sum of Ten Numbers

Note that every time you use a symbol in the address, tag, or decrement field, that symbol must be defined in some location field.

If you know the absolute address of a piece of data, you can, if you wish, use this absolute number when referring to the data.

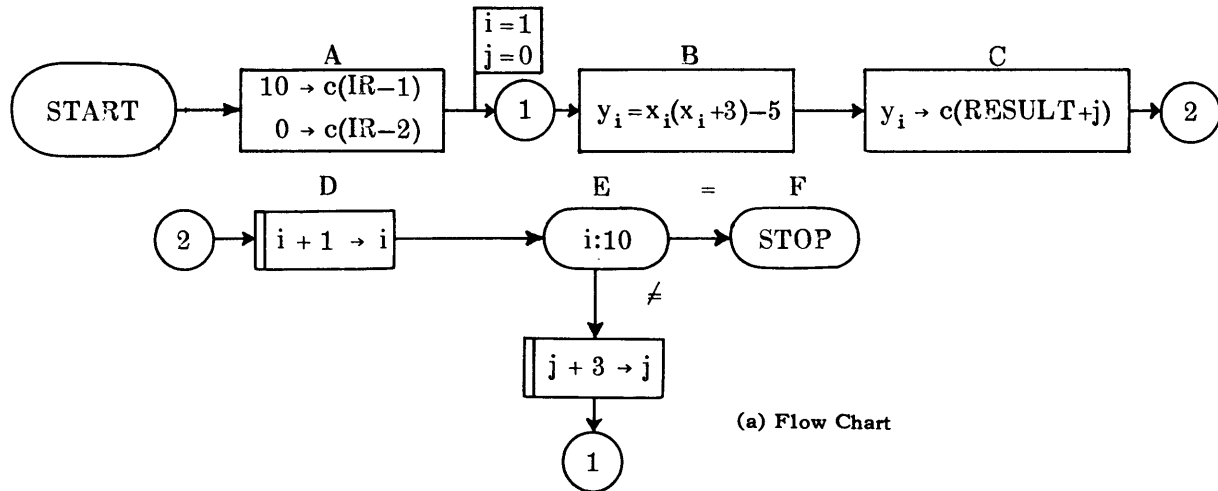
Here is another example, similar to the last example in Chapter IV.

Find the value of each y_i in the following expression, and store y_i in every third location beginning at location RESULT. Assume that all products will not exceed the capacity of the MQ. Begin data at location DATA

$$y_i = (x_i^2 + 3x_i - 5), i = 1, 2, \dots, 10$$

In solving this problem, you will illustrate the use of an index register to increase rather than to decrease the address of an instruction. Since the computer always subtracts the contents of the index register from the address of the instruction tagged with that register, you need only to make provision for the index register to contain a negative integer. Do this by initially setting the contents of the index register equal to zero, and then by using a negative integer in the decrement part of a TXI instruction used to modify the index register.

The flow chart is Figure 21a, and the program is Figure 21b.



(a) Flow Chart

IBM Data Processing Division SHARE 704 Symbolic Coding Form

Problem												
Coder John Smith						Date September 15, 1959			Page 1		Of 1	
H	Location	Op	Address, Tag, Decrement				Comments			Identification		
1	2	6	7	8	10	11	12	71	72	80		
(Initial Control Cards)												
A	LXA	TEN, 1	10 -> c(IR-1).									
	LXA	ZERO, 2	0 -> c(IR-2).									
B	CLA	DATA+10, 1	x_i -> c(AC).									
	ADD	THREE	(x_i+3) -> c(AC).									
	STO	TEMP	(x_i+3) -> c(TEMP).									
	LDQ	TEMP	(x_i+3) -> c(MQ).									
	MPY	DATA+10, 1	x_i(x_i+3) -> c(AC) and c(MQ).									
	STQ	TEMP	x_i(x_i+3) -> c(TEMP).									
	CLA	TEMP	x_i(x_i+3) -> c(AC).									
	ADD	NFIVE	[x_i(x_i+3)-5] -> c(AC).									
C	STO	RESULT, 2	[x_i(x_i+3)-5] -> c(RESET+j).									
DEF	TNX	FINISH, 1, 1	If last computation, goes to FINISH.									
			If not, goes to next instruction.									
G	TXI	B, 2, -3	Adds 3 to IR-2, then goes to B.									
TEN	DEC	10										
ZERO	DEC	0										
DATA	DEC	78										
Reserves 10 core unit beginning at DATA.												
	DEC	65										
THREE	DEC	3										
TEMP	BSS	1	Reserves 1 core unit for TEMP.									
NFIVE	DEC	-5										
RESULT	BSS	10	Reserves 10 core units beginning at RESULT.									
(Final Control Cards)												

(b) Program

Figure 21 - Symbolic Program to Evaluate $y_i = (x_i^2 + 3x_i - 5)$, $i = 1, 2, \dots, 100$

When you assemble this program, the symbolic assembly program, or SAP 3-7, will change all symbolic addresses to absolute addresses and will prepare a program on tape 4 and on punched cards ready to be run on the computer. At the same time, SAP will cause the computer to print on tape 9 an assembly listing which consists of the symbolic program and the octal location of each of these instructions.

SUMMARY – Chapter V

In this chapter you began your study of SAP 3–7, an automatic coding system. In SAP 3–7, you use symbolic addresses which consist of one to six consecutive characters, one of which must be alphabetic.

SAP 3–7 also will interpret the arithmetic characters (+, -, *, /) in an address, tag, or decrement as add, subtract, multiply, and divide.

The following pseudoinstruction was introduced:

M BSS N Block Reservation
 Reserves N core units beginning at location M.

EXERCISES – Chapter V

1. Punch the cards for the last example of this chapter and run the example on the computer. Look at the assembly listing and see what locations SAP assigns to each instruction and data item.
2. Do the problems at the end of Chapter IV, using symbolic addresses. Redraw the flow diagrams. Store the answer in core storage, beginning at symbolic location RESULT.

CHAPTER VI

BELL INPUT-OUTPUT SYSTEM

To get information from punched cards into the core storage of the IBM 704 and to get printed results, you have been using certain control cards. Before these cards can be explained, certain details about the computer and its operation must be mentioned.

The Applied Mathematics Laboratory of the David Taylor Model Basin uses for the IBM 704 an operating system developed by the Bell Telephone Laboratories, and known as BE SYS 2. This system is itself a computer program designed to control input-output and other general operations. It is stored in locations 0-23 and 28,672-32,768. Thus these locations are not available for use by the programmer. Since the IBM 704 operates internally in the binary number system, it is more convenient to refer to locations in the octal number system (see Appendix C) rather than the decimal system. Thus, in the octal system then, locations 0-27 and 70000-77777 are reserved for the Bell System.

Here at the Applied Mathematics Laboratory the 704 has ten magnetic tape units numbered 1-10. Information can be transferred from core storage to tape, and vice versa. Tapes 1, 2, and 9 are reserved for use by the Bell System; tapes 3, 4, and 5 are used for SAP 3-7 and FORTRAN, an automatic coding system which will be explained in Chapter XIII. The other tape units are available for use by the programmer.

The Bell System is activated by the reading of control cards. On these control cards are punched the *pseudo-operations* (defined below) in exactly the same way as SAP instructions are punched; i.e., the operation code is punched in columns 8-10; the location, in columns 1-6; and the address, tag, and decrement, in columns 12-72.

Notice that on these control cards there is a comment following the address, tag, and decrement. (The address, tag, and decrement are often called the *variable field*.) However, the machine does not interpret the comment as part of the instructions, since this comment is separated from the variable field by a space.

JOB (Initial Control Card No. 1)

JOB prepares the system for execution of the program to be performed and causes the computer to read the next card.

This is the first card in performing any job.

SAP (Initial Control Card No. 2)

This card loads SAP into core storage and executes an assembly as explained in the previous chapter. If the assembly is successful, it translates the SAP program into machine language and places the result on tape 4 and on punched cards, which may be reserved for

rerunning the problem.* When an unsuccessful assembly occurs, the instruction SAP causes information to be printed, telling why the assembly failed, and then causes the computer to stop or to transfer control to the next JOB card if one exists. (See Reference 3.)

This card is placed after the JOB card.

END "START" (Final Control Card No. 9)

This signifies the end of the program, where "START" is the symbolic location of the first instruction to be executed in the program.

This card is placed at the end of the symbolic deck.

LOD 4 (Final Control Card No. 10)

This instruction loads the translated program which has been written on tape 4 into core storage and causes the computer to read the next card.

This card is placed after the END card.

TRA (Final Control Card No. 11)

This card transfers control to the location "START" specified in the END card of the program if the assembly is successful. That is, the computer begins now to execute the assembled program.

This card is placed after the LOD 4 card.

The above instructions leave Final Control Cards Nos. 2-8 yet to be explained, but this will be done in the next chapter. These unexplained cards should be inserted just before the END card.

*For a description of the binary cards and their use, see Chapter X.

SUMMARY – Chapter VI

To run a SAP program on the 704, the Bell System is used for input-output operations. The following pseudoinstructions are punched on cards and used in the sequence given in Figure 22.

Problem											
Coder John Smith						Date September 15, 1959			Page 1		Of 1
H	Location		Op		Address, Tag Decrement			Comments		Identifi- cation	
1	2	6	7	8	10	11	12	72	73	80	
				JOB				Initializes the system.			
				SAP				Translates instructions and writes them on tape 4 and cards.			
					(Object Program)						
					(Final Control Cards #2-8)						
				END				Ends translation.			
				LOD		4		Loads translated instructions into core storage.			
				TRA				Causes translated instructions to be executed beginning in location "START".			

Figure 22 – Order of Instructions to Run a Symbolic Program Using SAP 3-7

EXERCISES – Chapter VI

1. Write a symbolic program to evaluate the following polynomial:

$$y_i = ax_i^5 + bx_i^4 + cx_i^3 + dx_i^2 + ex_i + f; i = 1, 2, \dots, 10$$

First draw a flow chart. Assume small integer values for x and the constants. Begin the instructions and data at any convenient location and store the answers beginning at symbolic location **RESULT**. Then punch the necessary cards and run the problem on the 704. Check your answers with the machine and if they are not correct, then correct your program.

2. Write a symbolic program to solve by determinants the following two equations. Store the answers beginning at location **RESULT**.


$$+ 7x - 5y = 9$$

$$+ 3x + 4y = 10$$

CHAPTER VII

SUBROUTINES

In mathematical problems, functions such as the square root of a number, the cosine of an angle, etc. frequently have to be evaluated. The manufacturer of a computer could make an instruction that would evaluate any one of these. However, to build into a computer all the functions that one might need would be prohibitive in cost and would require a machine of enormous size. Therefore, *subroutines* are used, as explained below, to evaluate any special functions which are needed and are not provided as instructions in the computer.

By using approximations involving only the four arithmetic operations, a program can be written to evaluate such functions. For example, a program can be written to evaluate trigonometric or logarithmic functions by use of Chebyshev polynomial approximations. Such a program could be saved on tape or cards and used whenever a problem requires the evaluation of this function. A prewritten program used in this way is called a *subroutine*. Subroutines are used not only for special mathematical functions but also for many other operations that are frequently needed, such as input-output. The flow chart symbol for a subroutine is: 

For practical use of a subroutine, a set of pseudoinstructions, or the *calling sequence*, is used to incorporate the subroutine into the main routine. This is best illustrated by an example.

In the exercises of the previous chapters, Final Control Cards Nos. 2–8 have been causing the computer to write the results on tape 9. These cards contain the calling sequence of the print subroutine, which is in core storage as part of the Bell System. In the Bell System instructions, the calling sequence of this subroutine is described as follows. (See Final Control Cards Nos. 3, 4, 5, 7, and 8. Final Control Card No. 2 is ORG 500, which stores the calling sequence beginning at location 500. Final Control Card No. 6 is explained later.)

TSX OUTPUT, 4 (Final Control Card No. 3)
NTR F, , N (Final Control Card No. 4)
MON A, , B (Final Control Card No. 5)
Normal Return

The Bell System instructions further state that this routine will write on tape N a consecutive block of words from core locations A to B, inclusive, where F is the location of the format of the printout desired. The following format was used in the Final Control Cards of the exercises to print integers (see Reference 2):

F BCD 1(N 13) (Final Control Card No. 7)
SVN -1, 7, -1 (Final Control Card No. 8)

In the exercises, N was set equal to 9 because the Bell System uses tape 9 for all output; thus the result will be printed after the SAP listing.

The last line of the calling sequence is called the *normal return*. This means that after the integer is written on tape the computer will take its next instruction from this line. In the Final Control Cards for the exercises, the instruction used as the normal return was (see Final Control Card No. 6):

TRA ENDJOB

This is a Bell System pseudocode which signifies the end of the problem and writes on tape 9 the words *Post-Mortem Dump* followed by the contents of the index registers, the accumulator, the MQ register, and other information to be explained later.

Since all the control cards have now been explained, a complete program illustrating their use will be given. Find the sum of two numbers, a and b, and print the sum on tape 9. The flow chart is in Figure 23, followed by the program in Figure 24.

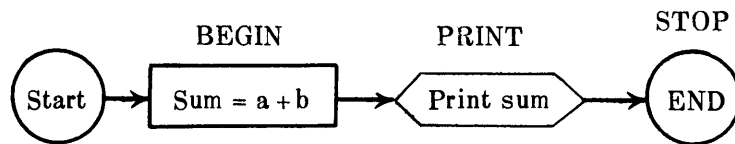


Figure 23 – Flow Chart to Evaluate the Sum of Two Numbers

IBM Data Processing Division												SHARE 704 Symbolic Coding Form											
Problem												Date September 15, 1959											
Coder John Smith												Page 1 Of 1											
H	Location					Op	Address, Tag, Decrement					Comments					Identification						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22		
						JOB																	
						SAP																	
						ORG																	
						BEGIN																	
						CLA																	
						ADD																	
						STO																	
						PRINT																	
						TSX																	
						NTR																	
						MON																	
						STOP																	
						TRA																	
						ENDJOB																	
						DATA																	
						DEC																	
						DEC																	
						SUM																	
						BSS																	
						F																	
						BCD																	
						1(N8)																	
						SYN																	
						-1, 7, -1																	
						END																	
						BEGIN																	
						LOD																	
						4																	
						TRA																	

Figure 24 – Symbolic Program Associated with Flow Chart of Figure 23

The Bell System has a few other subroutines, besides the "PRINT" subroutine used above, as part of its program. In addition, many more prewritten routines are available through an organization called SHARE, made up of installations having IBM 704's, and organized for the purpose of exchanging information about the computer and avoiding duplication of effort in preparing subroutines. Thus, when one installation writes a routine to evaluate a function, that routine becomes available to all other members of SHARE. These subroutines are punched in symbolic form on cards. To use one in a program, the symbolic deck is placed before or after the data, and the calling sequence is placed in the position where the subroutine is to be used.

All symbols in the calling sequence of the subroutine must be defined, and core storage must be reserved for the "COMMON" of each subroutine, as specified in the writeup of that subroutine. For example, the routine shown on Figure 25 specifies that the "COMMON" is COMMON through COMMON+3 or 4 storage units.

The subroutines that are most frequently used are stored on tape rather than on cards because it is easier and quicker to get the subroutine into the core storage from tape than from cards. The Bell System uses Tape 7 as the library tape. To incorporate in the main program a subroutine which is on tape, a library card, rather than a symbolic deck, is used and is punched as follows:

<u>Columns</u>	<u>Data</u>
1-6	SHARE identification number
7	Blank
8-10	LIB
11	Blank
12	7

An example is now given. Suppose it is desired to evaluate $Y = \tan X$.

To find the value of $\tan X$, a SHARE subroutine which is on the Bell System library tape is used. A description of this subroutine is reproduced in Figure 25. Notice that to use this subroutine, X must be expressed in radians and it must be stored in the accumulator in normalized floating-point form before entering the subroutine. Floating-point numbers will be explained in the next chapter. Here it is necessary to know only that the instruction DEC will place a number into core storage in normalized floating-point form if the number is specified with a decimal point in the address field.

The flow chart for this problem is given in Figure 26 and the program is given in Figure 27.

Notice that F here is defined differently than in the previous chapter. This is because the result is not an integer but a decimal number, and the F as defined above is a format for printing numbers with a decimal point.

Another example using subroutines is given and at the same time a new flow chart symbol is introduced, the *variable connector*.

SHARE SUBROUTINE

IDENTIFICATION

Tangent, CL TAN1

R. G. Johnson - 1-7-56

Lockheed Aircraft Corporation, California Division

PURPOSE

Compute: tan X for all X in radians.

Error $X(1+\tan^2 X) (5 \cdot 10^{-8})$.

METHOD

The continued fraction,

$$\tan X = \frac{X}{1 - \frac{X^2}{3 - \frac{X^2}{5 - \frac{X^2}{7 - \dots}}}} \quad \text{is used for } 10^{-4} < |X| < \pi \cdot 2^{26}.$$

If $|X| < 10^{-4}$, tan X is set equal to X.

If $|X| \geq \pi \cdot 2^{26}$, tan X is set equal to zero.

USAGE

<u>LOC</u>	<u>OP</u>	<u>ADDR.</u>	<u>TAG</u>	<u>DECR.</u>
L	TSX	TAN	4	
L+1		Normal Return		

Normalized floating point X, in radians, must be in the AC.

Normalized floating point tan X is in the AC on Normal Return.

STORAGE

65 words plus ~~COMMON~~ through ~~COMMON~~ + 3.

DISCLAIMER

Although this program has been carefully tested by its contributor, no guarantee is made of its correct functioning under all conditions, and no responsibility is taken by him in case of possible failure.

Figure 25 - SHARE Subroutine for Evaluating TAN X

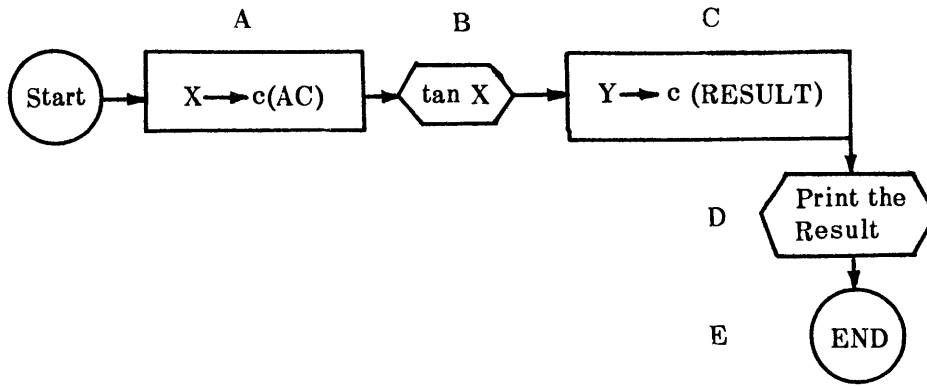


Figure 26 -- Flow Chart to Evaluate the TAN X Using a SHARE Subroutine

IBM Data Processing Division

SHARE 704 Symbolic Coding Form

Problem												
Coder John Smith						Date September 15, 1959			Page 1 Of 1			
H	Location	Op	Address, Tag, Decrement				Comments			Identification		
1	2	6	7	8	10	11	12	72	73	80		
		JOB										
		SAP										
		ORG	100									
	A	CLA	X									
	B	TSX	TAN, 4									
	C	STO	RESULT									
	D	TSX	OUTPUT, 4									
		NTR	F, 9									
		MON	RESULT, , RESULT									
	E	TRA	ENDJOB									
	CLTAN1	LIB	7									
	X	DEC	1. 25									
	RESULT	BSS	1									
	F	BCD	2(F16. 8)									
		SVN	-1, 7, -1									
	COMMON	BSS	4									
		END	A									
		LOD	4									
		TRA										

Figure 27 -- Symbolic Program Associated with Flow Chart of Figure 26

It may happen that a section of coding which is repeated many times calls alternately upon each of two or more subroutines. This situation requires the repetition of the coding of the section before the calling sequence for each subroutine. Such repetition can be avoided by the use of a switch, or *variable connector*. Suppose you have a large program which consists of three parts: main routine A, subroutine B, and subroutine C. And suppose that all parts are long and that A is repeated 50 times. Further, suppose that the sections are done in the order A, B, A, C. Each time after C is done, a test is made to see if the conditions for problem completion are satisfied, in which case the problem is stopped. A suggested flow chart is given in Figure 28. In this flow chart the symbol for the variable connector or switch

is $\delta_{1,2}$. The computer, when it comes to this connector, will go either to δ_1 or δ_2 , according to how δ has been set. The delta switch is merely a transfer instruction. The program associated with this flow chart is given in Figure 29. Notice that after you load index register 1 with N, you set the variable switch δ to δ_1 as follows:

The instruction

CLA DELTA

clears and adds the contents of DELTA, which is the instruction TRA X4. Thus the computer, when it first comes to this instruction, will transfer control to location X4 and continue from there. After the calling sequence for subroutine B, notice that you again set the delta switch, this time to δ_2 . Then the computer, when it comes the second time to the delta switch, will transfer control to location X6, etc.

It is worth noting that the variable connector may connect more than two parts, also that a program may contain several of these variable connectors or switches.

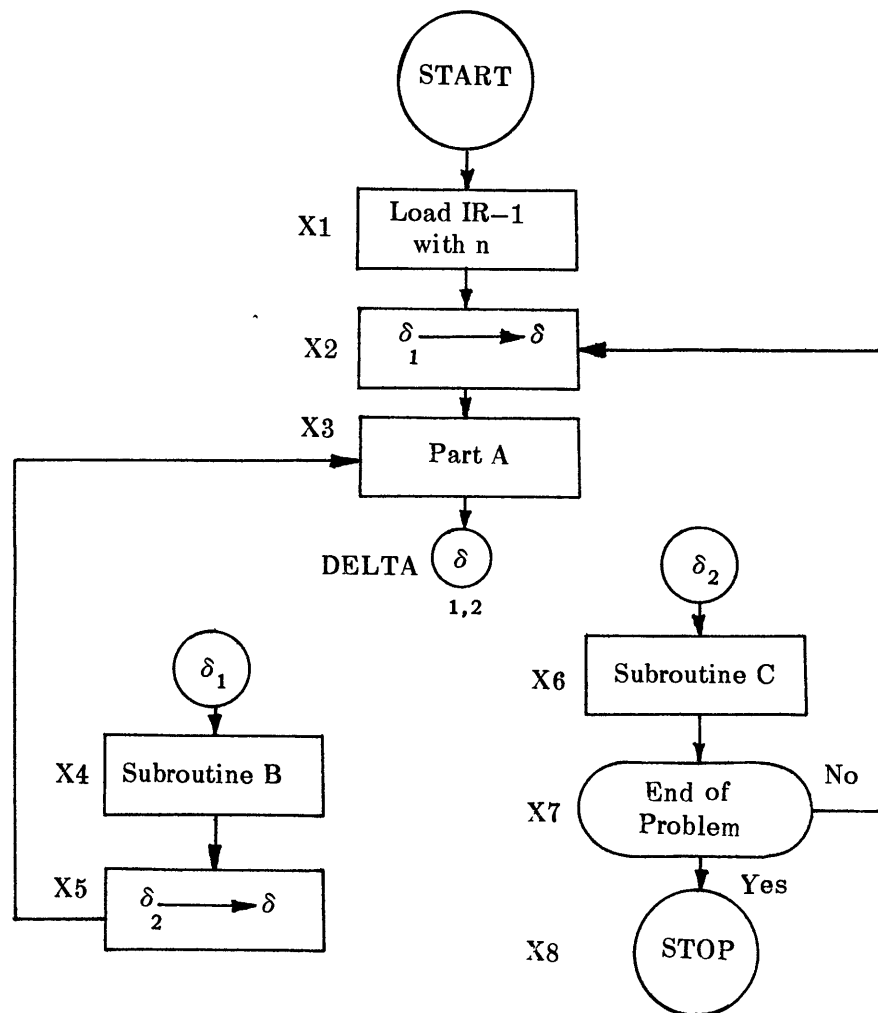


Figure 28 – Flow Chart Demonstrating Variable Connector

Problem											
Coder						Date			Page		Of
John Smith						September 15, 1959			1		1
H	Location			Op	Address, Tag, Decrement			Comments		Identification	
1	2	6	7	8	10	11	12	72	71	80	
				JOB							
				SAP							
				ORG			100				
	X1			LXA			N, 1		Loads index register 1 with n.		
	X2			CLA			DELTA1		Sets switch delta to delta 1.		
				STO			DELTA				
	X3			PART							
				A					Instructions for Part A.		
				()					Space for variable connector.		
	DELTA			Calling							
	X4			sequence					Instructions for Part B.		
				for							
				Subroutine							
				B							
	X5			CLA			DELTA2		Sets switch delta to delta 2.		
				STO			DELTA				
				TRA			X3				
	X6			Calling					Instructions for Part C.		
				sequence							
				for							
				subroutine							
				C							
	X7			TIX			X2, 1, 1		Tests for end of problem.		
	X8			TRA			ENDJOB		Completes the problem.		
	N			DEC			50		Program constant.		
	DELTA1			TRA			X4		Program constant.		
	DELTA2			TRA			X6		Program constant.		
				END			X1				
				LOD			4				
				TRA							

Figure 29 – Program Associated with Flow Chart of Figure 28

SUMMARY – Chapter VII

Subroutines are prewritten routines which are stored either on tape or on cards. To place the program of a subroutine on cards in the main program, the subroutine deck is placed before or after the data. If the subroutine is on tape, a card with the pseudoinstruction LIB must be used. In Columns 1–6 is punched the SHARE identification number, in Columns 8–10, the letters LIB; and in Column 12, the number 7.

To use the subroutine, the calling sequence must be given and all symbols including the COMMON must be defined.

If the subroutine is a part of the Bell System program, the subroutine is already in the core storage; therefore, you need only to define the symbols and use the calling sequence.

A new programming technique was introduced here, the variable connector or switch. This is indicated in a flow chart as $\textcircled{\delta}$, where δ is the name of the connector and n is the number of connections.

1,2,...,n

EXERCISES – Chapter VII

1. Evaluate $Y = \tan X_i$ ($i=1, 2, 3$) for $X_1 = 1.00$ radians, $X_2 = 1.25$ radians, and $X_3 = 1.30$ radians. Use the subroutine reproduced on Figure 25, and notice that this subroutine is on the library tape.

2. The subroutine reproduced on Figure 30 will evaluate \sqrt{X} . However, this subroutine is punched in symbolic form on cards. Find the square root of the following three numbers: 12, 158.92, and 18932.51. The symbolic deck may be obtained from the supervisor of computer operations.

3. Store 36 numbers in the memory as follows:

$$a_i \quad (i = 1, 2, \dots, 12)$$

$$b_i \quad (i = 1, 2, \dots, 12)$$

$$c_i \quad (i = 1, 2, \dots, 12)$$

Then draw a flow chart using a variable connector to evaluate the following:

$$A_i C_i \quad (i = 1, 4, 7, 10)$$

$$A_i + C_i \quad (i = 2, 5, 8, 11)$$

$$A_i \div C_i \quad (i = 3, 6, 9, 12)$$

where

$$A_i = \frac{(0.212 a_i + 7.852 b_i)^2}{1.784}, \quad i = 1, 2, \dots, 12$$

$$C_i = 5.736 c_i, \quad i = 1, 2, \dots, 12$$

Finally, write a program using the flow chart and run the program on the computer.

SHARE SUBROUTINE

IDENTIFICATION

Square Root, CL SQRT 03

R. Johnson - 11-22-55

Lockheed Aircraft Corporation, California Division

PURPOSE

Takes the square root of the absolute value of X, a floating point number.

RESTRICTIONS

None

METHOD

Four Newtonian iterations.

Accurate to eight significant decimal figures.

Indication of negative X.

USAGE

<u>LOC</u>	<u>OP</u>	<u>ADDR.</u>	<u>TAG</u>
n	TSX	SQRT	4
n+1		Return for X negative	
n+2		Normal return	

Place X in the accumulator in normalized floating point. The square root of X will be in the accumulator after return in normalized floating point.

CODING INFORMATION

Location symbols used are P₁, P₂, and C. Constants start at C and are three octal words, 001000000000, 100000000000, 000000000004, respectively. Erasable storage, COMMON through COMMON + 1, must be assembled with subroutine.

2.055 milliseconds if X > 0.

.024 milliseconds if X = 0.

2.103 milliseconds if X < 0 error return.

Figure 30 - SHARE Subroutine for Evaluating the Square Root of the Absolute Value of X

CHAPTER VIII

NUMBERS IN MACHINE LANGUAGE

The IBM 704 is a binary computer; that is, it uses the binary number system in performing all arithmetic operations. As explained in Appendix C, a number, $b_n b_{n-1} \dots b_2 b_1 b_0$, is represented in binary form as $b_n \cdot 2^n + b_{n-1} \cdot 2^{n-1} + \dots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$, where b_i ($i = 1, 2, \dots, n$) is either a one or a zero. For example, the binary integer 101 is $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$ or 5 in the decimal system. Of course, the size of a number you can place in core storage (or a register) is limited by the capacity of the core storage unit (or register). Each core storage unit, commonly called a *word*, has a capacity of 36 binary digits (bits). The first bit is reserved for the sign of a number. A zero in this position indicates plus, and a one indicates minus. The other bits are numbered 1–35 and are represented as shown in Figure 31.

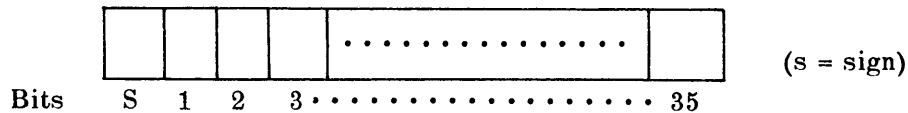


Figure 31 – Bits of a Core Unit

Thus the numerical capacity of one word is $2^{35}-1$ or 34,359,738,367, if you consider the number to be an integer.

In the arithmetic operations ADD, SUB, DVH, and MPY, that you have learned, the numbers used were integers; that is, the binary point was assumed to be to the right of the last digit. Actually, the binary point can be considered to be between any two bits of the word if sufficient care is taken to scale properly when using the arithmetic operations. These numbers and operations are called *fixed-point numbers* and *fixed-point operations*, respectively.

Fixed-point numbers are rarely used except to represent integers. Numbers that are not integers are usually represented in *floating-point form*. A number is in floating-point form when it is expressed as a signed fraction F times 2^n , where n is an integer. For example, the binary number $0.01 = 0.1 \times 2^{-1}$. If the most significant digit is immediately after the binary point, the number is in normalized floating-point form.

A *floating-point number* is represented in the core storage as follows: the first bit is the sign bit; bits 1–8 are reserved for the exponent of 2; and bits 9–35 are reserved for the fraction F (Figure 32). So that you have to work with positive exponents only, the exponent is increased by 128 before the computer stores it in positions 1–8. This number is then referred to as the *characteristic* (i.e., the exponent-of-two plus 128) of the number. Thus, the range of the characteristic is $0 \leq c \leq 255$, and the range of n is $-128 \leq n \leq 127$.

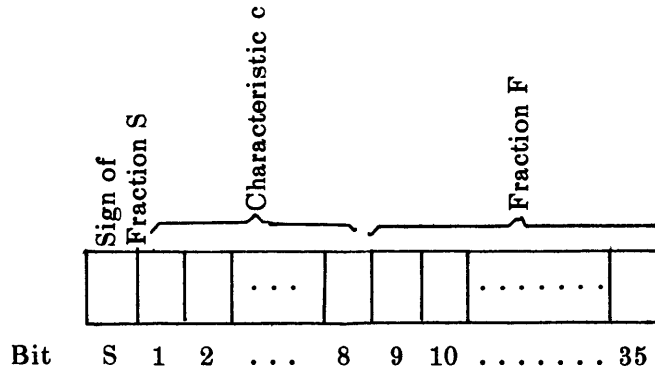


Figure 32 – Floating-Point Number Representation

The integer 5 is represented in normalized floating binary point form as 0.101×2^3 . Before placing this number in the core storage, the exponent is of two increased by 128 to get a characteristic of 131. (See Figure 33)

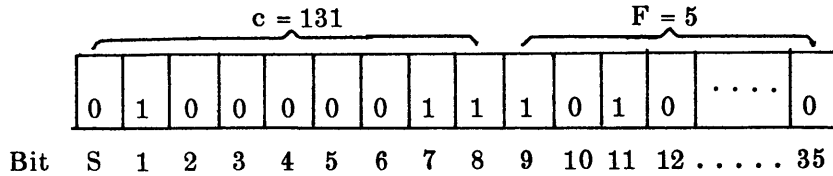


Figure 33 – Decimal 5 in Normalized Floating-Point Binary Form

If you store numerical data in floating-point form, you must use the floating-point arithmetic operations, which are FAD Y, FSB Y, FMP Y, and FDP Y (add, subtract, multiply, and divide, respectively). These instructions are used much like the corresponding fixed-point instructions, and will not be explained in detail here. (For details, see “Floating-Point Arithmetic Operations,” Reference 1.*)

However, an explanation of what happens when the computer does fixed-point operations using integers as data is given here. Much of this explanation can be readily extended to apply to floating-point operation as well.

First, note that the accumulator contains 38 bits: a sign bit, Q and P bits, and bits numbered 1–35, as indicated in Figure 34.

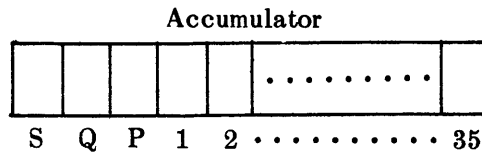


Figure 34 – Bits of the Accumulator

If the sum of two integers exceeds $2^{35}-1$ (the capacity of bits 1–35), the extra digits are shifted first into the P bit and then into the Q bit and otherwise are lost. This is called overflow.

* In the IBM 704 Reference Manual, there are two numbers associated with each instruction. These numbers will be explained in the next chapter.

Since the instruction STO SUM will place only the sign and bits 1–35 of the AC into core location SUM, then in case of overflow the location SUM will not contain the correct value of the SUM (because the Q and P bits are lost). Naturally, you do not desire incorrect answers. Therefore, an indicator and light are built into the computer to tell when there is overflow in the accumulator, and there are also instructions to test this indicator. These operate in the following manner. Whenever a 1 passes into or through bit P of the accumulator as the result of an instruction, the overflow indicator and light (for visual checking) are turned on. Then either of the instructions TOV Y or TNO Y tests the condition of the accumulator overflow indicator.

The instruction “Transfer on Overflow”

TOV Y

is used to check overflow. That is, if the overflow indicator and light are on when the computer comes to this instruction, the computer will turn off the indicator and light and take the next instruction from location Y. If the indicator and light are off, the computer proceeds to the next instruction in sequence. Thus, on overflow you might have some sort of error routine beginning at location Y. For instance, you might wish to print out the contents of the accumulator and the data that caused the overflow.

The instruction “Transfers on No Overflow”

TNO Y

also is used to check overflow. When the computer comes to this instruction and the overflow indicator and light are off, the computer takes its next instruction from location Y. If the indicator and light are on, the computer first turns off the indicator and light and then proceeds to the next instruction in sequence.

The use of these instructions is illustrated here with a simple problem. Find the sum of two numbers and store the result unless overflow occurs. In that case, do not store the result.

IBM Data Processing Division

SHARE 704 Symbolic Coding Form

Problem															
Coder John Smith										Date September 15, 1959		Page 1 Of 1			
H	Location					Op	Address, Tag, Decrement					Comments		Identification	
1	2	6	7	8	10	11	12					72	73	80	
						JOB									
						SAP									
						ORG	100								
						CLA	X				$c(x) \rightarrow c(AC).$				
						ADD	Y				$c(x) + c(Y) \rightarrow c(AC).$				
						TOV	ERROR				If overflow occurs, transfers to error routine. If not, continues to next instruction.				
						STO	RESULT				$c(x) + c(Y) \rightarrow c(RESULT).$				
						TRA	ENDJOB								
						ERROR	(Instructions for Error Routine)								
						END	100								
						LOD	4								
						TRA									

Figure 35 – Program to Check for Overflow in Addition

At ERROR, you might have the computer print out the contents of X, Y, and the accumulator so that you can find out why overflow occurs. If there is no overflow, the computer will store the sum in RESULT and continue from there. AC overflow is also possible in fixed-point subtraction (SUB) .

Since, in fixed-point division (DVH), division does not take place if $|C(Y)| \leq |C(AC)|$, a signal called the divide-check indicator and light is turned on when the above condition occurs. At the same time, the computer stops. When the programmer observes the divide-check light, he can tell the operator of the 704 to push the control button and continue the program. However, there is another instruction, called "Divide or Proceed"

DVP Y

Division takes place under the same condition as with the instruction DVH Y. However, if $|C(Y)| \leq |C(AC)|$, division does not take place, the divide-check indicator and light are turned on, and the computer proceeds to the next instruction.

The instruction that is used to test whether or not division has taken place is "Divide Check Test"

DCT

When the divide-check indicator and light are off and the computer comes to the DCT instruction, it will skip the next instruction and proceed from there. If the divide-check indicator and light are on, the computer turns them off, then proceeds to the next instruction in sequence. For example, suppose you want to divide X by Y, where X does not exceed the capacity of the MQ, and you want to check whether or not division takes place. You would program as in Figure 36.

The instruction DCT is also used to check whether or not floating-point division takes place.

In the floating-point arithmetic operations, AC overflow occurs when the characteristic c of the result is too large; i.e., $c > 255$. If the characteristic c is less than zero, it is called *underflow*. The instructions TOV and TNO are also used to test overflow or underflow of floating-point arithmetic operations. During floating multiply or divide, it is possible that the characteristic in the MQ is too small or too large. In this case, the MQ overflow indicator and light are turned on. To test MQ overflow or underflow, use the instruction "Transfer on MQ Overflow"

TQO Y

This instruction operates as follows. If the MQ overflow indicator and light have been turned on by an overflow or underflow in the MQ characteristic during a previous floating-point operation, the computer turns off the indicator and light and takes the next instruction from location Y. If the indicator and light are off, the computer proceeds to the next instruction in sequence.

For a full explanation of the floating-point operations, see pp. 21–26 of Reference 1.

Problem													
Coder John Smith						Date September 15, 1959			Page 1 Of 1				
H	Location					Op	Address, Tag, Decrement					Comments	Identifi- cation
1	2	3	4	5	6	7	8	9	10	11	12	13	14
						JOB							
						SAP							
						ORG							
						CLA	ZERO					$0 \rightarrow c(AC).$	
						STQ	X					$X \rightarrow c(MQ).$	
						DVP	Y					$Quotient \left(\frac{X}{Y}\right) \rightarrow c(MQ)$ and remainder $\rightarrow c(AC), \text{ if } c(Y) > c(AC) .$	
						DCT						Checks to see if division takes place. If division takes place, computer skips next instruction and proceeds from there. If division does not take place, computer goes to next instruction.	
						TRA	ERROR					Goes to "ERROR" routine.	
						STO	REM					Remainder $\rightarrow c(REM).$	
						STQ	QUOT					QUOTIENT $\rightarrow c(QUOT).$	
						TRA	ENDJOB						
						ERROR	Instructions for Error Routine						
						END	100						
						LOD	4						
						TRA							

Figure 36 – Program to Check Whether or Not Division Takes Place

SUMMARY – Chapter VIII

Since the IBM 704 is a binary machine, all numbers are represented as a combination of ones and zeros. Each core unit consists of 35 bits plus a sign bit.

Numbers in storage can be represented either as signed integers, fixed-point numbers, or floating-point numbers.

For integers and fixed-point numbers, the first or S bit is reserved for the sign, and bits labeled 1–35 are reserved for the number. (See Figure 31.)

For floating-point numbers, $F \times 2^n$, the S bit is reserved for the sign, bits 1–8 for the characteristic of the number, (which equals $n + 128$), and bits 9–35 for the Fraction F. (See Figure 32.) Instructions for arithmetic operations with floating-point numbers are:

- FAD - addition,
- FSB - subtraction,
- FMP - multiplication, and
- FDP - division.

The accumulator has two more bits than a core storage unit has. These are called the Q and P bits, and they are used for overflow in the accumulator. (See Figure 34.) The computer has an indicator and light to indicate this condition. The instructions used to test whether overflow has taken place in the accumulator are TNO and TOV.

To indicate whether or not division has taken place, the 704 has a divide-check light and indicator that are turned on if division fails to take place. To test whether this indicator is on, the instruction DCT is used.

TOV Y Transfer on Overflow

If the overflow indicator and light are on when the computer comes to this instruction, the computer will turn off the indicator and light and take the next instruction from location Y. If the overflow indicator and light are off, the computer proceeds to the next instruction in sequence.

TNO Y Transfer on No Overflow

If the overflow indicator and light are off when the computer comes to this instruction, the computer takes its next instruction from location Y. If the overflow indicator and light are on, the computer first turns off the indicator and light, and then proceeds to the next instruction in sequence.

DCT Divide Check Test

If the divide-check indicator and light are off, when the computer comes to this instruction, it will skip the next instruction and proceed from there. If the divide-check indicator and light are on, the computer will proceed to the next instruction in sequence after turning off the indicator and light.

TQO Y Transfer on MQ Overflow

If the MQ overflow indicator and light have been turned on by an overflow or underflow in the MQ characteristic during a previous floating-point operation, the computer turns off the indicator and light and takes the next instruction from location Y. If the indicator and light are off, the computer proceeds to the next instruction in sequence.

DVP Y Divide or Proceed

Division takes place exactly as for DVH if $|c(Y)| > c |c(AC)|$. However, if $|c(Y)| \leq |c(AC)|$, division does not take place, the divide-check indicator and light are turned on, and the computer proceeds to the next instruction in sequence.

EXERCISES – Chapter VIII

1. Add the integer 534 to the integer 34,359,738,367 using fixed-point arithmetic and check for overflow. If there is no overflow, store the sum and write it on tape 9. If there is overflow (which there should be), transfer to ENDJOB. This will cause the computer to stop and write on tape a "Post Mortem," which will be printed after your SAP listing. In this Post Mortem, you will see the contents of the accumulator, bits 1–35, and the contents of the Q and P bits printed side by side. Notice that the P bit contains a 1, which indicates an overflow.

2. Divide 50 by 0 using fixed-point arithmetic divide. If you do not use the instruction DCT, you will notice in the Post Mortem the words "divide check on," indicating that division did not take place.

3. Place the number 3.10×10^3 into the computer in floating-point form and multiply it by the number 2.14×10^4 . To place a number $n \times 10^e$ in the computer in floating-point form, use the pseudoinstruction

DEC nEe

(See the definition of DEC in the SAP 3-7 instructions).

4. Solve the following system of equations, using floating-point arithmetic:

$$9.2x - 16.1y = 11.5$$

$$3.5x - 2.0y = -2.5$$

CHAPTER IX

INSTRUCTIONS IN MACHINE LANGUAGE

In the last chapter you saw what numbers, both fixed and floating type, looked like in the core storage. Now you will see how the various instructions appear in the computer. Actually, you have been using at all times *letters* to represent an instruction; for instance, ADD, SUB, etc. However, the 704 is a binary machine and everything is represented internally as a combination of 1's and 0's.

All instructions are divided into two types, A and B.

Type A instructions have a decrement part, whereas Type B instructions do not.

In Type A instructions, bits S, 1, 2 are reserved for the operation code. Since each instruction is represented by a combination of 1's and 0's, you could have 2^3 or 8 A instructions; however, there are only five; namely, TIX, TNX, TXI, TXL, and TXH. Bits 3–17 are reserved for the decrement; bits 18–20 are reserved for the tag; and bits 21–35 are reserved for the address of Type A instruction. Type A instructions are represented as indicated in Figure 37.

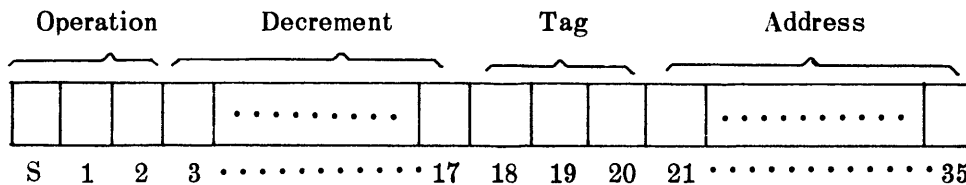


Figure 37 – Type A Instruction

Notice that the decrement and address fields contain 15 bits each. Therefore, the greatest number for the decrement or address is $2^{15}-1 = 32,767$, which makes it possible to refer to any one of the 32,768 core locations. Further, notice that each index register also has a capacity of 15 bits.

In Type B instructions, bits S, 1–11 are reserved for the operation code, bits 12–17 are not used, bits 18–20 are reserved for the tag, and bits 21–35 are reserved for the address, as indicated in Figure 38.

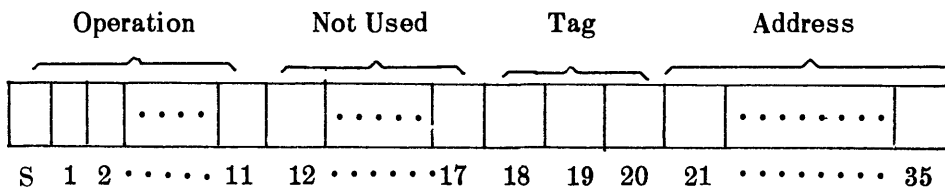


Figure 38 – Type B Instruction

With each instruction you associate an *octal code*, which corresponds to bits 1–11, and a sign, which represents bit S. These codes are found in the IBM 704 Reference Manual

(Reference 1)* and are printed beside each instruction in the SAP printout. For instance, the octal code for ADD is +0400, and since each octal number is equivalent to 3 binary numbers the octal code for ADD is represented in binary form as shown in Table 1 below.

TABLE 1
Octal and Binary Codes for ADD

Octal Code	+	0	4	0	0							
Binary Code	0	0	0	1	0	0	0	0	0	0	0	
Bits	S	1	2	3	4	5	6	7	8	9	10	11

The octal code for TIX is +2000. If you use the instruction TIX 101, 1, 1, it would appear in the SAP printout thus (Figure 39):

Sign	Operation	Decrement	Tag	Address	Operation	Address	Tag	Decrement
+	2	00001	1	00145	TIX	101,	1,	1
Octal Code**								

Figure 39 – SAP Printout of TIX 101, 1, 1

This instruction is represented in core storage, as shown in Figure 40.

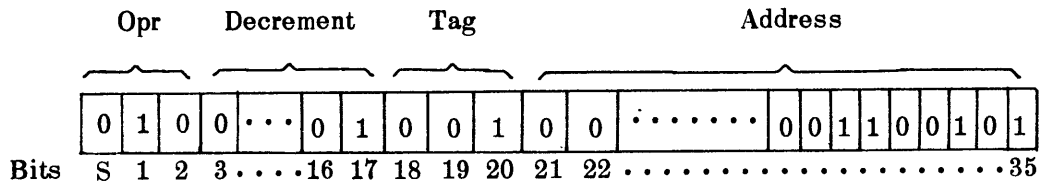


Figure 40 – Machine Representation of TIX 101, 1, 1

Thus far, you have seen that numbers and instructions are represented as a series of 1's and 0's in the computer. But suppose you have alphabetical characters which you wish to print. How do these appear in the computer?

*The number which appears in front of the instruction in the Reference Manual refers to the number of cycles required for the instruction to take place. Each cycle represents 12 microseconds.

**Of course, if the decrement is large enough, any of the first three octal digits of the decrement may be non-zero, even though the octal code for the instruction represents them as zeros.

First, note that punched cards can carry alphabetic as well as numeric characters. These characters are called Hollerith characters, named for Dr. Hollerith of the U. S. Bureau of the Census, who originated the idea of representing information by punched cards. Hollerith characters are represented in the computer by six binary digits or two octal digits. (See Table 2.) For example, the octal code for H is 30 and its binary form is 011000. Thus, you see that one core unit can hold six Hollerith characters. Of course, you can mix the letters with numbers, if you wish. To place an alphabetic character or a blank in core unit, use the pseudoinstruction.

BCD VADDRESS

This instruction will convert the Hollerith data ADDRESS into binary form and store it in V consecutive core locations, as indicated below.

Next, define a Hollerith word to consist of six Hollerith characters; i.e., six letters, five letters and a blank, or three letters followed by a blank and two more letters, etc.

ADDRESS is the Hollerith data to be converted and may occupy columns 13-72 on a card. If V is a blank, ten Hollerith words are converted and entered into ten consecutive core units. If V equals n, a number from 1 to 9, n Hollerith words will be converted and stored in n consecutive locations. V is punched in column 12. Here is an example.

Suppose you wish to use the heading FISCAL YEAR 1959. You must put this in storage before writing it on tape. Since you have sixteen Hollerith characters, you must reserve three core units for the title (i.e., V = 3). Thus, your instruction is

BCD 3FISCAL YEAR 1959

The octal code for this Hollerith data is given in Figure 41. Note that Δ indicates a space.

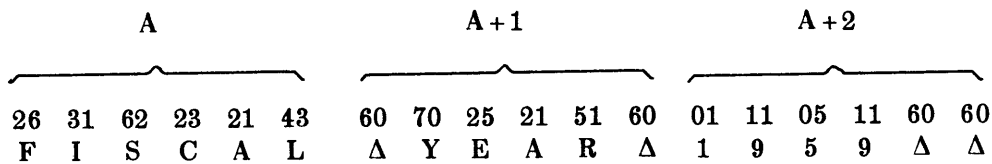


Figure 41 – Octal Code for FISCAL YEAR 1959

The data will go into three core units: A, A+1, A+2, as indicated below in octal representation (Figure 42).

Loc	Contents
A	26 31 62 23 21 43
	60 70 25 21 51 60
	01 11 05 11 60 60

Figure 42 – Octal Representation of FISCAL YEAR 1959 in Core Storage

Suppose now you desire to change the year to 1960. This can be done by using the logical operations to extract the date and replace it with a new date. Notice the date consists

of the first 12 bits of the third core unit, A + 2. The logical operations will be explained in the next chapter.

To write the Hollerith data on tape to be printed on paper, use the Bell System BCD print subroutine, X PRINT, making sure the first digit to be printed is a space.* Its calling sequence is:

```
TSX XPRINT, 4
MZE A, , Z
```

This will print the Hollerith data contained in consecutive locations from A to Z. Thus to print the data, FISCAL YEAR 1959, you would use the calling sequence:

```
TSX XPRINT, 4
MZE A, , A + 2
```

For further details of the F statements you have been using in the OUTPUT routine, see the Bell System instructions under the heading (Reference 2) "System Subroutines."

TABLE 2
704 BCD CODE

Character	Octal Code	Character	Octal Code
A	21	Δ (space)	60
B	22	=	13
C	23	"	14
D	24	+	20
E	25	-	40
F	26)	34
G	27	(74
H	30	\$	53
I	31	*	54
J	41	-	40
K	42	◆	33
L	43	/	61
M	44	,	73
N	45	+0	32
O	46	-0	52
P	47	0	00
Q	50	1	01
R	51	2	02
S	62	3	03
T	63	4	04
U	64	5	05
V	65	6	06
W	66	7	07
X	67	8	10
Y	70	9	11
Z	71		

*See "Carriage Control," Reference 5.

SUMMARY – Chapter IX

All IBM 704 instructions are divided into two types, A and B. Type A instructions have an address, tag, and decrement field, as indicated in Figure 43.

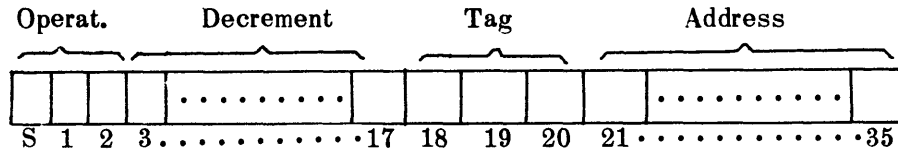


Figure 43 – Type A Instruction

Type B instructions do not contain a decrement field. (See Figure 44.)

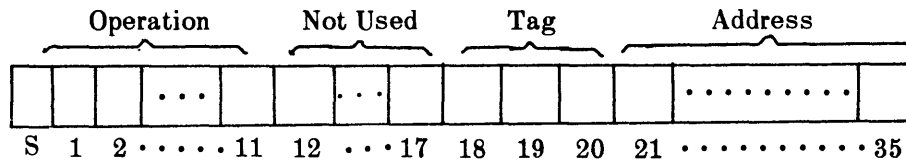


Figure 44 – Type B Instruction

With each instruction is associated a number called the *octal code* and a sign of the instruction. The sign represents the S bit, and the four octal digits represent bits 1–11 of the instruction.

Each Hollerith character (those produced by a manual key punch) is represented by two octal or six binary digits. (See Table 2.)

BCD VADDRESS

This instruction will convert the Hollerith data ADDRESS into binary form and store it into V consecutive core locations. V is either 1–9, or a blank which means 10 locations.

F BCD 1(n H)
SVN -1, 7, -1

This is the format for Hollerith characters when using the print subroutine, OUTPUT, where n is the number of Hollerith characters.

EXERCISES – Chapter IX

1. Place your name in the computer and then write it on tape 9 so that it will be printed after the SAP listing.

2. Hastings⁸ gives the following approximations for 10^x , $0 \leq x \leq 1$.

$$10^x = [1 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6]^2$$

$$a_1 = 1.1512,87586 \qquad a_4 = 0.0754,67547$$

$$a_2 = 0.6628,43149 \qquad a_5 = 0.0134,20940$$

$$a_3 = 0.2536,03317 \qquad a_6 = 0.0056,54902$$

Write a program to evaluate 10^x for some value of $0 \leq x \leq 1$ and check your result with a table.

3. Study your SAP listing from Exercise 2, observing the octal code of each instruction and the contents of each field.

CHAPTER X

LOGICAL OPERATIONS

You previously saw that the index registers can be used to modify an address portion of an instruction before executing that instruction. However, this did not change the address portion of the instruction as it appeared in core storage. To change the address portion of an instruction in a core unit, you could first clear and add the instruction into the accumulator, then add an integer equal to the amount by which you wish to modify the address, and, finally, store the modified instruction in its original location. This is illustrated in the following example.

Suppose you have been using the instruction

A MPY RESULT

but wish to change this to read

A MPY RESULT + 10.

You could do this as shown in Figure 45.

Operation and Address	Comment
CLA A	$c(A) \rightarrow c(AC)$
ADD TEN	$[c(A) + 10] \rightarrow c(AC)$
STO A	$[c(A) + 10] \rightarrow c(A)$
.	.
.	.
.	.
TEN DEC 10	Places decimal 10 in core storage

Figure 45 – Program to Modify the Address of an Instruction

The above program will increase the address of any instruction, provided the sign of that instruction is positive. If the sign of the instruction is negative, you must either add a negative number or use SUB in place of ADD.

The instruction “Store Address ”

STA A

could have been used instead of STO A. STA A places bits 21–35 of the accumulator into bits 21–35 of location A.

If, in place of the above set of instructions, you used the following set of logical operations, you would not need to be concerned about the sign of the instruction.

First, use the instruction "Clear and Add Logical" word:

CAL A

This will replace bits P and 1-35 of the accumulator, $c(AC)_{P,1-35}$, with the contents of core location A. (See Figure 46.)

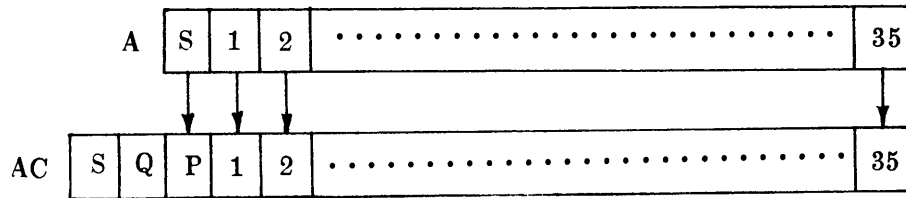
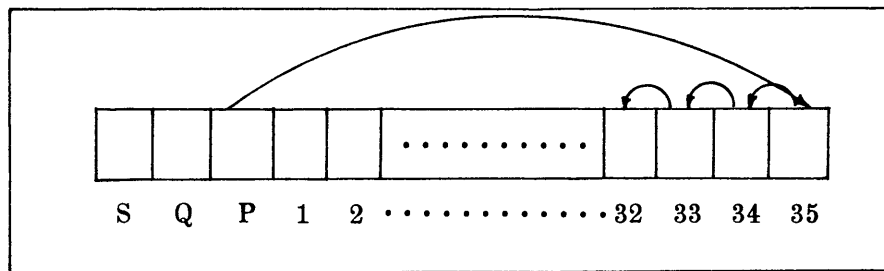


Figure 46 – Exchange of Bits as Result of CAL A

Then the instruction "Add and Carry Logical" word:

ACL TEN

will add the $c(Y)_{S,1-35}$ to the $c(AC)_{P,1-35}$, respectively, and replace $c(AC)_{P,1-35}$ with this sum. Position S of location Y is treated as a numerical bit, and the sign of the accumulator is ignored. A *carry* from position P of the AC adds into position 35 of the AC. It does not add into position Q. Bits Q and S of the accumulator and the $c(Y)$ are left unchanged. Since a carry from position P adds into position 35, no overflow is possible. (See Figure 47.)



ACL Y

Figure 47 – Operation of ACL Y

Finally, the instruction "Store Logical Word"

SLW A

replaces the $c(Y)_{S,1-35}$ with the $c(AC)_{P,1-35}$, leaving $c(AC)$ unchanged.

Suppose now you have a large amount of numerical data which consist of binary integers of not more than 11 binary digits and a sign. To put as many numbers as possible into core storage you could "pack" three numbers N_1 , N_2 , and N_3 into one core unit, DATA, as indicated in Figure 48, a *packed word*. Bits S, 12, 24 contain the sign of N_1 , N_2 , and N_3 , respectively.

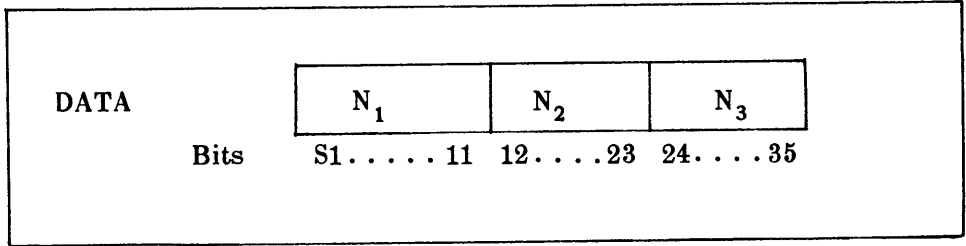


Figure 48 – A Packed Word

Next, assume you want to operate on each number. To do this, you would have to “unpack” (separate) each number from the others. One way to do this is by using the logical operations as explained below.

First, bring the contents of location DATA into the accumulator by means of the logical operation,

CAL DATA

This instruction will replace the $c(AC)_{P,1-35}$ with the contents of location DATA. Thus the sign bit of DATA does not go into the sign bit of the AC (as in CLA) but into bit P.

Next, you want to operate on one of the numbers, say N_1 . To eliminate N_2 and N_3 from the accumulator, you can use an *extractor pattern* and the instruction ANA Y. The extractor pattern will consist of a word stored in EXT (Figure 49) with 1's in the bits you wish to keep and 0's elsewhere:

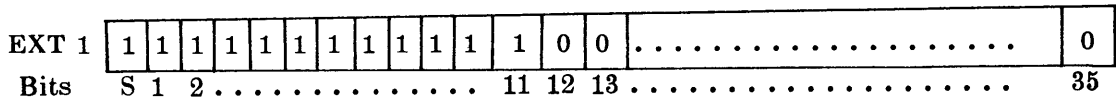


Figure 49 – Extractor Pattern 1

Then the instruction “AND to Accumulator”

ANA EXT 1

will leave unchanged $c(AC)_{P,1-11}$ and replace $c(AC)_{12-35}$ with 0's. Thus you have N_1 in $c(AC)_{P,1-11}$ (Figure 50).

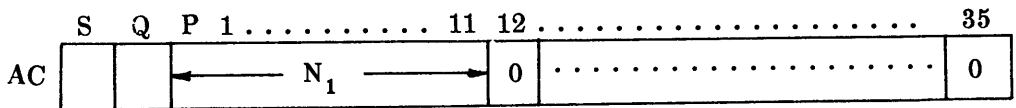


Figure 50 – N_1 in the $c(AC)_{P,1-11}$

The logical instruction "Store Logical Word"

SLW TEMP

will store $c(AC)_{P,1-35}$ into the $c(TEMP)_{S,1-35}$; i.e., it stores N_1 into location TEMP. Then if you do the instruction

CLA TEMP

you will have N_1 in bits S, 1-11 of the AC. The shift instruction "Long Right Shift"

LRS 59

will shift $c(AC)_{Q,P,1-35}$ and $c(MQ)_{1-35}$ 59 bits to the right, fill vacated positions with 0's, and make the sign of the MQ the same as the sign of the AC. Thus the sign bit of the MQ now contains the sign of N_1 . (See Figure 51.)

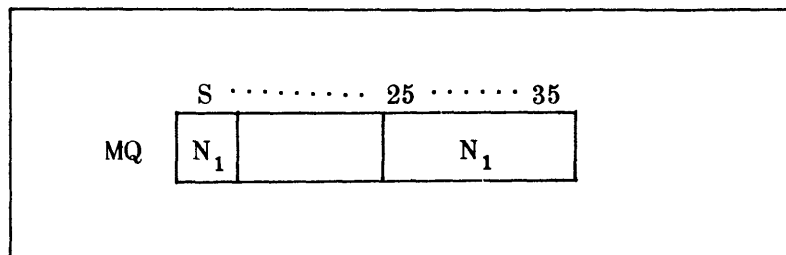


Figure 51 - N_1 in $c(MQ)_{S,25-35}$

Next, multiply N_1 by $c(MULT)$ and assume that the product P_1 will not exceed the capacity of 11 bits:

MPY MULT

Since the product is now in the (MQ), shift left 24 to put the product into $(MQ)_{S,1-11}$ with the instruction "Long Left Shift"

LLS 24

This instruction shifts the $c(AC)_{Q,P,1-35}$ and the $c(MQ)_{1-35}$ 24 places. (See Figure 52.) Bits from position 1 of the MQ enter position 35 of the AC, and the sign of the AC is made the same as the sign of the MQ. To have P_1 available for future use, store it in location PROD with the instruction

STQ PROD

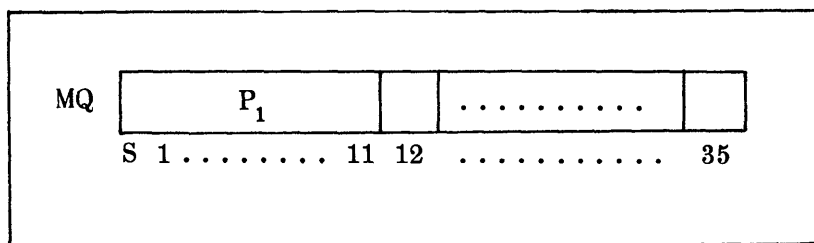


Figure 52 - P_1 in $c(MQ)_{S,1-11}$

So far you have extracted N_1 from the packed word, multiplied N_1 by another number, and stored the product P_1 into core location PROD. Now you would like to insert this new number P_1 into the packed word DATA in place of N_1 .

First, extract N_1 from core storage, DATA with EXT 2 (Figure 53) which has 0's in position S, 1-11 and 1's elsewhere and with the instruction ANS DATA.

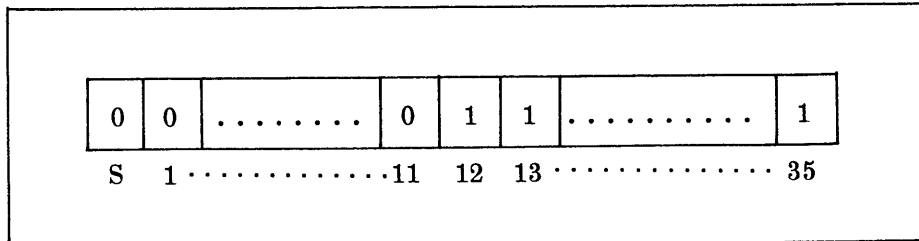


Figure 53 – Extractor Pattern 2

The instruction

CAL EXT 2

will replace bits P, 1-35 of the AC with bits S, 1-35 of core location EXT 2. Then the instruction “AND to Storage”

ANS DATA

will erase N_1 from core location DATA, leaving N_2 and N_3 as shown in Figure 54.

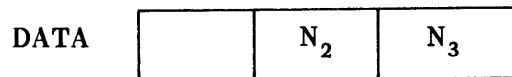


Figure 54 – Location DATA Now Contains N_2 and N_3

Next the instruction

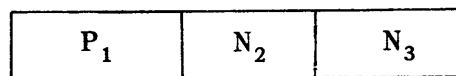
CAL PROD

will place the $c(\text{PROD})_{S,1-35}$ into the $c(\text{AC})_{P,1-35}$.

Then the instruction “OR to Storage”

ORS DATA

will insert P_1 into bits S, 1-11 of location DATA, to get



New Contents of DATA

Figure 55 – Location DATA Now Contains P_1 , N_2 , and N_3

Thus the packed word now contains P_1, N_2, N_3 , as shown in Figure 55.

To place the extractors in core storage, use the pseudo-operation

LOC OCT N

where LOC is the octal storage location of the extractor and N is the octal code of the extractor. For example, here is given the octal code of EXT 1 and EXT 2 (Figures 56a and 56b).

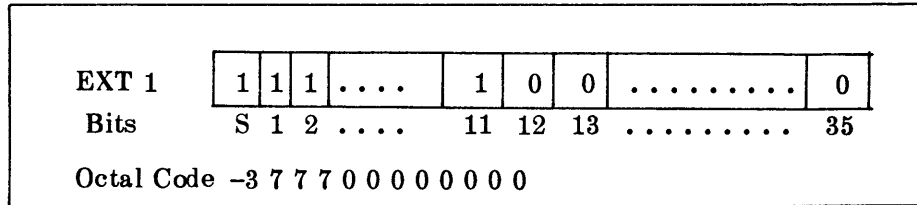


Figure 56a – Extractor Pattern 1 and its Octal Code

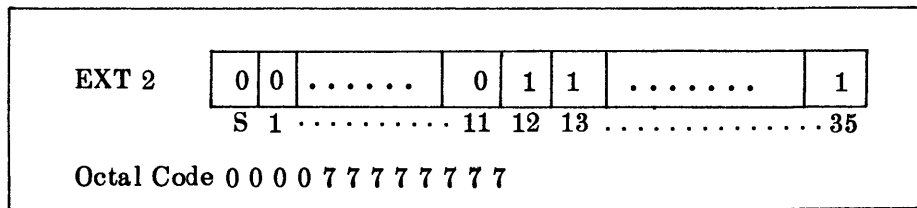


Figure 56b – Extractor Pattern 2 and its Octal Code

The flow chart and program for the above problem are given in Figures 57 and 58.

Before closing this chapter, the logical instruction ANA will be explained in detail; then you can read and interpret the other logical instructions as given in the summary and in the IBM 704 Reference Manual.¹

The instruction ANA Y is called “AND to Accumulator.” It matches each bit of $c(AC)_{P,1-35}$ with the corresponding bit of $c(Y)_{S,1-35}$, the $c(AC)_P$ being matched with the $c(Y)_S$. If the corresponding bit of both the AC and location Y is a 1, a 1 replaces the contents of that position in AC. If the corresponding bit of either the AC or location Y is a 0, a 0 replaces the contents of that position in the AC. The $c(AC)_{S,Q}$ are cleared and the $c(Y)$ are left unchanged.

In the example given, N_1 was to be left in the accumulator so the extractor pattern was 12 1's in positions S, 1–11 of location EXY 1 and 0's elsewhere. Suppose N_1 is the binary number -00000110101. Since the minus sign is indicated by a 1, N_1 appears in storage as

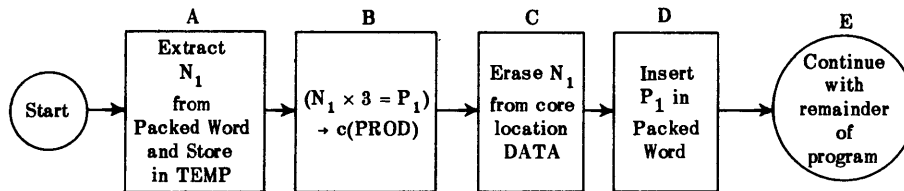


Figure 57 – Flow Chart for Extraction of N_1 from Packed Word and Insertion of P_1 into Packed Word

Problem							
Coder John Smith			Date September 15, 1959		Page 1	Of 1	
H	Location		Op	Address, Tag, Decrement		Comments	Identification
1	2	6	7	8	10	11	12
				JOB			
				SAP			
				ORG	100		
	A			CAL	DATA	Places packed word into positions P ₁ -35 of AC.	
				ANA	EXT 1	Leaves N ₁ in (AC) _P 1-11.	
				SLW	TEMP	Stores N ₁ in TEMP.	
	B			CLA	TEMP	Clears N ₁ in positions S ₁ -11 of AC.	
				LRS	59	Shifts contents of bits 1-11 into bits 25-35 of	
						MQ and places sign of N ₁ in c(MQ) _S .	
				MPY	MULT	Multiplies N ₁ by c(MULT) and places product	
						P ₁ in c(MQ) _S , 25-35.	
				LLS	24	Shifts product P ₁ into bits S ₁ -11 of MQ.	
				STQ	PROD	Stores product P ₁ with sign in location, PROD.	
	C			CAL	EXT 2	Places EXT 2 into positions P ₁ -35 of AC.	
				ANS	DATA	Erases N ₁ from core storage, DATA.	
	D			CAL	PROD	Places product P ₁ into positions P ₁ -11 of AC.	
				ORS	DATA	Inserts product P ₁ into position S ₁ -11 of AC.	
	E	(Beginning of instructions to be performed after P ₁ has been inserted in packed word.)					
	EXT1			OCT	777700000000		
	EXT2			OCT	000077777777		
	TEMP			BSS	1		
	PROD			BSS	1		
	MULT			DEC	3		
	DATA			OCT	0001110010101011	Storage of packed data, three 12-bit sets of	
						data per word.	
				END	100		
				LOD	4		
				TRA			

Figure 58 – Program Associated with Flow Chart of Figure 57

10000110101

Then when you match the extractor EXT 1 with this number, you get N_1 , left in the AC as indicated in Figure 59.

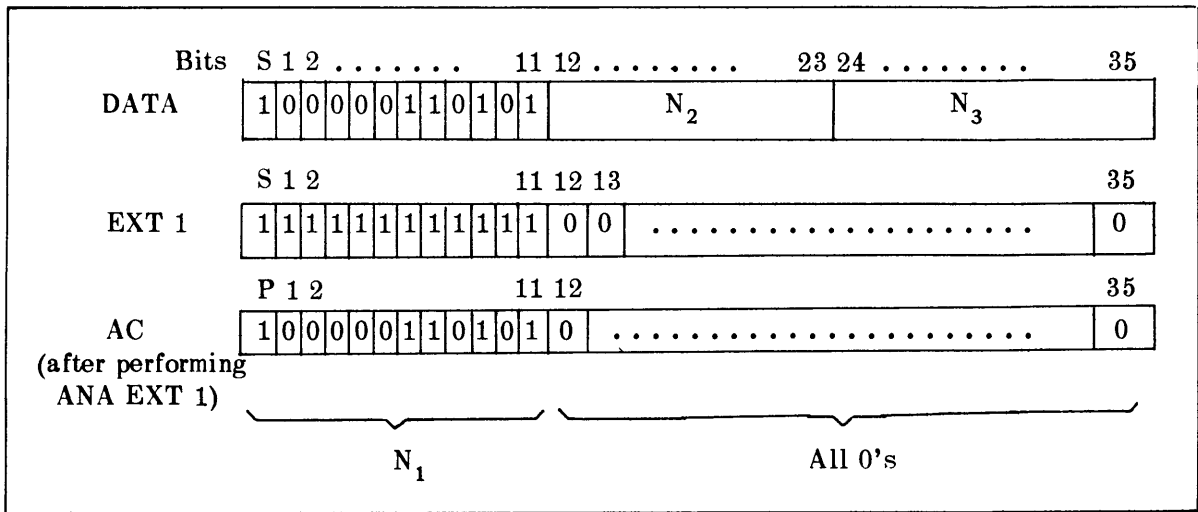


Figure 59 – Operation of ANA Instruction

SUMMARY – Chapter X

In this chapter you learned how to modify the address of the instruction by adding an integer to the instruction.

Then you studied how to pack several pieces of data into one computer word and how to unpack a number from a packed word.

Below are listed the new instructions used in this chapter.

STA Y Store Address

The $c(AC)_{21-35}$ replaces the $c(Y)_{21-35}$. The $c(AC)_{S,1-20}$ and the $c(Y)_{S,1-20}$ are left unchanged.

CAL Y Clear and Add Logical Word

Replaces the $c(AC)_{P,1-35}$ with the $c(Y)_{S,1-35}$. The sign of the $c(Y)$ replaces position P of the AC. The $c(Y)$ are left unchanged. (See Figure 46.)

ACL Y Add and Carry Logical Word

Adds $c(Y)_{S,1-35}$ to the $c(AC)_{P,1-35}$, and replaces $c(AC)_{P,1-35}$ with this sum. Position S of location Y is treated as a numerical bit, and the sign of the accumulator is ignored. A carry from position P of the AC adds into position 35 of the AC. It does not add into position Q. Bits Q and S of the accumulator and the $c(Y)$ are left unchanged. Since a carry from position P adds into position 35, no overflow is possible. (See Figure 47.)

ORS Y OR to Storage

Matches each bit of $c(AC)_{P,1-35}$ with the corresponding bit of the $c(Y)_{S,1-35}$, the P bit of the AC being matched with the S bit of the $c(Y)$. If the corresponding bit of either the AC or location Y is a 1, a 1 replaces the contents of that position in location Y. If the corresponding bit of both the AC and location Y is a 0, a 0 replaces the contents of that position in location Y. The $c(AC)$ are left unchanged.

LLS Y Long Left Shift

The $c(AC)_{Q,P,1-35}$ and the $c(MQ)_{1-35}$ are shifted left Y places (modulo 256). Bits from position 1 of the MQ enter position 35 of the AC. Positions made vacant are filled with zeros. If a nonzero bit is shifted into or through P, the AC overflow indicator and light are turned on. Bits shifted beyond position Q are lost. The sign of AC is made the same as the sign of the MQ.

LRS Y Long Right Shift

The $c(AC)_{Q,P,1-35}$ and the $c(MQ)_{1-35}$ are shifted right Y places (modulo 256). Bits from position 35 of the AC enter position 1 of the MQ.

Positions made vacant are filled with 0's, and bits shifted beyond position 35 of the MQ are lost. The sign of the MQ is made the same as the sign of the AC.

SLW Y Store Logical Word

Replaces the $c(Y)_{S,1-35}$ with the $c(AC)_{P,1-35}$. The $c(AC)$ are left unchanged.

ANA Y AND to Accumulator

Matches each bit of $c(AC)_{P,1-35}$ with the corresponding bit of the $c(Y)_{S,1-35}$, the P bit of the AC being matched with the S bit of the $c(Y)$. If the corresponding bit of both the AC and location Y is a 1, a 1 replaces the contents of that position in the AC. If the corresponding bit of either the AC or location Y is a 0, a 0 replaces the contents of that position in the AC. The $c(AC)_{S,Q}$ are cleared and the $c(Y)$ are left unchanged.

Note that in Reference 1 there are two additional instructions which shift only the contents of accumulator, and two logical operations. For details of these instructions see Reference 1, pp. 19-21. These instructions are:

ALS Y	Accumulator Left Shift
ARS Y	Accumulator Right Shift
ANS Y	AND to Storage
ORA Y	OR to Accumulator

EXERCISES – Chapter X

1. Do Exercise 3 of Chapter III making use of the operations of this chapter to modify addresses. Use the smallest number of core units possible.
2. Place in consecutive core storage units the following data:

- ABCDEF
DEFGHI
134DFE
CROD21
IE453L

Then use the logical operations to check to see which core units have a letter D in bits 18–23. If the word contains such a D, print the word on the output tape (tape 9). If it does not contain a D in bits 18–23, skip over it.

CHAPTER XI

PROGRAM CHECKING

After a problem has been defined, you must find the best numerical solution to the problem. Then you must flow-chart the solution, code the problem from the flow-chart, and finally run the program (the list of instructions and data) on the computer.

Of course, during each of these processes, you must check for errors and correct each error as you find it. Here a few methods to check and then to correct your program are discussed.

When you write a program in SAP language and have cards punched and loaded into the computer under Bell System control, you receive a SAP printout which lists all the pseudoinstructions used. If the assembly was successful, the computer translates the pseudoinstruction into machine language and then punches these binary instructions on IBM cards, with up to 22 instructions per card. (See Figure 60.)* The first instruction is punched in the first 36 columns of row 8; the second instruction is punched in columns 37-72 of row 8; the third instruction is punched in the first 36 columns of row 7; the fourth instruction is punched in columns 37-72 of row 7, etc. In row 9, columns 14-18, is the number of instructions contained on the card. In Figure 60, this binary word count is 10110; i.e., 22 instructions. Columns 22-36 of row 9 are reserved for the location of the first instruction. In Figure 60 this is 1100100 binary or 100 decimal.

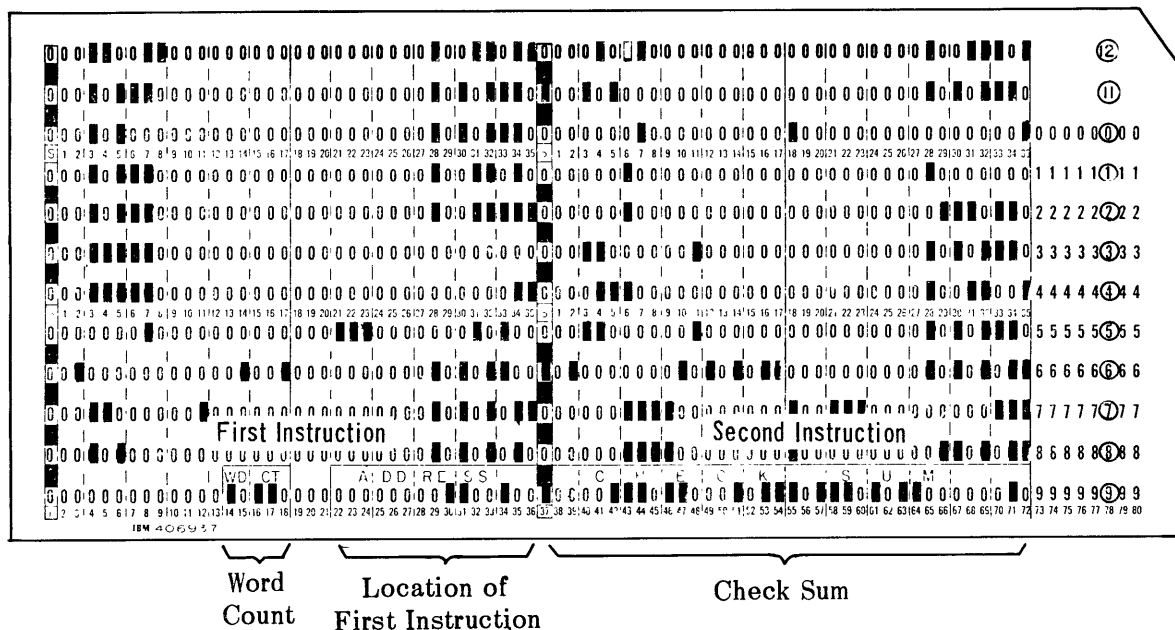


Figure 60 - A Binary Card with Twenty-two Instructions

*See description of "SAP" instruction, page 32.

Columns 37-72 of row 9 are reserved for the "check sum." The check sum is the logical sum (ACL) of all the words contained on the binary card, plus the contents of columns 1-36 of row 9. When SAP punches a binary card, it punches the check sum here. When the binary cards are loaded into the machine, SAP checks this sum after the instructions are in the computer. If this sum is incorrect, SAP stops and writes on tape "loading error." At the same time, SAP also prints on the on-line printer the words "loading error." Thus, you see that SAP assembly has built into it a check to see that the information from cards is placed in the core storage correctly. If this sum is incorrect, you must find the error and correct it. The card might have been punched incorrectly; the computer might have read the card incorrectly; etc.

To correct a word on a binary card, you can use a card containing any one of the following pseudoinstructions:

L OCT A
L DEC N
L BCD H

where L is the octal location of the incorrect word,

A is correct octal code of the word to be corrected,

N is correct decimal form of a number either fixed-point or floating as defined in SAP 3-7, and

H is the Hollerith word to be inserted in location L.

To run a program using the binary deck with or without correction, use the sequence of cards as indicated in Figure 61.

H	Location						Op	Address, Tag Decrement				
1	2	3	4	5	6	7	8	9	10	11	12	
							J	O	B			
							L	O	D			
							(Binary Deck)					
							(Correction Cards, if any)					
							T	R	A			
							(Data Cards)					

Figure 61 - Sequence of Cards to Run with Binary Deck

The Bell System data will load in the binary deck, make any corrections, then transfer to the first instruction to be executed.

Another method of checking a program is by the use of *snapshot dumps*. A *snapshot dump* is the printed contents of a specified number of core locations taken at particular times during the running of a program. The snapshots are first written on the Bell print tape and

then on paper. This affords you an opportunity to check to see if the contents of a specific core unit are what you anticipated at any given time.

The instruction used to obtain a snapshot dump is given in Figure 62.

H	Location						Op	Address, Tag Decrement			
1	2	3	4	5	6	7	8	9	10	11	12
					L		Y	-	-		S, A, B

Figure 62 – Instruction for Snapshot Dump

L specifies the location in octal form where you desired to take the snapshot dumps. The snapshot is taken before the instruction at L is executed. L must not contain a TSX instruction or a TXI instruction with a zero tag, nor may L be a location occupied by instructions of the Bell System.

The snapshot will be printed either in octal, floating-point decimal, BCD, or fixed decimal form, according to whether S is 8,F,BCD, or Pn, where n is a decimal integer specifying the location of the binary point.

A, B specifies that octal locations A to B inclusive shall be dumped.

Y-- specifies whether the snapshot shall be taken under specified conditions (Conditional Dump) or not (Unconditional Dump). Some of the forms of Y-- are listed here, followed by the condition which determines whether or not a snapshot will take place.

Instruction	Condition for Dump
YUN	None.
YMI	If accumulator is minus.
YZE	If accumulator is zero.
YPL	If accumulator is plus.

To obtain a snapshot dump after the complete running of the program, insert a card with the instruction YPM followed by a dump card without a location specified. Such a dump is commonly called a *Post-Mortem Dump*.

All dump cards are inserted just after the LOD card when using either a symbolic deck or a binary deck, the post-mortem dump card being placed after the snapshot dump cards.

Thus, when using a symbolic or a binary deck, place the dump cards as indicated in Figure 63.

IBM Data Processing Division												SHARE 704	
Problem													
Coder												Date	
For Symbolic Deck													
H	Location					Op	Address, Tag, Decrement						
1	2	3	4	5	6	7	8	9	10	11	12	13	14
					JOB								
					SAP								
					(Symbolic Deck)								
					LOD	4							
					(Snapshot Dump Card)								
					(Post-Mortem Dump Cards)								
					TRA								
					(Data Cards)								

IBM Data Processing Division												SHARE 704	
Problem													
Coder												Date	
For Binary Deck													
H	Location					Op	Address, Tag, Decrement						
1	2	3	4	5	6	7	8	9	10	11	12	13	14
					JOB								
					LOD								
					(Binary Deck)								
					(Correction Cards)								
					(Snapshot Dump Cards)								
					(Post-Mortem Dump Cards)								
					TRA								
					(Data Cards)								

Figure 63 – Sequence of Cards when Making Corrections and/or Dumps with a Symbolic or Binary Deck

Here is an example. Assume that it is desired to assemble a SAP program and then run it, obtaining an unconditional octal snapshot dump of core units 210–250 (octal) at location 165 (octal) and an unconditional octal post-mortem dump of core units 144–510 (octal).

The program in Figure 64 will accomplish this.

IBM Data Processing Division												SHARE 704 Symbolic Coding Form					
Problem																	
Coder												Date		Page	Of		
John Smith												September 15, 1959		1	1		
H	Location					Op	Address, Tag, Decrement							Comments		Identification	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
					JOB												
					SAP												
					(Symbolic Deck)												
					LOD	4											
				165	YUN	8, 210, 250	Snapshot Dump										
					YPM		Post-Mortem Dump										
					YUN	8, 144, 510											
					TRA												
					(Data Cards)												

Figure 64 – Program to Obtain a Dump at Location 165 and a Post-Mortem Dump

For information concerning other snapshots, see Reference 2, under the heading “Post-Mortem and Snapshot Dumps.”

Notice that in the above program you place the data after the TRA card, whereas previously you had been storing the data into the core units by means of the pseudoinstruction DEC. Storing data within the program by means of the DEC instruction carries with it the disadvantage that the program must be reassembled with SAP each time the data is changed. By including in the program the calling sequence for an input read routine and allowing the data cards to follow the TRA card, you can provide for many sets of data being processed with the same binary (or symbolic) program. For example, suppose you are doing matrix multiplication of two 10 × 10 matrices and assume that each element is an integer. You could read in the data with the XINPUT routine of the Bell System as shown in Figure 65.

Problem											
Coder John Smith						Date September 15, 1959			Page 1		Of 1
H	Location			Op	Address, Tag, Decrement				Comments		Identification
1	2	3	4	5	6	7	8	9	10	11	12
					JOB						
					SAP						
					ORG	100					
					TSX	XINPUT					
					NTR	F,, 0					
					MON	A,, A+199					
					.						
					.						
					.						
					.						
					.						
					.						
				F	BCD	1(N13)					
					SVN	-1, 7, -1					
					END	100					
					LOD	4					
					TRA						
					(Data Cards)						

Figure 65 – Program to Multiply Two 10 × 10 Matrices, A and B

The computer will first assemble the above program, then load the binary program in the memory from tape 4, and finally begin to execute the instructions. The first three instructions cause the computer to read in the data and place it in core units A through A+199. Then the computer continues with the remainder of the program.

Two other ways to check a program will now be discussed. Most problems involve either a few calculations and a great amount of data or a great number of calculations and a relatively small amount of data.

If a program involves a great number of calculations, it is good practice to print intermediate results during the checkout of the program and check these with hand calculations. Also, if a problem runs a very long time, it is advisable to provide for dumping on tape the necessary part of the memory (core storage) at approximately 10-minute intervals. This can be done by the instruction "Write Block on Binary Tape N"

WTB N,A,B

which will write the contents of the core storage from core Storage A through core Storage B in binary form on tape N. This enables you to restart the program with a loss of, at most, 10 minutes of computer time by reading tape N into the memory and using the values of the parameters at that time. It is wise to alternate between two tapes, if possible, in taking such dumps, thereby guarding against complete loss of restart information in the event of a tape failure. Thus if something goes wrong near the end of the problem, you will not have to go back to the beginning but can restore the memory to its previous state as given in the snapshot just before the error occurred.

On the other hand, if a problem consists of much data and a few calculations, a snapshot may not be so important for usually you read data as you need it. Thus, to rerun a problem with much data, you can reset the instructions to their original form and run the program with the unused data.

SUMMARY – Chapter XI

When you compile a program with SAP, the pseudoinstructions are translated to machine language (binary) and punched on cards. These cards can contain as many as 22 instructions. Each card contains a check sum which the computer recomputes and checks when reading the cards. This affords you a check to see that the correct instructions are being placed into the memory. If an error is found in a binary deck, it can be corrected by an OCT, DEC, or BCD correction card as explained below.

If you want to check the contents of specific core units at any given time, you can take a snapshot dump of the core units. This dump may be *conditional* or *unconditional* and may be either in octal, fixed-point, floating-point, or BCD form. Dumps are useful when running a long problem because this makes it possible to rerun a problem at any intermediate point in which a memory dump has been taken.

The following pseudoinstructions are used for correcting a binary deck.

L OCT A Octal Correction

A is the corrected octal code of the word to be stored in Octal Location L.

L DEC N Decimal Correction

N is the correct decimal form of a fixed-point or floating-point number to be stored in Octal Location L.

L BCD H BCD Correction

H is the new Hollerith word to be stored in Octal Location L.

If a dump is to be made at the end of a program, it is called a *Post-Mortem Dump*, and all such dump cards should be preceded by a card with YPM in the operation columns.

The following pseudoinstructions are used to obtain snapshot dumps at octal location L of core units from A through B. The dump is obtained before the instruction at L is executed. Two of the more important restrictions are: L may not be a location occupied by the instructions of the Bell System, nor may L contain a TXI instruction with a zero tag or a TSX instruction with any tag. S in the instruction below may be either 8 for octal, F for floating-point decimal, BCD for Hollerith, or Pn for a fixed-point decimal number where n is a decimal integer specifying the assumed position of the binary point in the computer word.

L YUN S,A,B Unconditional Dumps

Dumps core units A-B on output tape 9.

L YMI S,A,B Conditional Dump

Dumps core units A-B if accumulator is minus.

L YZE S,A,B Conditional Dump

Dumps core units A-B if accumulator is zero.

L YPL S,A,B Conditional Dump
 Dumps core units A-B if accumulator is plus.
 YPM This card must be inserted before the first of the Post-Mortem Dumps.

You also have learned how to write a block of information in the binary mode on tape N.
 This instruction is

WTB N,A,B Write Block on Binary Tape N
 Writes the contents of the core block A to B, inclusive, on tape N.
 This block is followed by a check sum.

To run a symbolic program with dumps or a binary deck with corrections and/or dumps,
 place the cards as indicated in Figure 66.

IBM Data Processing Division												
Problem		For Symbolic Deck										
Coder												
H	Location					Op			Address, Tag Decrem			
1	2	6	7	8	10	11	12					
					JOB							
					SAP							
					(Symbolic Deck)							
					LOD			4				
					Snapshot Dump Cards							
					Post-Mortem Dump Cards							
					TRA							
					(Data Cards)							
IBM Data Processing Division												
Problem		For Binary Deck										
Coder												
H	Location					Op			Address, Tag Decrement			
1	2	6	7	8	10	11	12					
					JOB							
					LOD							
					(Binary Deck)							
					Correction Cards							
					Snapshot Dump Cards							
					Post-Mortem Dump Cards							
					TRA							
					(Data Cards)							

Figure 66 – Programs to Run a Symbolic Deck with Dumps or a Binary Deck with Corrections and/or Dumps

EXERCISES – Chapter XI

1. Obtain a SHARE subroutine that will do matrix multiplication and find the product of two 5×5 matrices, A and B, and print the results on tape 9. Use the input routine of the Bell System, XINPUT, to read the data in. Plant a snapshot dump after the read routine to obtain the elements of A and B. Also obtain an unconditional octal post-mortem dump of all the core units used except for those of the subroutine.

Check the results and, if necessary, correct the binary deck and rerun the problem.

2. Using the binary deck of Exercise 1, run Exercise 1 with different data.

CHAPTER XII

READING AND WRITING ON TAPE

As previously noted, information can be stored on tape in binary form or binary-coded decimal (BCD) form. If the tape is to be printed, however, the information must be in BCD format.

Here you will see exactly how information is stored on tape and how to read or write on a tape in either the BCD or binary mode.

Information is stored on tape, six bits at a time, placed laterally across the tape, plus a redundancy check bit. The six bits are referred to as a frame (see Figure 67). Thus six frames constitute one computer word.

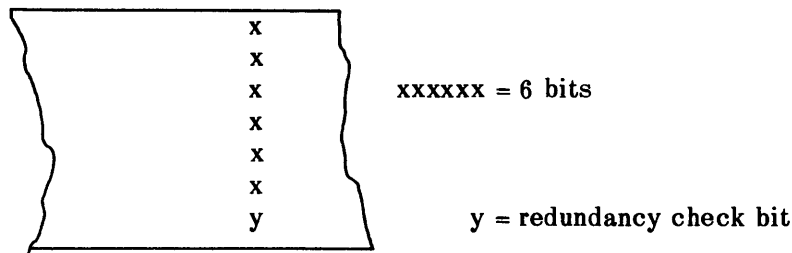


Figure 67 – One Frame on Tape

When the information is in binary form, the redundancy check is an odd check. Thus, if the group of six bits xxxxxx contains an even number of 1's, y is a 1, so that the number of 1's is odd. Otherwise, y is 0, again making the number of 1's odd. The y bit is also called a check bit and an odd check.

When the information is in the BCD form, the redundancy check is an even check; i.e., y is a 1 if the six bits contain an odd number of 1's and a 0 otherwise. This is illustrated in Figure 68.

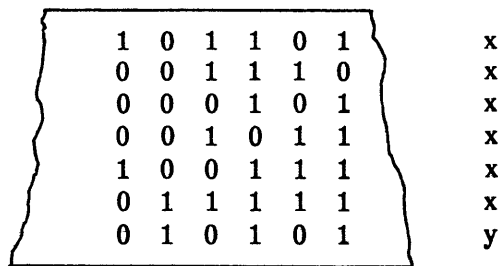


Figure 68 – BCD Mode of Tape

In keeping payroll information, a company may have a man's name, his number, rate of pay, etc., all written on one card. This is called the payroll record of the particular employee. To put this information on tape may require, say, ten computer words. Thus, you say that every ten words on this tape constitute a record. A record can be of any length. At the end

of each record, there is a longitudinal redundancy check, which is always an even check. Suppose you have a tape which consists of two word records. Figure 69 shows what a typical record on this tape might look like.

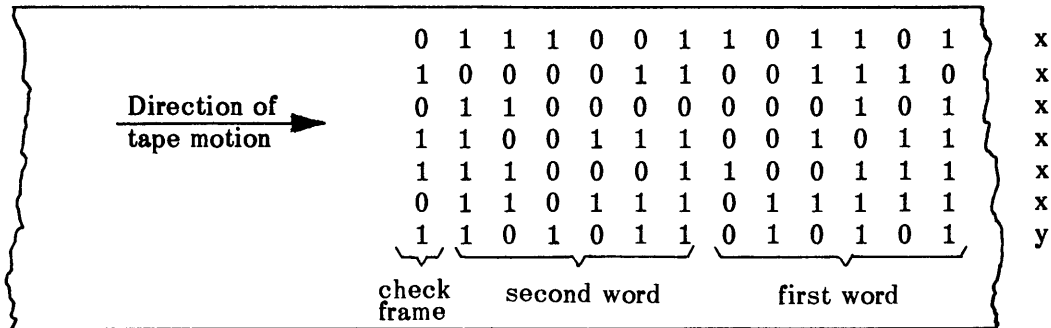


Figure 69 – Longitudinal Check

Of course, each employee's record is only one of several in the employee's file. Similarly on tape, a file may contain several records. To designate an end of a file on tape, there is an instruction "Write End of File"

WEF N

which will make a special mark at the end of file on tape N. A tape may contain more than one file. Each tape is 2400 feet long and holds 200 frames per inch, or 33 1/3 words per inch. The length of interrecord gap is 3/4 inch, and the length of an end-of-file gap is 4 1/2 inches which includes one end-of-record gap. Thus a tape can hold as many as 5,760,000 BCD characters or about 960,000 words.

To read or write on a tape, you must select the tape desired and specify in which mode (binary or BCD) the information is to be read or written. To prepare to read one record of information from tape N, which is in binary mode, use the instruction "Read Tape Binary"

RTB N

To prepare to read one record of information from tape N in the BCD mode, use the instruction "Read Tape Decimal"

RTD N

After the tape is selected, you must copy each word from the tape and store it in location Y of the core storage. You do this with the instruction "Copy and Skip"

CPY Y

This will copy one word from the tape specified by RTB and place this word first in the MQ, and then in core location Y. If an end-of-record gap is detected during a CPY order, this order is not executed. Instead, the computer skips the next two instructions following the CPY instruction and proceeds from there. If during a read order an end-of-file mark is detected, the computer skips the first CPY instruction and proceeds from there. These instructions are illustrated by an example.

Suppose tape 8 consists of 1000 10-word records in binary form followed by an end-of-file mark. Place all those records into the memory, beginning at core location X.

The flow chart for this problem is given in Figure 70.

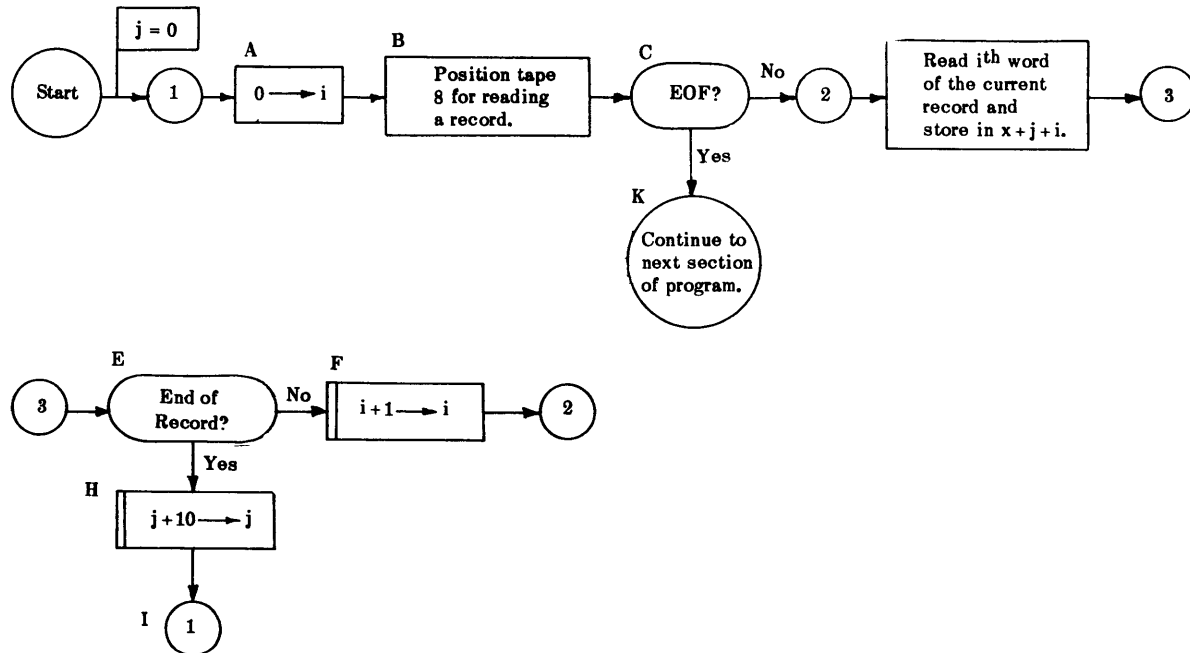


Figure 70 – Flow Chart to Copy 1000 10-Word Records from Tape to Memory

The program associated with the above flow chart is given in Figure 71.

Problem															
Coder John Smith										Date September 19, 1959		Page 1		Of 1	
H	Location					Op	Address, Tag, Decrement					Comments		Identification	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
						JOB									
						SAP									
						ORG					100				
						CAL					DSET		0 → j.		
						SLW					D				
	A					LXA			ZERO, 1				0 → i.		
	B					RTB			2				Positions tape 2 for reading.		
	D					CPY			X, 1				ith word → x + i + j.		
	EFG					TXI			*-1, 1, -1				If no EOR, i+1 → i, goes to D.		
	C					TRA			K				If no EOF, goes to K.		
	H					CAL			D						
						ACL			TEN				If EOR j+10 → j.		
						SLW			D						
	I					TRA			A				Goes to location A.		
	K								beginning of instructions to be performed after all records are read.						
	ZERO					DEC			0						
	TEN					DEC			10						
	DSET					CPY			X, 1						
						END			100						
						LOD			4						
						TRA									

Figure 71 – Program Associated with Chart of Figure 70

To prepare to write one record of information on tape N in binary mode, use the instruction “Write Tape Binary”

WTB N

To prepare to write one record of information on tape N in the BCD mode, use the instruction “Write Tape Decimal”

WTD N

Either of the above instructions will prepare the computer to write one record of binary information on tape number N. Then the instruction

CPY Y

will place one word from memory location Y into the MQ and then write this word on tape N. At the end of the copy loop, the computer writes the longitudinal check bits and end-of-record

gap and disconnects the tape. After writing the last record, you can make an end-of-file mark on tape N with the instruction

WEF N

For example, suppose you do the reverse of the previous problem; i.e., copy 1000 words from core storage in units of 10-word records onto tape 8. Assume that the first word is located at X. The flow chart in Figure 72 and the program (Figure 73) following it are the solutions to this problem.

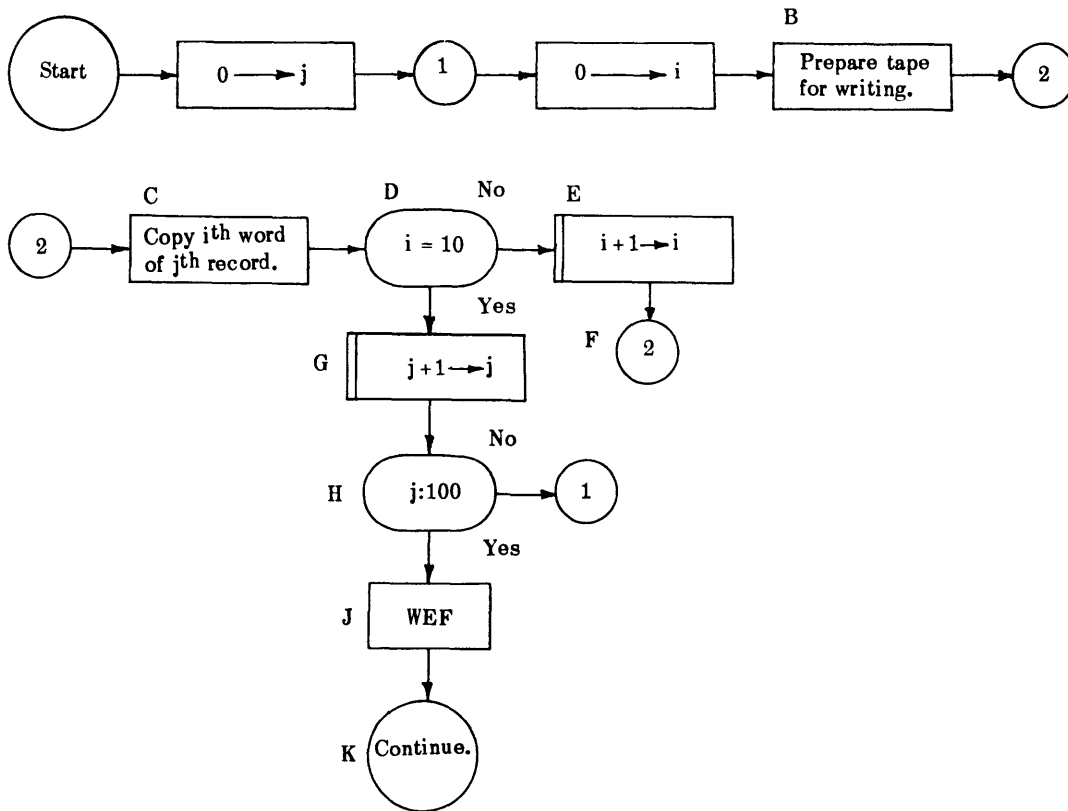


Figure 72 – Flow Chart to Copy 1000 Words from Core Storage in Units of 10-Word Records onto Tape

Problem											
Coder John Smith						Date September 19, 1959			Page 1 Of 1		
H	Location		Op	Address, Tag, Decrement				Comments		Identification	
1	2	6	7	8	10	11	12	72	73	80	
			JOB								
			SAP								
			ORG	100							
			CAL	C SET			i = 0.				
			SLW	C							
	A1		LXA	1HNRD, 1			100 → c(IR-1).				
	A2		LXA	TEN, 2			10 → c(IR-2).				
	B		WTB	8			Prepares tape 8 for writing.				
	C		CPY	X+10, 2			Copies ith word onto tape 8.				
	D, E, F		TIX	C, 2, 1			Checks for end of record.				
	G		CAL	C							
			ACL	TEN							
			SLW	C							
	HI		TIX	A2, 1, 1			Checks for last record.				
	J		WEF				Writes end of file.				
	K		()				Continues with program.				
	1HNRD		DEC	100							
	TEN		DEC	10							
	CSET		CPY	X+10, 2							
			END	A1							
			LOD	4							
			TRA								

Figure 73 – Program Associated with Flow Chart of Figure 72

After any copy order (when writing on tape) you are able to do other operations which do not involve the MQ before the next copy and EOR gap are made on tape, for the computer does not write the longitudinal check bits and end-of-record gap until 336 microseconds after the last copy order. Thus you can do any operation not involving the MQ after any copy instruction, provided the total time elapsed before the next copy does not exceed 336 microseconds. (For the timing of instructions, see Reference 1.) If another copy is given within this time, no EOR gap is written on the tape. However, if another copy is not given within this time or if a new write order of the same tape is given, the longitudinal check sum and EOR gap are written on the tape.

To rewind a tape N, use the instruction

REW W

All tapes used in a problem should be rewound before the problem is finished.

It is also a good idea to rewind tapes before using them, to be sure they are positioned at the beginning.

You can move tape N backwards one record with the backspace tape instruction

BST N

However, if tape N is at the load point, the instruction BST is interpreted as no operation.

If you are writing a large number of records on a tape or reading a nearly full tape, you do not want to pull the tape from its reel, so when the physical end of a tape is reached or when the tape breaks, an indicator and light, called the "tape indicator and light," are turned on. To test for this condition, use the instruction "End-of-Tape Test "

ETT

Also, when reading a tape, the computer recomputes the lateral and longitudinal check sum. If there is a mistake, an indicator light called the "tape-check indicator and light" is turned on. This indicator can be tested with the instruction "Redundancy Tape Test"

RTT

For full details of these last two instructions, see the summary at the end of this chapter.

Information can be transferred from cards to tape, or vice versa. In transferring from cards to tape, the computer writes the longitudinal check sum automatically on tape when it has read the last card.

SUMMARY – Chapter XII

Information is stored laterally on a tape, six bits at a time plus another bit which is the lateral redundancy check. For BCD information this check is an even check, and for binary information it is an odd check. At the end of each record there is a longitudinal check which is an even check.

When reading a tape, these check sums are automatically recomputed, and if one of these new sums does not agree with the corresponding one on the tape, the tape check indicator and light are turned on.

When reading and writing tape, the tape number and mode of information must be specified because BCD words are stored on tape differently than in memory. This is explained in detail in the Reference 1 under the heading "Character Alteration in BCD Mode."

The following instructions were explained in this chapter, but they by no means constitute all the read and write instructions. (See Reference 1, pp. 26–28.)

RTB N	Read Tape Binary Prepares to read one record of information from tape N in the binary mode.
RTD N	Read Tape Decimal Prepares to read one record of information from tape N in the BCD mode.
WTB N	Write Tape Binary Prepares to write one record of information from tape N in the binary mode.
WTD N	Write Tape Decimal Prepares to write one record of information from tape N in the BCD mode.
REW N	Rewind Tape Rewinds tape N.
WEF N	Write End of File Writes end of file on tape N.
BST N	Backspace Tape N Moves tape N in a backward direction one record. If the tape is at the load point, no operation is performed.
CPY Y	Copy and Skip Transfers one word of information between location Y and the tape N as specified in the read or write order which precedes this instruction.

If an EOR gap is detected when reading from a tape, the computer does not execute this instruction but skips the next two instructions after the CPY order and proceeds from there. If an EOF gap is detected in reading from tape, the computer does not execute this instruction but skips the next instruction after the CPY and proceeds from there. If another read instruction is given after a CPY instruction, the computer skips the first CPY instruction after this read order plus the next instruction following the CPY order and proceeds from there.

RTT **Redundancy Tape Test**

If the tape check indicator and light are on, the computer turns them off and takes the next instruction in sequence. If the tape check indicator and light are off, the computer skips the next instruction and proceeds from there.

ETT **End of Tape Test**

This instruction must be given after a read or write order and before the tape is disconnected. If the tape indicator and light are off, the computer skips the next instruction and proceeds from there. If the tape indicator and light are on, the computer turns them off and takes the next instruction in sequence.

EXERCISE – Chapter XII

1. Place enough BCD characters to fill up 100 core storages on cards. Then read this into memory and write this data into 10-word records in the binary mode on tape 6. Then copy this data back into memory and, finally, write the data in the BCD mode on tape 9. If you have done this exercise correctly, the printout from the tape should be exactly the same as the input.

CHAPTER XIII

FORTRAN, AN AUTOMATIC CODING SYSTEM

As has been observed, the machine language of the 704 is binary. To actually program a problem in binary form would certainly be time-consuming and error prone. Therefore, we have been using pseudo-operations, which a prewritten program, SAP, translates into machine language, and which the computer then executes. However, since it is desirable to have a language which is more similar to descriptive English and algebraic symbolism than the Bell and SAP languages are, a computer program called *FORTRAN* (Formula Translator) has been written.

FORTRAN enables you to write a program for a problem in a relatively few statements similar in form to algebraic formulae. These statements are punched on cards and then read into the computer. The FORTRAN translator, which is stored on tapes 1 and 2 when used with the Bell System, then causes the 704 to change these statements into a program in machine language which the computer can execute. The 704 writes the new program on tape 5 and punches it on cards. At the same time, it writes a SAP-type listing on tape 9. If the assembly is successful, the translated program can be immediately loaded into memory and executed.

A short example will now be given, using the FORTRAN language.

Given that the norm of an $n \times n$ matrix $A = (a_{ij})$ is $\text{norm } A = \left(\sum_{i,j} a_{ij}^2 \right)^{1/2}$, write a FORTRAN program to evaluate the norm of a matrix A, where

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

The program to do this is reproduced on Figure 74 on a standard FORTRAN coding form. The statements are written on this sheet first and then punched on cards. Notice that columns 1–5 are reserved for the statement number and columns 7–72 for the statement. If a character is written in column 6, it means that the statement on this line is a continuation of the statement on the previous line.

The first statement identifies the program that follows. A *c* is punched in column 1 to indicate that this statement is not to be interpreted as an instruction. The computer will, however, always print out the first statement of a FORTRAN program, whether it is an instruction or not.

The next two statements (JOB and FOR) are the Bell System control cards which load the FORTRAN program into the memory. They are punched in columns 8, 9, and 10.

The DIMENSION statement says to reserve 3×3 or 9 core units for the matrix A.

The READ statement says to read in the elements of the matrix A, which is punched on cards and placed after the TRA card and has the format as given by statement 1.

NORM of Matrix A		John Smith	September 19, 1959	
			1	1
C ←	FORTRAN STATEMENT			
C	NORM of Matrix A, John Smith, September 19, 1959			
	ΔJOB			
	ΔFOR			
	DIMENSION A(3,3)			
	READ1, ((A(I,J)J=1,3), I=1,3)			
1	FORMAT (5E14.8)			
	SUMSQ = 0.0			
	DO 2 I=1,3			
	DO 2 J=1,3			
2	SUMSQ = SUMSQ + A(I,J)* A(I,J)			
	PRINT 3, SUMSQ			
3	FORMAT (8HΔSUMSQ = ΔE25.8)			
	XNORMA = SQRTF(SUMSQ)			
	PRINT4, XNORMA			
4	FORMAT (8HΔNORMA = ΔE25.8)			
	ΔEND			
	ΔLOD			
	(AMFNR2, a subroutine needed with all FORTRAN programs)			
	(BESQRT, a subroutine needed with this program)			
	(A card with 9 punched in column 2)			
	ΔLODΔ5			
	ΔTRA			

Figure 74 – A FORTRAN Program to Evaluate the NORM of a Matrix

FORMAT statement 1 says that each element of A is written in floating-point decimal form with a field width of 14 and 8 places after the decimal point, and that 5 elements appear on a card. (See Figure 75.) Format and dimension statements are not executed.

+.10000000E+01					+.20000000E+01					+.30000000E+01					+.40000000E+01					+.50000000E+01				
000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000
11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22
33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33	33
44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44
55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55
66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66
77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77
88	88	88	88	88	88	88	88	88	88	88	88	88	88	88	88	88	88	88	88	88	88	88	88	88
99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Figure 75 – Data Card with Format 5E14.8

The next statement sets the quantity SUMSQ equal to zero.

The first DO statement means to do all the statements through statement 2 for $I = 1, N$. The next DO statement is similarly interpreted except that the running variable is J and not I.

Statement 2 gives the formula for the quantity SUMSQ. It reads that the SUMSQ is equal to the preset value of SUMSQ plus the square of each element. The two DO statements form the sum of those products.

The next statement says to print the sum of the squares according to the FORMAT statement 3.

The statement $XNORM4 = SQRTF(SUMSQ)$ says that the NORM A is equal to the square root of SUMSQ. The next two statements print the value of the NORM A.

The END statement signifies the end of the translation.

The LOD statement loads the next subroutine AMFNR2, which is needed with all FORTRAN programs, and the subroutine, BESQRT, which is needed for this program. These subroutines come immediately after the LOD card.

The final two cards are Bell control cards. The LOD 5 loads the binary instruction from tape 5 into the memory and TRA causes the computer to begin executing the program.

It is noted that if any FORTRAN statements are not written as specified in the FORTRAN manuals, the computer will print a diagnosis of the mistakes and not punch a binary deck.

Also the above program is only a simple example and by no means illustrates all the statements available in FORTRAN. See the FORTRAN Primer,⁴ FORTRAN Reference Manual,⁵ and the FORTRAN II Reference Manual⁶ for a full discussion of this automatic coding system.

SUMMARY – Chapter XIII

FORTTRAN is an automatic coding system similar to descriptive English and algebraic symbolism. FORTTRAN statements are punched on IBM cards in columns 7–72 and the statement number, if desired, in columns 1–5. Column 6 is the continuation column.

To compile and run a FORTTRAN program on the IBM 704, using the Bell Operating System, use the following cards in the order given:

JOB } Bell System Control Cards
FOR }
(FORTTRAN Program)
END } Bell System Control Cards
LOD }
(AMFNR2, a subroutine needed with all FORTTRAN programs)
(Any other subroutines needed)
(A card with 9 punched in column 2)
LOD 5 } Bell System Control Cards
TRA }

ACKNOWLEDGMENTS

The author wishes to express his thanks to all those who read the manuscript and offered many valuable suggestions, most of which were adopted. In addition, the author is indebted to Mr. Ernest Hairston, Mr. Dennis Hardy, and Mrs. Sharon Good, who always so kindly answered the many questions that occurred during the writing of this manual. Finally, the author is especially grateful to Mr. Kenton Meals, without whose assistance and encouragement this manual could not have been written.

APPENDIX A – OPERATIONS BY ALPHABETIC CODE

Alpha Code	Octal Code	Operation	Page
ACL	+0361	Add and Carry Logical Word	61
ADD	+0400	Add	3
ALS	+0767	Accumulator Left Shift	69
ANA	-0320	AND to Accumulator	62
ANS	+0320	AND to Storage	64
ARS	+0771	Accumulator Right Shift	69
BST	+0764	Backspace Tape	86
CAL	-0500	Clear and Add Logical Word	61
CLA	+0500	Clear and Add	3
CPY	+0700	Copy and Skip	81
DCT	+0760...012†	Divide Check Test	49
DVH	+0220	Divide or Halt	13
DVP	+0221	Divide or Proceed	49
ETT	-0760...011†	End of Tape Test	86
FAD	+0300	Floating Add	51
FDP	+0241	Floating Divide or Proceed	51
FMP	+0260	Floating Multiply	51
FSB	+0302	Floating Subtract	51
LDQ	+0560	Load MQ	10
LLS	+0763	Long Left Shift	63
LRS	+0765	Long Right Shift	63
LXA	+0534	Load Index from Address*	18
MPY	+0200	Multiply	10
ORA	-0501	OR to Accumulator	69
ORS	-0602	OR to Storage	64
REW	+0772	Rewind	86
RTB	+0762, 221-232††	Read Tape Binary	81
RTD	+0762, 201-212††	Read Tape Decimal	81
RTT	-0760...012	Redundancy Tape Test	86
SLW	+0602	Store Logical Word	61
STA	+0621	Store Address	60
STO	+0601	Store	3
STQ	-0600	Store MQ	10
SUB	+0402	Subtract	8
TIX	+2000	Transfer on Index**	18
TNO	-0140	Transfer on No Overflow	48
TNX	-2000	Transfer on No Index**	20
TOV	+0140	Transfer on Overflow	48
TPL	+0120	Transfer on Plus	25
TQO	+0161	Transfer on MQ Overflow	49
TRA	+0020	Transfer	20
TSX	+0074	Transfer and Set Index*	36
TXH	+3000	Transfer on Index High**	54
TXI	+1000	Transfer with Index Incremented**	20
TXL	-3000	Transfer on Index Low or Equal**	54
WEF	+0770	Write End of File	81
WTB	+0766, 221-232††	Write Tape Binary	75
WTD	+0766, 201-212††	Write Tape Decimal	83

*Not indexable.

**Not indexable but contains a decrement part.

†These instructions require the indicated numbers in the last three octal positions.

††The second number is the address of the instruction and specifies the tape unit. See p. 27 of Ref. 1.

APPENDIX B – INSTRUCTIONS TO THE SAP AND BELL SYSTEMS

SAP SYSTEM		
Code	Pseudo-operation	Page
BCD	Binary-Coded Decimal or Hollerith Data	30, 56, 57, 58
BSS	Block Reservation	8, 27
DEC	Decimal Data	5, 34, 53
END	End of Program	33
OCT	Octal Data	72, 77
ORG	Origin	4, 8

BELL SYSTEM		
Code	Pseudo-operation	Page
FOR	Load FORTRAN	90
JOB	Beginning of Job	32, 34, 72
LIB	Library Routine	38, 43
LOD	Load Program	33, 34, 72, 92
SAP	Load SAP	32, 34
TRA	Transfer to first instruction of program to be executed.	33, 34, 72, 92, 93
YMI	If c(AC) is minus, dump core storage.	73, 77
YPL	If c(AC) is plus, dump core storage.	73, 78
YPM	Post-Mortem Dump Cards follow.	73, 78
YUN	Unconditional Dump	73, 77
YZE	If c(AC) is zero, dump core storage.	73, 77

APPENDIX C – DECIMAL, BINARY, AND OCTAL NUMBER SYSTEMS

DECIMAL SYSTEM

In the decimal (or base 10) number system, any number can be represented by using the ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, where each digit can occupy any position in the number. Each position is associated with a power of ten. In particular, the integer 53712 in decimal notation represents

$$5 \times 10^4 + 3 \times 10^3 + 7 \times 10^2 + 1 \times 10^1 + 2 \times 10^0$$

The positions from right to left in an integer are spoken of as the units, tens, hundreds, thousands, etc., digits.

BINARY SYSTEM

Using this same scheme, a binary (or base 2) number can be represented by using the digits 0 and 1 where each digit may occupy any position in the number. Each position is now associated with a power of 2. Some examples of binary integers and their decimal equivalents are given in Figure 76.

Binary Number	Binary Formula	Decimal Equivalent
101	$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$	5
1011	$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$	11
1110	$1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$	14

Figure 76 – A Few Binary Integers and Their Decimal Equivalents

To use any number system, you must know how to add and multiply any two digits. Thus at an early age, you learned the multiplication and addition tables of the decimal numbers from 0 to 9. These each consist of 100 entries. However, in the binary system each table has only 4 entries because there are only 2 digits, 0 and 1. To add and multiply in the binary system, you must use for each digit of the number the tables given in Figure 77.

+	0	1
0	0	1
1	1	10

Addition

	0	1
0	0	0
1	0	1

Multiplication

Figure 77 – Addition and Multiplication Tables for Binary Number System

To illustrate the use of these tables, multiply 6 by 7 using binary arithmetic. (See Figure 78.)

Decimal System	Binary System
$\begin{array}{r} 6 \\ 7 \\ \hline 42 \end{array}$	$\begin{array}{r} 110 \text{ (6)} \\ 111 \text{ (7)} \\ \hline 110 \\ 110 \\ 110 \\ \hline 101010 \end{array}$

Figure 78 – Multiplication of 6 by 7

OCTAL SYSTEM

In a manner similar to the decimal and binary number systems, a number in the octal, or base 8, number system can be represented by the digits 0, 1, 2, 3, 4, 5, 6, 7 in positions associated with powers of 8. Since $8 = 2^3$, every octal digit can be written as a binary number with three or less digits. Hence, if the digits of a binary number are grouped by threes, the octal equivalent can be immediately written down. Figure 79 illustrates the equivalence of a few binary, octal, and decimal integers.

Binary Number	Octal Number	Octal Formula	Decimal Equivalent
00100100	144	$1 \times 8^2 + 4 \times 8^1 + 4$	100
010011010001	2321	$2 \times 8^3 + 3 \times 8^2 + 2 \times 8^1 + 1$	721
100111110	476	$4 \times 8^2 + 7 \times 8 + 6$	318

Figure 79 – A Few Binary Integers with Their Octal and Decimal Equivalents

The multiplication and addition tables for the octal number system are given in Figure 80.

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Addition

·	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Multiplication

Figure 80 – Addition and Multiplication Tables for Octal Number System

Here is an example using the above tables to multiply 100×318 using octal arithmetic:

Decimal System	Octal System
318	476
100	144
31,800	2370
	2370
	476
	76070

An integer can be represented in a number system with any base, b , as follows:

$$b_n b_{n-1} \dots b_2 b_1 b_0$$

where

$$0 \leq b_i \leq b-1 \quad (i = 0, 1, \dots, n)$$

Its value is given by the formula

$$b_n \cdot b^n + b_{n-1} \cdot b^{n-1} + \dots + b_2 \cdot b^2 + b_1 \cdot b + b_0$$

In discussing the different number systems, only integers have been considered here. The next question is how do you interpret numbers after the decimal or binary point. If you consider a number to the base b

$$0.b_{-1} b_{-2} b_{-3} \dots b_{-n}$$

it is represented by the following formula

$$b_{-1} b^{-1} + b_{-2} b^{-2} + \dots + b_{-n} b^{-n}$$

For example, the binary number 0.111 is equivalent to

$$\begin{aligned} 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} &= 0.500 + 0.250 + 0.125 \\ &= 0.875 \end{aligned}$$

The above discussion gives the definition of numbers with different bases. Now will be discussed some practical methods to convert from one system to another and to convert a floating-point* octal number to a decimal number.

INTEGERS

Decimal to Octal or Octal to Decimal

Use the table of Appendix C of the IBM 704 Reference Manual.¹

Decimal to Binary

First convert the number to octal; then write the binary equivalent of each digit.

Binary to Decimal

Change the binary integer to octal, then to decimal, using Appendix C of the IBM 704 Manual.¹

FRACTIONS

Decimal to Octal

This method is illustrated with an example. Suppose you wish to convert the decimal fraction 0.957.

- 1) Write the number as a fraction:

$$0.957 = \frac{957}{1000}$$

- 2) Multiply and divide the fraction by 8:

$$\frac{957}{1000} \times \frac{8}{8} = \frac{7656}{8000}$$

*See Chapter VIII.

3) Divide the result into 2 parts, thusly:

$$\frac{7656}{8000} = \frac{7000}{8000} + \frac{656}{8000} = \frac{7}{8} + \frac{656}{8000}$$

The numerator of the first fraction is the first octal digit, namely 7.

4) Multiply the second fraction of step 3 by $\frac{8}{8}$:

$$\frac{7}{8} + \frac{656}{8 \times 10^3} \times \frac{8}{8} = \frac{7}{8} + \frac{5248}{8^2 \times 10^3}$$

5) Break the second fraction of 4 into parts to obtain the second octal digit:

$$\frac{7}{8} + \frac{5248}{64,000} = \frac{7}{8} + \frac{5}{8^2} + \frac{248}{8^2 \times 10^3}$$

6) Repeat the above process to obtain as many digits as desired. For the number of your example, its octal equivalent to 6 places is 0.751770.

You can also convert from a decimal fraction to an octal fraction using Appendix D of the IBM 704 Manual.¹

Decimal to Binary

Change the decimal fraction first to octal, then to binary. Or if you wish, use the same method as for changing a decimal fraction to octal except use the multiplicative factor $2/2$ instead of $8/8$.

Octal to Decimal

Appendix D of the IBM 704 Reference Manual¹ is an octal-decimal fraction conversion table. Notice that the biggest octal fraction is 0.377. However, this represents no difficulty, for if the number you are converting is bigger, you can split it up into parts. Thus to convert the octal fraction 0.751770, find the decimal equivalent of each part as indicated below:

<u>Octal</u>	<u>Decimal</u>
0.300000	0.375000
0.300000	0.375000
0.100000	0.125000
0.051000	0.080078
<u>0.000770</u>	<u>0.001922</u>
0.751770	0.957000

Binary to Decimal

Change the binary number to octal and then to decimal form.

Floating Point Octal to Decimal

As explained in Chapter VIII, a floating ($F \times 2^n$) point number is stored in the memory in three parts. The first bit represents the sign bit; bits numbered 1–8 represent the characteristic of the number; and bits 9–35 represent the fraction F . The characteristic is 128 (decimal) plus the exponent of two ($n + 128$) or 200 (octal) plus the exponent of two ($n + 200$).

Now to convert a floating-point number to a decimal number, first change the fraction to a decimal fraction. Then multiply this result by 2^n .

For example, suppose you have the floating-point binary number as it would appear in an octal printout

$$+210751770000$$

where 210 is the characteristic and 0.751770000 is the fraction.

The fraction 0.751770000, as shown above, is equivalent to 0.957000 (decimal). The characteristic 210 represents $n = 8$. Thus the above floating-point number is equivalent to:

$$0.957000 \times 2^8 = 244.992000$$

For your convenience, Table 3 gives a list of characteristics from 131 to 247 with their equivalent exponents n of two and the decimal equivalent of 2^n .

TABLE 3

The Octal Characteristics from 131 to 247 with the Corresponding Exponent (n) of Two, and 2^n .

Octal Characteristic	Exponent (n) of Two	2^n												
131	-39	0.000	000	000	001	818	989	403	545	856	475	830	078	125
132	-38	0.000	000	000	003	637	978	807	091	712	951	660	156	25
133	-37	0.000	000	000	007	275	957	614	183	425	903	320	312	5
134	-36	0.000	000	000	014	551	915	228	366	851	806	640	625	
135	-35	0.000	000	000	029	103	830	456	733	703	613	281	25	
136	-34	0.000	000	000	058	207	660	913	467	407	226	562	5	
137	-33	0.000	000	000	116	415	321	826	934	814	453	125		
140	-32	0.000	000	000	232	830	643	653	869	628	906	25		
141	-31	0.000	000	000	465	661	287	307	739	257	812	5		
142	-30	0.000	000	000	931	322	574	615	478	515	625			
143	-29	0.000	000	001	862	645	149	230	957	031	25			
144	-28	0.000	000	003	725	290	298	461	914	062	5			
145	-27	0.000	000	007	450	580	596	923	828	125				
146	-26	0.000	000	014	901	161	193	847	656	25				
147	-25	0.000	000	029	802	322	387	695	312	5				
150	-24	0.000	000	059	604	644	775	390	625					
151	-23	0.000	000	119	209	289	550	781	25					
152	-22	0.000	000	238	418	579	101	562	5					
153	-21	0.000	000	476	837	158	203	125						
154	-20	0.000	000	953	674	316	406	25						
155	-19	0.000	001	907	348	632	812	5						
156	-18	0.000	003	814	697	265	625							
157	-17	0.000	007	629	394	531	25							
160	-16	0.000	015	258	789	062	5							
161	-15	0.000	030	517	578	125								
162	-14	0.000	061	035	156	25								
163	-13	0.000	122	070	312	5								
164	-12	0.000	244	140	625									
165	-11	0.000	488	281	25									
166	-10	0.000	976	562	5									
167	- 9	0.001	953	125										
170	- 8	0.003	906	25										
171	- 7	0.007	812	5										
172	- 6	0.015	625											
173	- 5	0.031	25											
174	- 4	0.062	5											
175	- 3	0.125												
176	- 2	0.25												
177	- 1	0.5												

TABLE 3 (Continued)

Octal Characteristics	Exponent (n) of Two	2 ⁿ			
200	0				1
201	1				2
202	2				4
203	3				8
204	4				16
205	5				32
206	6				64
207	7				128
210	8				256
211	9				512
212	10		1	024	
213	11		2	048	
214	12		4	096	
215	13		8	192	
216	14		16	384	
217	15		32	768	
220	16		65	536	
221	17		131	072	
222	18		262	144	
223	19		524	288	
224	20	1	048	576	
225	21	2	097	152	
226	22	4	194	304	
227	23	8	388	608	
230	24	16	777	216	
231	25	33	554	432	
232	26	67	108	864	
233	27	134	217	728	
234	28	268	435	456	
235	29	536	870	912	
236	30	1 073	741	824	
237	31	2 147	483	648	
240	32	4 294	967	296	
241	33	8 589	934	592	
242	34	17 179	869	184	
243	35	34 359	738	368	
244	36	68 719	476	736	
245	37	137 438	953	472	
246	38	274 877	906	944	
247	39	549 755	813	888	

REFERENCES

1. "IBM Reference Manual, 704 Data Processing System," International Business Machines Corporation, New York (1958).
2. "704 Input-Output and Monitor System - BE SYS 2," Bell Telephone Laboratories, by George H. Mealy, Murray Hill, N. J. (May 1959).
3. "704 Symbolic Assembly Program UA SAP 3-7," Bell Telephone Laboratories, by George H. Mealy, Murray Hill, N. J. (Jan 1958). (Revision of UA SAP 1-2, United Aircraft Corporation, Roy Nutt, Hartford)
4. "Programmer's Primer for FORTRAN," International Business Machines Corporation, New York (1958).
5. "FORTRAN Reference Manual," International Business Machines Corporation, New York (1958).
6. "FORTRAN II Reference Manual," International Business Machines Corporation, New York (1958).
7. "704 Snapshots," Applied Mathematics Laboratory Bulletin, David Taylor Model Basin, Carderock, Maryland, Vol. 1, Nos. 1-18 (1958-1959).
8. Hastings, Cecil, Jr., "Approximations for Digital Computers," Princeton University Press, Princeton, N. J. (1955).

BIBLIOGRAPHY

1. Eckert, J., and Jones, Rebecca, "Faster, Faster," McGraw-Hill Book Company, Inc., New York (1955).
2. McGee, W.C., "Generalization: Key to Successful Electronic Data Processing," Journal of the Association for Computing Machinery, Vol. 6, No. 1 (Jan 1959).
3. "Programming for the UNIVAC System," Remington Rand UNIVAC, New York. Chapter 10 (1953).
4. McCracken, D.D., "Digital Computer Programming," John Wiley & Sons, Inc., New York (1957).
5. Alt, Franz L., "Electronic Digital Computers," Academic Press, Inc., New York (1958).
6. Gorn, Saul, "Standardized Programming Methods and Universal Coding," Journal of the Association for Computing Machinery, Vol. 4, No. 3 (Jul 1957).
7. "Faster than Thought," Edited by B. V. Bowden, Sir Isaac Pitman & Sons, Ltd., London (1955).

8. Vazsonyi, Andrew, "Scientific Programming in Business and Industry," John Wiley & Sons, Inc., Boston (1958).

9. Kemeny, J.A., et al., "Introduction to Finite Mathematics," Prentice-Hall, Inc., Englewood Cliffs, N. J. (1957).

INDEX

- Accumulator, 3, 47
- Addition:
 - fixed-point, 3, 46, 60
 - floating-point, 47, 51
- Address:
 - definition of, 6
 - modification of, 15, 60
 - symbolic, 26
- Alphabetic code for operations, 54, Apps. A, B
- Alphabetical characters, 55–57
 - octal code for, 57
- Arithmetic operations:
 - fixed-point, 3, 47
 - floating-point, 47, 51
- Assembly, 26
- Automatic programming, 26–30, 90–93

- Bell Telephone System, 32–35
- Binary coded decimal, 56, 57
 - table of, 57
- Binary corrections, 72
- Binary number system, 46, 47, App. C

- Calling sequence, 36–38
- Card:
 - binary, 71
 - IBM, 5, 6, 71, 91
 - reading, 3, 74
- Carry, 61
- Characteristic, 46, 49
- Check sum, 72
- Compiler, 26
- Computational flow symbol, 21
- Connector:
 - nonvariable, 22
 - variable, 38, 40, 41
- Control instructions, 16, 20, 21
- Copy loop, 81
- Core storage, 3, 46, 47

- Corrections, 72, 77
- Counting symbol, 22

- Decimal system, App. C
- Decrement, 6, 16, 54, 55, 58
- Divide check light and indicator, 49, 50, 51
- Division:
 - check indicator, 49, 50
 - fixed-point, 11
 - floating-point, 47
- Dumps:
 - conditional, 73, 74
 - Post-Mortem, 73, 74
 - Snapshot, 72, 73, 74
 - unconditional, 73, 74

- End of file, 81, 82, 84
- End of record, 81
- End of tape, 86
- Errors, 71
- Extractors, 62, 64, 65, 66, 67

- File, 81
- Fixed-point numbers, 46, 49
- Floating-point numbers, 46, 47, 49
 - normalized, 46, 47
- Flow charts, 21, 22, 24
- FORTTRAN, 90–93
- Fractional part of a floating-point number, 46, 47
- Frame, 80

- Hollerith characters, 56, 57

- IBM card, 3, 5, 6, 71, 91
- Index registers, 15–20
- Input, 3, 32, 74, 90
- Instruction location counter, 15

Instruction register, 15
 Instructions:
 indexable, 15, 20
 nonindexable, 15, 20
 Type A, 54, 55
 Type B, 54

 Lateral check of tape, 80
 Location counter, 15
 Logical choice symbol, 21
 Logical operations, 60–71
 Longitudinal check of tape, 81

 Magnetic tape, 4, 80–89
 MQ register, 10, 11
 Multiplication:
 fixed-point, 10
 floating-point, 46

 Normal return, 36
 Normalizing floating-point numbers, 46, 47
 Numbers:
 Fixed-point, 46, 49
 floating-point, 46, 47, 49
 Integers, 5, 46, App. C

 Octal code, 54, 55, 65
 BCD characters, table of, 57
 Octal number system, 55, App. C
 Output, 3, 4, 32, 57, 92
 Overflow, 47, 48, 49
 AC, 47, 48
 MQ, 49

 Packed word, 61, 62, 64
 Physical arrangement of data on tape, 80, 81
 Physical end of tape, 86
 Post Mortem, 73, 74
 Printer, 4

 Program, 5

 Record, 80, 81
 Redundancy check bit, 80, 81
 Registers:
 accumulator, 3, 47
 MQ, 10, 11
 storage, 3, 46, 47
 Rewinding tapes, 86

 SAP, 26, 32
 Snapshots, 72, 73, 74
 Storage, 3, 46, 47
 Storage register, 3, 46, 47
 Subroutines, 36–45, 92
 Subtraction:
 fixed-point, 8, 9
 floating-point, 47

 Symbolic programming, 26
 Symbols, definition of, 26

 Tag, 6, 15, 54, 55
 Tape, 4, 32, 36, 57
 Tape-check indicator and light, 86
 Tape indicator and light, 86

 Variable connector, 38, 40, 41
 Variable field, 32

 Write end of file, 81
 Writing tape, 4, 32, 37, 92
 Word, computer, 46

 XINPUT, 74
 XPRINT, 57

INITIAL DISTRIBUTION

Copies		Copies	
9	CHBUSHIPS	1	DIR, USNEES
	3 Tech Info Sec (Code 335)	1	DIR, USNRL
	1 Tech Asst to Chief (Code 106)		
	3 Electronic Computer Div (Code 280)	1	DIR, Natl BuStand
	1 Asst Chief for Field Activities (Code 700)		
	1 Asst Chief for Nuclear Prop (Code 1500)		
2	CHBUWEPS		
1	CHBUSANDA		
1	CHBUCEN		
1	CHONR		
1	CDR, NAVSHIPYD BSN		
1	CDR, NAVSHIPYD CHASN		
1	CDR, NAVSHIPYD LBEACH		
2	CDR, NAVSHIPYD NYK		
	1 Material Laboratory		
1	CDR, NAVSHIPYD MARE		
1	CDR, NAVSHIPYD NORVA		
1	CDR, NAVSHIPYD SFRAN		
1	CDR, NAVSHIPYD PHILA		
1	CDR, NAVSHIPYD PTSMH		
1	CDR, NAVSHIPYD PUG		
1	CDR, NAVSHIPYD PEARL		
1	CO & DIR, USNBTL		
1	CO & DIR, USNEL		
1	CO & DIR, USNRDL		
1	CO & DIR, USNAVTRADEVCCEN		
1	CO & DIR, USNMDL		
1	CO & DIR, USNUSL		
1	CDR, USNWL		
3	CDR, USNOTS, China Lake		
	1 Michelson Lab (Code 5038)		
	1 Attn: Library		
1	CDR, USNOL, White Oak		

David Taylor Model Basin. Report 1368.

TRAINING MANUAL ON PROGRAMMING FOR THE IBM 704,
by Carl L. Tibery. Apr 1960. vi, 110p. illus., tables, refs.
UNCLASSIFIED

This training manual consolidates the essential information from the IBM 704 Reference Manual, the Bell Telephone Laboratories IBM 704 Input-Output System - BE SYS 2, and the United Aircraft SAP 3-7 Programmer's Notes into one presentation, from which the mathematician can learn to write a program for the IBM 704 using those systems.

Among the topics covered are flow charting, machine language, symbolic programming, subroutines, input-output operations, and FORTRAN, an IBM automatic coding system. Each chapter includes simple examples and exercises.

1. Digital computers - IBM 704 - Instruction manuals
2. Digital computers - IBM 704 - Programming
3. Digital computers - IBM 704 - Operation
- I. Tibery, Carl L.

David Taylor Model Basin. Report 1368.

TRAINING MANUAL ON PROGRAMMING FOR THE IBM 704,
by Carl L. Tibery. Apr 1960. vi, 110p. illus., tables, refs.
UNCLASSIFIED

This training manual consolidates the essential information from the IBM 704 Reference Manual, the Bell Telephone Laboratories IBM 704 Input-Output System - BE SYS 2, and the United Aircraft SAP 3-7 Programmer's Notes into one presentation, from which the mathematician can learn to write a program for the IBM 704 using those systems.

Among the topics covered are flow charting, machine language, symbolic programming, subroutines, input-output operations, and FORTRAN, an IBM automatic coding system. Each chapter includes simple examples and exercises.

David Taylor Model Basin. Report 1368.

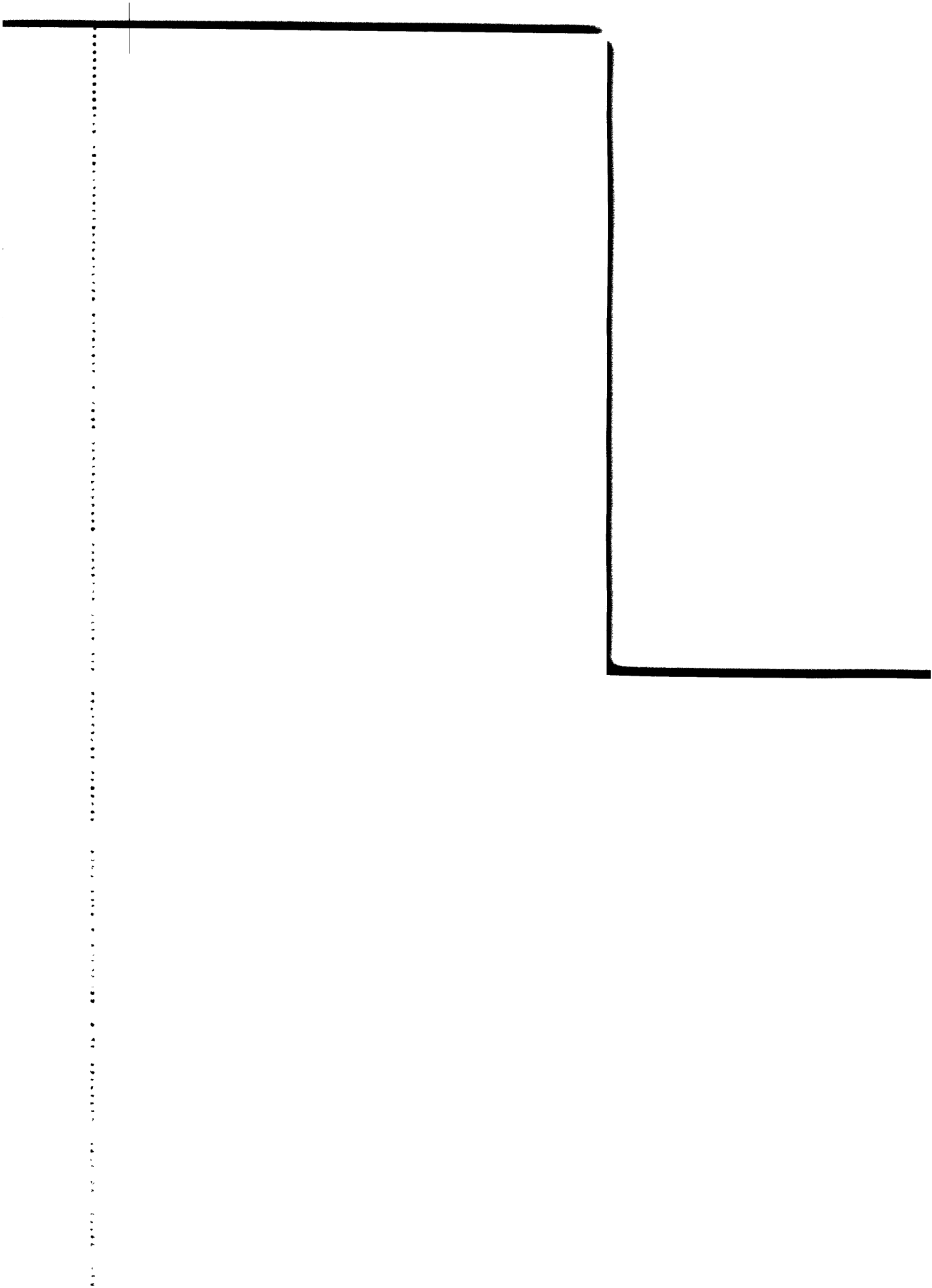
TRAINING MANUAL ON PROGRAMMING FOR THE IBM 704,
by Carl L. Tibery. Apr 1960. vi, 110p. illus., tables, refs.
UNCLASSIFIED

This training manual consolidates the essential information from the IBM 704 Reference Manual, the Bell Telephone Laboratories IBM 704 Input-Output System - BE SYS 2, and the United Aircraft SAP 3-7 Programmer's Notes into one presentation, from which the mathematician can learn to write a program for the IBM 704 using those systems.

Among the topics covered are flow charting, machine language, symbolic programming, subroutines, input-output operations, and FORTRAN, an IBM automatic coding system. Each chapter includes simple examples and exercises.

1. Digital computers - IBM 704 - Instruction manuals
2. Digital computers - IBM 704 - Programming
3. Digital computers - IBM 704 - Operation
- I. Tibery, Carl L.

1. Digital computers - IBM 704 - Instruction manuals
2. Digital computers - IBM 704 - Programming
3. Digital computers - IBM 704 - Operation
- I. Tibery, Carl L.



CONTROL CARDS

The control cards in the pocket are to be used with the exercises in this manual. The cards should be in the following order:

INT 2 (Notice that the label is in columns 73-80.)
FIN 1
FIN 2
FIN 3
FIN 4
FIN 5
FIN 6
FIN 7
FIN 8
FIN 9
FIN 10
FIN 11

Then:

CHAPTER II	EXERCISE 1	FIN 5
CHAPTER II	EXERCISE 2	FIN 5
CHAPTER III	EXERCISE 2	FIN 5
CHAPTER III	EXERCISE 3	FIN 5
CHAPTER IV	EXERCISE 1	FIN 5
CHAPTER IV	EXERCISE 2	FIN 5
CHAPTER V	EXERCISE 1	FIN 5
CHAPTER V	EXERCISE 2-1	FIN 5
CHAPTER V	EXERCISE 2-2	FIN 5
CHAPTER VI	EXERCISE 1	FIN 5
CHAPTER VI	EXERCISE 2	FIN 5

MIT LIBRARIES

DUPL



3 9080 02754 3401

