Programming Training Manual

CONTROL DATA® 1604 COMPUTER

Record of Revisions

REVISION	NOTES
A	Minor corrections only; does not obsolete previous
(11-1-61)	editions.

Pub. No. 60001500
February 1965

Address comments concerning this manual to:

Control Data Corporation
Technical Publications Department
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet located in the back of this manual.

PREFACE

This manual applies to the 1604-C in all respects. It is applicable to the 1604-A and 1604-B except for some details in Input/Output.

TABLE OF CONTENTS

CHAPTER I

PRELIMINARY INFORMATION

INTRODUCTION

The 1604 is a general-purpose, parallel, digital computer designed for large scale scientific problems or for large-volume data processing. It features the following general characteristics:

1. Equipment is low-power, solid-state throughout.

2. Physical size is such that it can be used in a semi-permanent office environment.

3. Several of the peripheral equipments can communicate directly with computer storage through independent access-channels. This permits reading and writing on magnetic tapes without external buffering devices.

4. Internal storage contains two magnetic core units operating together to form a single storage system of 1.6 million bits capacity; storage cycle time is 6.4 microseconds.

5. Real time inputs and outputs may be processed with peak rates up to 15 million bits per second.

6. Fixed and floating binary point arithmetic operations are possible with an average instruction time of 10 microseconds and a basic addition time of 1.2 microseconds.

7. The full use of index registers along with the regular instruction repertoire makes it possible to use almost 500 types of instructions in programming.

Basic Peripheral Equipment

a. Magnetic Tape provides the principal input-output medium. Recording on tape is in IBM coded or binary format. One magnetic tape cabinet is provided with the basic unit. Additional cabinets are available optionally.

b. Paper Tape equipment consists of a Control Data 350 Reader and a Teletype BRPE Punch.

c. Typewriter monitoring information, with keyboard entry, is provided by a modified IBM typewriter.

Optional Peripheral Equipment

a. A Model 1605 Adaptor converts character information from IBM equipments to 48 bit words. This adaptor communicates directly with 1604 storage through a separate input-output channel. This makes possible the use of the following additional peripheral equipments:

IBM 714 Card Reader
IBM 722 Card Punch
IBM 717 Line Printer
IBM 727 Magnetic Tape Transports

Summary of Characteristics

General Type -	Parallel
Word Length -	48 Bits
Storage -	32,768 words, two phase core
Storage Cycle Time -	6.4 microseconds
Instruction Type -	Single address (two per word)
Index Registers -	Six
Average Instruction Time -	10 microseconds
Arithmetic Operations -	Fixed & floating binary point
Basic Add Time -	1.2 microseconds
Circuits -	Transistor-diode, direct-coupled
Clock -	Two phase, 5 mc rate
Buffered Communication Channels -	Six 48-bit channels
Buffer Processing Time -	16 microseconds per word
Block Transfer Channels -	Two 48-bit channels
Block Transfer Time -	3.2 microseconds per word
Power -	4KW (central computer)
Minimum Floor Space -	400 sq. ft.

Internal Storage

The storage section contains two independent magnetic core storage units each with a capacity of 16,384 words of data 48 bits in length. These units operate together during execution of a program and can be considered as one 32,768 word storage system. FIG. 1 indicates the cores.

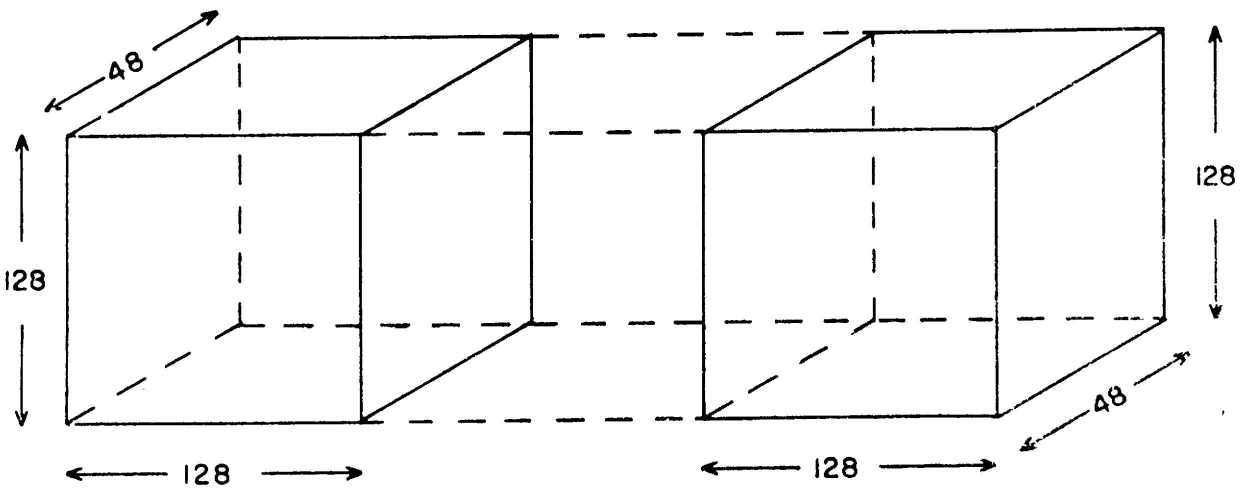


FIG. 1

Two Magnetic Core Units
Each 16,384 Words

All odd storage addresses reference one of the above units; all even storage addresses reference the other unit. Each unit has a storage cycle time of 6.4 microseconds. Actually the cycles of the two units overlap one another in the execution of a program. For example, if a person thinks of a storage reference consisting of a "read" and "write" time; while one reference is engaged in "writing" the next instruction can be using up the "read" time (see FIG.2). This results in an effective cycle time considerably less than the nominal 6.4 microseconds.

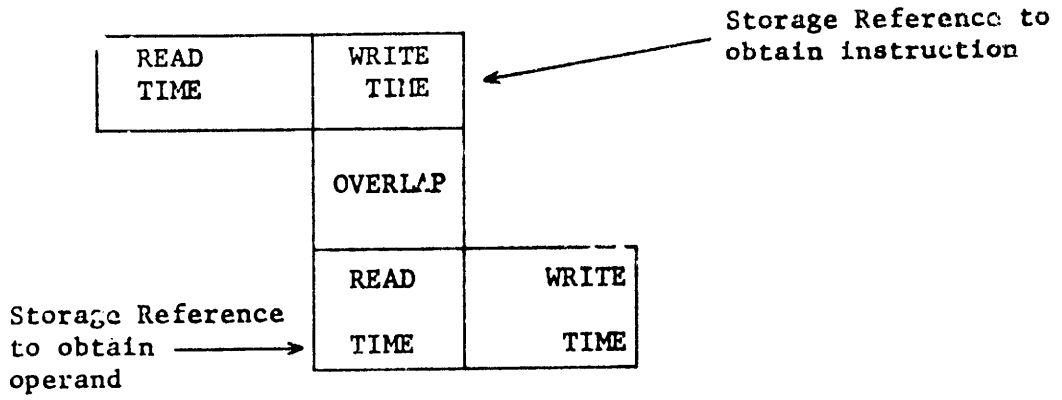

READ TIME
WRITE TIME
Storage Reference to obtain instruction
OVERLAP
READ TIME
WRITE TIME
Storage Reference to obtain operand

FIG. 2

Overlap of two Storage References

Address System

The model 1604 is a single-address computer. This means that each instruction can contain but one address. As previously mentioned, even addresses make one of the core units (16,384 locations) available; odd addresses make the other unit (16,384 locations) available. Thus a total of 32,768 locations are available for addresses. Since 15 bits are set aside for address designation, the total address range (in octal) is

0 0 0 0 0 to 7 7 7 7 7

In this range, addresses 00000, 00002, 00004 - - - 77776 (even) refer to one storage unit; whereas addresses 00001, 00003, 00005 - - - - 77777 (odd) refer to the other storage unit.

Even Addresses

$$\left.\begin{array}{l}00000\\00002\\00004\\ \\77776\end{array}\right\}\longrightarrow \begin{array}{c}\text{Storage Unit}\\1\end{array}$$

Odd Addresses

$$\left.\begin{array}{l}00001\\00003\\00005\\ \\77777\end{array}\right\}\longrightarrow \begin{array}{c}\text{Storage Unit}\\2\end{array}$$

Certain storage locations in the memory are used for control and reference functions. These storage locations may be addressed as operands as well as being addressed implicitly by certain control functions. The address assignments for these functions are listed below and will be explained in detail later.

Special Address	Function or Purpose
00000	Real Time Clock (1/60th of a second steps)
00001	Channel 1 control
00002	Channel 2 control
00003	Channel 3 control
00004	Channel 4 control
00005	Channel 5 control
00006	Channel 6 control
00007	Interrupt program (exit-entrance)

1604 Program Step

Each program step is a 48-bit word in storage which usually contains two single-address instructions. Each instruction is made up of 24 bits. If a program step contains a single instructio it is necessary that the other instruction be a "do nothing" instruction. This "do-nothing" instruction will be referred to in this manual as the "PASS INSTRUCTION". It has a special code and is described in detail in Chapter IV.

The two instructions in a program step can be considered as logically separate entities in a program sequence. As a practical matter in programming coding, however, the pair is an entity similar to a two-address instruction. The two single address **instructions are not separable in the sense that one may not be executed without the other.**

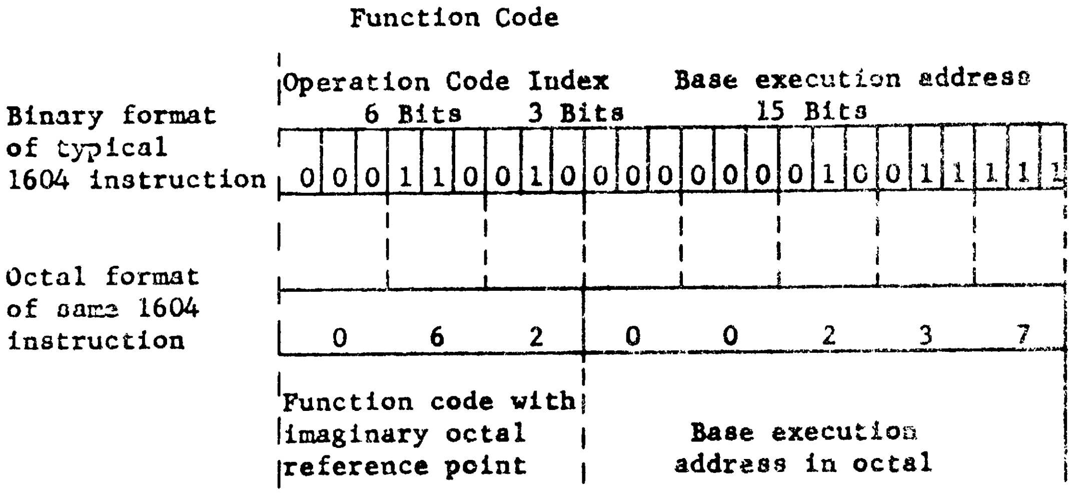
FIG. 3 indicates the possibilities of the general composition of a program step.

24 Bits Upper Instruction	24 Bits Lower Instruction
24 Bits Upper Instruction	24 Bits With the Pass Code
24 Bits With the Pass Code	24 Bits Lower Instruction

FIG. 3
Possible Word Formats

1604 Instruction

Each instruction is a 24-bit quantity specifying a certain operational entity in the execution of a stored program. The instruction format is shown below in FIG. 4.



Function Code
Operation Code 6 Bits
Index 3 Bits
Base execution address 15 Bits
Binary format of typical 1604 instruction
0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 1
Octal format of same 1604 instruction
0 6 2 0 0 2 3 7
Function code with imaginary octal reference point
Base execution address in octal

FIG. 4

Binary and Octal Formats of a Typical 1604 Instruction

Examining FIG. 4, it can be seen that a 1604 instruction contains 24 bits or 8 octal digits. Of this total, 9 bits or 3 octal digits make up the complete FUNCTION CODE. Actually, the operation code (first 6 bits) determines the specific instruction but there are eight different variations of most instructions, dependent upon the index designator. For this reason, it is probably better for the programmer to think of the complete FUNCTION CODE as containing both an operation code and an index designator (b).

The two operation codes which are not used (00 and 77) are not valid codes but can be used as fault designators which will stop computation and indicate malfunction when interpreted as operation codes.

Since index registers play such an important part in programming the 1604, it is necessary to define some of their general characteristics at this time. Six index registers are available for address modification of the base execution address. These index registers are numbered 1 through 6 and are selected by the corresponding octal number (1 through 6) placed in the index designator of the instruction. These index registers perform two different types of functions in the 1604 as defined below.

1. As address modifiers, the content of the designated index register is added to the base execution address before the computer interprets the specific instruction. (Addition is modulus 2^{15} minus one, one's complement)

2. As jump conditioners.

For programming facility, let us assume the operation code of 6 bits is separated from the index designator by an octal reference point as shown below.

06.0 Each one of these contains the same
06.1 operation code (06) but a different index
06.2 designator. Each index designator
06.3 designates a different functional operation
06.4 of the basic code (06). For this reason it
06.5 is customary to think of these as eight
06.6 variations of the same instruction or simply
06.7 as eight different instructions.

The operation code of six bits specifies the general character of the operation to be performed by the instruction. The index designator of three bits completes the function code so that the computer carries out a specific operational procedure. Of the possible 64 operation codes (six bits can hold 64 different codes), the 1604 uses 62 for general instructions. Since there are eight possible index designators (three bits can hold eight different codes 0 to 7), there are approximately 500 (8 x 62) different variations available.

It is important to note that regardless of the particular function they may serve, the use of index registers involves the contents of the index register which is designated except in the case of instructions 22, 23, 74, 75, and 76.

In addition to the codes 1 through 6, the index designator can also indicate 0 or 7. A zero index designation indicates that no index register is involved. (Exception: In some instructions, a designation of 0 may refer to a certain condition being fulfilled before a jump occurs.) An index designation of 7 will indicate that indirect addressing is to be used except in an instruction where the index designation defines a necessary condition for a jump and in the external function sense condition. (Indirect addressing is a means for expanding the reference capabilities of the instruction execution address. In indirect addressing, the instruction execution address specifies a storage location in which is located the operand address.)

Of particular significance to programmers with respect to the use of index registers is the fact that when the contents of the index register is added to the execution address, the addition is performed in a ones complement accumulator, modulus 2^{15} minus one. Use of index registers along with indirect addressing (for index 7) will be explained in detail for the individual instructions.

Operational Registers

The operational registers of the 1604 computer are defined as those registers which are capable of storing data from one instruction to another; the contents of those registers are displayed on the control console. These registers do not have special addresses (as is necessary in two or three address logic) but are referenced implicitly by the execution of the different instructions. However, it is important that the programmer know of these registers and of their functional use in the instructions.

Altogether ten different internal registers are available for auxiliary functions. Two of these are arithmetic registers, six are index registers, and two perform control functions. The ten are listed and described below:

Arithmetic Registers	A Register	48 bits
	Q Register	48 bits
Index Registers	Index 1.	15 bits
	Index 2.	15 bits
	Index 3.	15 bits
	Index 4.	15 bits
	Index 5.	15 bits
	Index 6.	15 bits
Control Registers	Program Control Register (U^1 upper)...	24 bits
	Program Address Register (P).........	15 bits

The A register, or accumulator, operates as a 48-bit subtractive accumulator during most of the arithmetic operations. Normal arithmetic operations are performed modulus ($2^{48}-1$). This register also has shifting capabilities which are explained under "shifting".

The Q register assists the A register in performing the more extensive arithmetic operations. For example, multiply, divide, and floating point instructions involve both the A and Q registers. The Q register serves as a "mask" during the logical operations and it also has shifting capabilities which are explained under "Shifting".

On some instructions the A and Q registers operate as one large register. For example, it is possible to shift A and Q as one register of 96 bits.

Six index registers are available for modification of execution addresses. In program loops, two approaches are possible in using these index registers for looping:

1. The contents of an index register may be advanced each pass through a loop and the exit is initiated when the contents of the index register attain a given threshold.

2. The contents of an index register may be preset to a certain amount and then reduced by one count each pass through the program loop until an exit is initiated upon the index register contents reading zero.

The two control registers are used to control the internal sequencing of instructions. The Program Control Register (U^1) holds the current instruction and interprets this instruction so that the proper execution can begin.

The Program Address Register (P) holds the storage location of the current program step during the interpretation and execution of an instruction contained in that program step. Upon completion of a step in the program, the program address is advanced one count to specify the location of the next step. FIG. 5 indicates the functional aspects of the U and P registers.

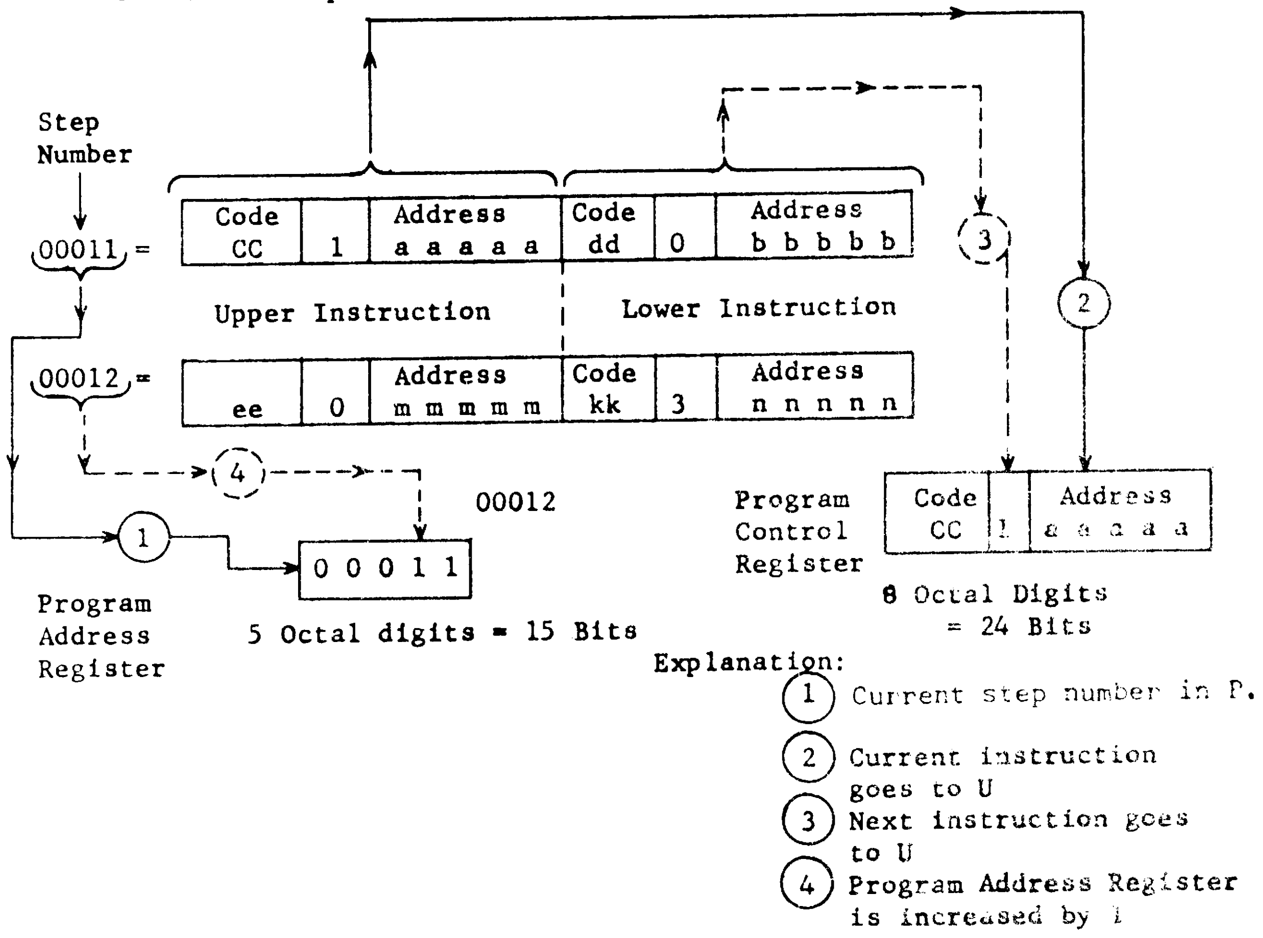


FIG. 5

Sequencing of P and U

Secondary Registers

In addition to the operational registers there is a group of twelve registers, not displayed on the control console, which also perform a vital function in machine operation. These are the secondary registers. Ordinarily they may be thought of as transfer registers, acting as intermediate storage for computer instructions or operands as they are manipulated during the execution of an instruction. The twelve registers include

Auxiliary program control register U^2	15 bits
Storage address register S^1	14 bits
Storage address register S^2	14 bits
Storage restoration register Z^1	48 bits
Storage restoration register Z^2	48 bits
Address buffer register R.	15 bits
Exchange register X	48 bits
Function register 0^0.	15 bits
Output (four provided) registers $0^{1,2,3,4}$ . .	48 bits

The auxiliary program control register, U^2, is an accumulator used in the modification of the execution address of the current instruction by the addition of the contents of an index register.

The even and odd 16,384 word memory units each has a storage address register, S^1 and S^2 respectively. These registers receive addresses of instructions from the address register, P, and addresses of operands from U^2.

Each even and odd memory unit also has a storage restoration register, Z^1 and Z^2 respectively, which holds the 48-bit word which is to be written in a given storage location.

The R register acts as an exchange register for transmissions involving the index registers. It is used for advancing or reducing the count in a given index register. During several instructions it is used to count repetitive operations. Floating-point instructions use R in performing arithmetic operations on the exponent or characteristic.

The X register is an exchange and auxiliary arithmetic register. All input and output data pass through the X register.

The external function register, 0^0, is used for exchanging control information with input-output equipment.

Four output registers, 0^1, 0^2, 0^3, and 0^4, are provided. 0^1, 0^2 and 0^3 are used for output buffer operations where the data are transmitted at the speed of the input-output equipment. Output register 0^4 handles all high-speed output transfer operations where the data are transferred at the internal speed of the computer.

CONTROL DATA MODEL 1604 BASIC INSTRUCTION LIST (omitting the eight possible variations of each instruction due to index designations 0 through 7)

ZRO	00	(not used)	SST	40	Selective Set
ARS	01	A Right Shift	SCL	41	Selective Clear
QRS	02	Q Right Shift	SCM	42	Selective Complement
LRS	03	Long Right Shift	SSU	43	Selective Substitute
ENQ	04	Enter Q	LDL	44	Load Logical
ALS	05	A Left Shift	ADL	45	Add Logical
QLS	06	Q Left Shift	SBL	46	Subtract Logical
LLS	07	Long Left Shift	STL	47	Store Logical
ENA	10	Enter A	ENI	50	Enter Index
INA	11	Increase A	INI	51	Increase Index
LDA	12	Load A	LIU	52	Load Index (upper)
LAC	13	Load A, Complement	LIL	53	Load Index (lower)
ADD	14	Add	ISK	54	Index Skip
SUB	15	Subtract	IJP	55	Index Jump
LDQ	16	Load Q	SIU	56	Store Index (upper)
LQC	17	Load Q, Complement	SIL	57	Store Index (lower)
STA	20	Store A	SAU	60	Substitute Address (upper)
STQ	21	Store Q	SAL	61	Substitute Address (lower)
AJP	22	A Jump	INT	62	Input Transfer
QJP	23	Q Jump	OUT	63	Output Transfer
MUI	24	Multiply Integer	EQS	64	Equality Search
DVI	25	Divide Integer	THS	65	Threshold Search
MUF	26	Multiply Fractional	MEQ	66	Masked Equality
DVF	27	Divide Fractional	MTH	67	Masked Threshold
FAD	30	Floating Add	RAD	70	Replace Add
FSB	31	Floating Subtract	RSB	71	Replace Subtract
FMU	32	Floating Multiply	RAO	72	Replace Add One
FDV	33	Floating Divide	RSO	73	Replace Subtract One
SCA	34	Scale A	EXF	74	External Function
SCQ	35	Scale AQ	SLJ	75	Selective Jump
SSK	36	Storage Skip	SLS	76	Selective Stop
SSH	37	Storage Shift	SEV	77	(not used)

Instructions are probably easier to learn if a few at one time are mastered along with an opportunity to use these in various short examples. This manual will follow this philosophy in the pages which follow. In addition, all floating-point instructions will be described under the chapter dealing with this topic (see Chapter on Floating Point).

It is also important to note that instructions will be presented with mnemonic codes as well as numeric codes. The mnemonic codes are for the convenience of the programmer and must be converted to the numeric codes when the final draft of a program is written.

Following each set of instructions, examples and exercises are presented and at the end of each chapter a general review test is presented. Solutions to these exercises and tests are presented in the appendixes.

CHAPTER I
Review Test

The following questions review the preliminary information presented in this chapter. Write the answers in the space provided and then check vour solutions with those presented in Appendix B.

1. The 1604 uses what type of internal storage? (Vacuum tubes, drum, or core)____________________

2. How many bits does each internal storage location hold?

3. What is the total number of internal storage locations?

4. The model 1604 computer is a two-address machine. (True-False)

5. What is the total range of addresses available for programming use?____________________

6. Sketch a 1604 instruction, showing the important parts and the number of bits for each part.

7. Describe or sketch the possible types of program steps.

8. Instructions must be written in octal format for input to the computer? (True-False)

9. The programmer can disregard the index register designation in some instructions? (True-False)

10. How many index registers are available?

11. Name two operational registers other than the index registers.

12. Which register holds the address only of the current instruction?

13. Which register holds the current instruction?

14. Which registers have shifting capabilities?

15. Are floating-point instructions available? (Yes or No)

16. Without considering the variations possible through index register designations, how many basic instructions are available?

17. What is the approximate minimum floor space required for the standard 1604 installation?

18. What are the standard input-output peripheral equipments used by the computer?

19. Since the computer contains transistors instead of electronic tubes, the power requirements are less. (True-False)

20. In some instructions it is possible to operate on 96 bits as though they made up the contents of one register. (True-False)

CHAPTER II

FIRST GROUP OF INSTRUCTIONS

FIRST INSTRUCTIONS (With Mnemonic and Numeric Codes) (where "b" represents the index designator which can be any number 0 through 7)

A RIGHT SHIFT	(ARS.b)	(01.b)	STORE A	(STA.b)	(20.b)
A LEFT SHIFT	(ALS.b)	(05.b)	STORE Q	(STQ.b)	(21.b)
Q RIGHT SHIFT	(QRS.b)	(02.b)	ADD	(ADD.b)	(14.b)
Q LEFT SHIFT	(QLS.b)	(06.b)	SUBTRACT	(SUB.b)	(15.b)
LOAD A	(LDA.b)	(12.b)	A JUMP	(AJP.b)	(22.b)
LOAD Q	(LDQ.b)	(16.b)	Q JUMP	(QJP.b)	(23.b)

The first four instructions are SHIFT instructions. These are not the only shift instructions - others will be described later. On the 1604 both end-around and end-off shifts are available. The term end-around indicates a shift in one direction (left on 1604) with bits coming out of one end of the register (left end) and being carried around to the other end (right) of the same register. No bits are discarded in an end-around shift. End-off shifts are shifts in one direction (right on 1604) with the bits being discarded as they come off one end (right) of the register. On the 1604 computer LEFT SHIFTS are end-around; RIGHT SHIFTS are end-off.

A RIGHT (ARSb) (01.b) (where b may be 0 through 7)

This instruction shifts the contents of the A register to the right the prescribed number of bit positions indicated by the sum of the execution address and the contents of the index register designated. The sign bit is extended (repeated) as the bits shift right. The lowest order bits are discarded as they are shifted out of the register. Shift counts greater than 127 (decimal) are treated as shift faults which may be sensed by the external function instruction. Shift faults do not stop the computer.

A LEFT (ALSb) (05.b) (where b may be 0 through 7)

This instruction shifts the contents of the A register circularly end-around to the left the prescribed number of bit positions indicated by the sum of the execution address and the contents of the index register designated. The lowest order bit positions, being vacated, are filled by the highest order bits as the shifting proceeds. No bits are discarded; those shifting out of the left end of the register are carried around to the right end. Shift counts greater than 127 (decimal) are shift faults which may be sensed by the external function instruction.

Q RIGHT (QRSb) (02.b) (where b may be 0 through 7)

This instruction shifts the contents of the Q register "end-off" to the right the prescribed number of bit positions indicated by the sum of the execution address and the contents of the index register designated. The sign bit is extended (to the right) as the bits shift right. Low order bits being shifted out of the register are discarded. A maximum shift count of 127 (decimal) results in a shift fault which may be sensed by the external function **instruction.**

Q LEFT (QLSb) (06.b) (where b may be 0 through 7)

This instruction shifts the contents of the Q register circularly "end-around" to the left the prescribed number of bit positions indicated by the sum of the execution address and the contents of the index register designated. The lowest order bit positions (being vacated) are filled with the higher order bits as the shifting proceeds. No bits are discarded - those shifting out of the left end of the register are carried around to the right end. Shift counts greater than 127 (decimal) cause a shift fault which may be sensed by the external function instruction.

The following examples should facilitate learning these four shift instructions. (Each student is advised to work out the examples and check the given solutions.)

<u>EXAMPLE I.</u> Assume the A register contains the octal quantity shown below. Show the instruction and the final octal contents of the A register after shifting 27_{10} bit positions to the right.

Given: Initial contents of A register $\left.\right\} \rightarrow \boxed{0\,0\;\;0\;\;0\,0\,0\,1\,7\;\;2\,2\,1\;\;0\,6\,2\,1\,3}$

$$\text{Instruction required} \longrightarrow \overbrace{0\;\;1\;\;0}^{\text{Function code}} \qquad \overbrace{0\,0\,0\,3\,3}^{\text{Execution address}} \qquad \text{Operand in this instruction}$$

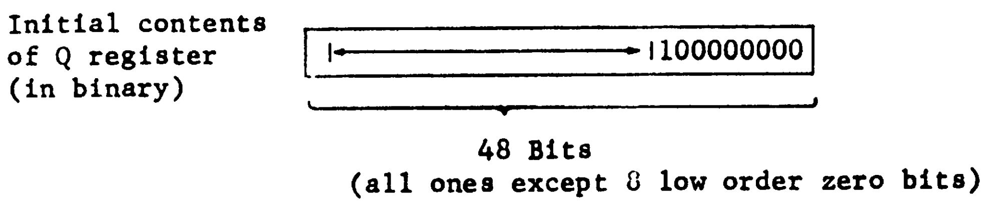
$$\underset{\text{(ARS0)}}{\uparrow} \qquad\qquad \text{(27 decimal = 33 octal)}$$

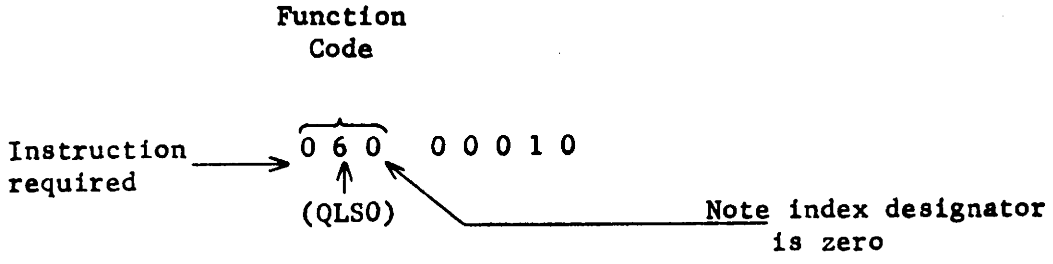
Final contents of A register $\left.\right\} \longrightarrow \boxed{0\,0\;\;0\;\;0\,0\,0\,0\,0\;\;0\,0\;\;0\;\;0\,0\,0\,0\,1}$

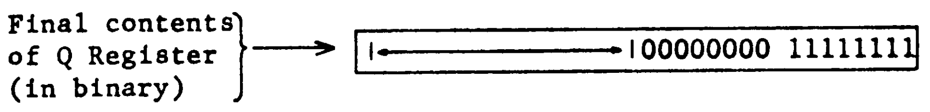
<u>Explanation:</u> A shift of 27 bit positions is equivalent to 9 octal places since 3 shifts (3 bits) equals 1 octal position. Since right shifts are "end-off", the 9 low order octal digits are discarded and the sign bits (in this case zeros) are extended. Also note that the index register designated in the above instruction is zero. Any combination of execution address and contents of index register whose sum is 33 (octal) would be correct. For example, an alternative solution to the above instruction might be 0 1 6 0 0 0 3 1 where Index Register 6 contains 0 0 0 0 2.

__EXAMPLE 2.__ Assume a programmer wants to shift the initial contents of the Q register (given below in binary form) right 40_{10} bit positions without discarding any of the given bits. Show the instruction needed and the final contents of the Q register after shifting.

Given:


Initial contents of Q register (in binary)
1100000000
48 Bits
(all ones except 8 low order zero bits)





__Explanation:__ Since no bits are to be discarded, a left shift instruction is required. However, the programmer wishes to shift right. Fortunately, for every desired right end-around shift there is an equivalent left end-around shift. The rule is to subtract the desired right shifts from 48 and the result is the equivalent number of left shifts. In this case 48-40 = 8. Then a left end-around of 8_{10} bit position gives the desires result. (8 decimal = 10 octal)

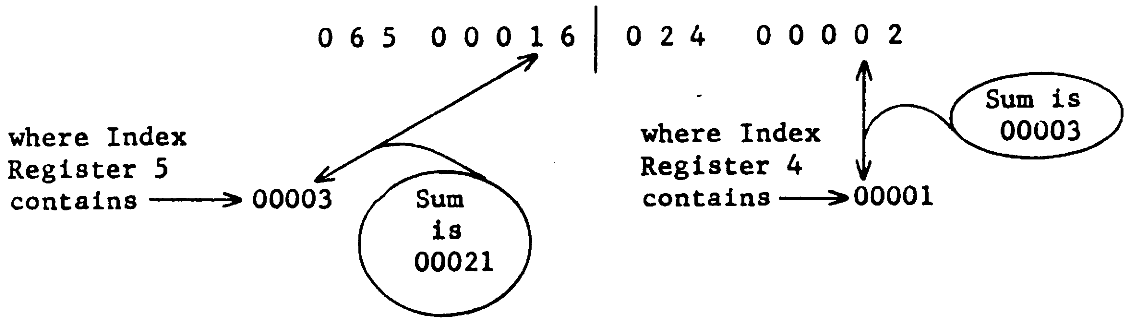
EXAMPLE 3. A programmer wishes to shift the contents of the Q register 17_{10} bit positions left, end-around, and then shift the resulting Q register contents 3 places right, end-off. Show the program step in octal to do this.

Function Code $\downarrow$ Function Code $\downarrow$

Required Program Step $\longrightarrow$

QLS-0	0 0 0 2 1	QRS-0	0 0 0 0 3

Explanation: The left shift instruction is programmed on the left side of the word since the left shift is desired first. The right shift instruction then follows on the right side of the word. Seventeen (decimal) = 21 (octal). Also note that other combinations could be used if the index designations are not zero. For example, an alternative solution might appear as:



EXAMPLE 4. A count is being generated in index register 5. At certain times within the program it is necessary to shift both the Q and A register left by the count in this index register. Show the program step which will do this.

Function Code $\downarrow$ Function Code $\downarrow$

Required program step

QLS 5 0 0 0 0 0	ALS 5 0 0 0 0 0

Explanation: Since the number of bit positions to be shifted is contained in index register 5, the index designator above becomes 5 and the execution address portion is zero. Thus the sum of the execution address and the contents of the index register is equivalent to the contents of the index register.

The following exercises are provided for practice on the four shift instructions presented on the previous pages. Work out the required solutions and check the answers against those shown in Appendix A. The number of correct answers will indicate the approximate mastery of these instructions by cross checking the Rating Table below.

Rating Table 1

If you have 4 or 5 correct answers	Excellent
If you have 3 correct answers.	Good
If you have 1 or 2 correct answers	Fair (Review)
If you have 0 correct answers.	Poor (Restart)

EXERCISE 1.

a. Write a 1604 instruction to shift Q 14_{10} bit positions right.

b. The A register contains 0 0 0 0 2 1 7 0 0 0 0 0 1 1 0 0. After shifting A left end-around 22_{10} bit positions, what will A contain? (Show answer in octal format.)

c. Write a program step which will first shift Q right 7 bits and then shift the new result in Q left by 12_{10} bit positions.

d. Write a 1604 instruction to shift A right by 5 bits without discarding any bits.

e. Would a right end off shift of 6 bits equal a left end-around shift of 42_{10} bit positions? Explain.

With a single address computer transferring data from one storage location to another is usually accomplished by loading a particular register and storing the contents of this register somewhere else. The next four instructions indicate the basic transfer instructions on the model 1604 computer. More powerful transfer instructions will be presented later.

LOAD A (LDA.b) (12.b) (where b may be 0 through 7)

This instruction replaces the A register contents with an operand whose location is specified by the sum of the execution address and the contents of the designated index register. A storage reference is made to obtain the 48-bit quantity at the location indicated by the sum of the execution address, and the contents of the designated index register. A is cleared, the 48-bit quantity is then loaded into the A register. The memory location remains unchanged.

LOAD Q (LDQ.b) (16.b) (where b may by 0 through 7)

This instruction replaces the Q register contents with an operand whose location is specified by the sum of the execution address and the contents of the designated index register. A storage reference is made to obtain the 48-bit quantity indicated by the sum of the execution address and the contents of the designated index register. Q is cleared, the 48-bit quantity is then loaded into the Q register. The memory location remains unchanged.

STORE A (STA.b) (20.b) (where b may be 0 through 7)

This instruction stores the contents of the A register at the storage location specified by the sum of the execution address and the contents of the designated index register. The A register contents are not modified by this instruction - that is, A still holds its initial contents upon the completion of the instruction.

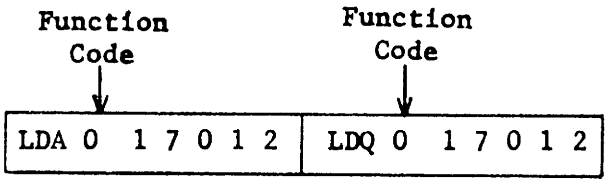
STORE Q (STQ.b) (21.b) (where b may be 0 through 7)

This instruction stores the contents of the Q register at the storage location specified by the sum of the execution address and the contents of the designated index register. The Q register contents are not modified by this instruction - that is, Q still holds its initial contents upon the completion of the instruction.

The following examples should facilitate learning these basic transfer instructions. (Each student is advised to work out the examples and check the given solutions.)

EXAMPLE 5. Assume zeros are contained at storage location 17012. Show the program step which will load these zeros into the A and Q registers.

Program step required

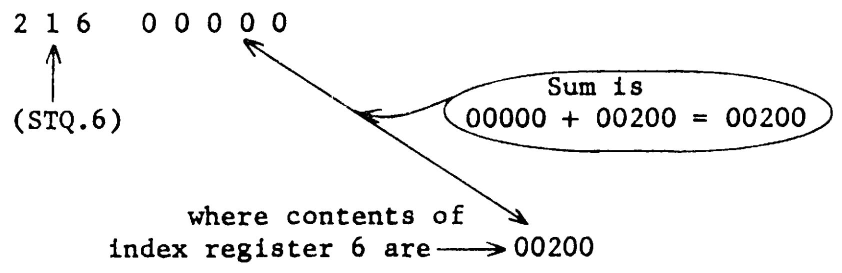


Explanation: In each instruction above, the computer will reference the storage (17012) given in the execution address and load the contents of 17012 into the A or Q register depending upon the Code 12 or 16. Note that the index designators are zeros above. This is not a necessary condition. An index designation other than zero is satisfactory if the contents of the designated index register plus the execution address equals 17012 above. For example, 12 3 17000 is equivalent to the left instruction above if the contents of index register 3 are 00012.

EXAMPLE 6. At storage location 00017 is the constant 1. The programmer desires to transfer this constant to storage location 00200. He does not want to disturb the present contents of the A register. Show the program step which will do this.

	Function Code	Function Code
Program step required →	LDQ 0 0 0 0 1 7	STQ 0 0 0 2 0 0

Explanation: The upper instruction loads the contents of 00017 into the Q register. The lower instruction then stores the contents of Q (which now contains the contents of 00017) into 00200, completing the transfer. Note the index designators are both zero. This is not a necessary condition. For example, if index register 6 contains 00200, the lower instruction above could be:


2 1 6   0 0 0 0 0
(STQ.6)
Sum is 00000 + 00200 = 00200
where contents of index register 6 are → 00200

EXAMPLE 7. An address is being generated in index register 4. Another address is being generated at index register 1. A programmer wants to store the contents of the generated address of index register 4 in the generated address of index register 1. He does not want to disturb the A register while doing this. Show the program step required.

	Function Code	Function Code
Required program step	LDQ 4 0 0 0 0 0	STQ 1 0 0 0 0 0

Explanation: Since A is not to be disturbed, the Q register instructions are used. The upper instruction loads Q with the contents of the generated address of index register 4. The lower instruction then stores the contents of Q into the generated address which is the contents of index register 1.

Did the contents of index registers 4 and 1 change because of this program step? What did change?

The following exercises are provided for practice on the four basic transfer instructions presented in this chapter. Work out the solutions and check the answers against those in Appendix A. The following Table indicates your approximate progress in learning to program these instructions.

Rating Table 2

If you have 4 correct answers	Excellent
If you have 3 correct answers	Good
If you have 0-2 correct answers	Review

EXERCISES 2.

a. Given, index register 3 contains 00005. What will the instruction, 12300010 accomplish?

b. Write a program step which will transfer the contents of storage location 01000 to location 01001 without disturbing the A register.

c. Index register 2 contains 77767.
What will instruction 20200020 accomplish?

d. What is the difference in effect upon the initial contents of A or Q when LOAD and STORE instructions are programmed?

The last four instructions of this chapter include two arithmetic and two jumps. The arithmetics are the basic ADD and SUBTRACT instructions. The jumps are the A-JUMP and Q-JUMP instructions. Several other instructions similar to both these types are available in the 1604 instruction repertoire. These will be presented in later chapters.

ADD (ADDb) (14.b) (where b may be any number 0 through 7)

This instruction adds a 48-bit operand to the previous contents of the A register. A storage reference is made to obtain the 48 bit quantity at the location specified by the sum of the execution address and the contents of the designated index register. The operand at this location is then added to the previous contents of the A register. An overflow fault results when the sum of two quantities exceeds the capacity of A.

SUBTRACT (SUBb) (15.b) (where b may be any number 0 through 7)

This instruction subtracts a 48-bit operand from the previous contents of the A register. A storage reference is made to obtain the 48-bit quantity at the location specified by the sum of the execution address and the contents of the designated index register. The operand at this location is then subtracted from the previous contents of the A register. An overflow fault results when the difference of two quantities exceeds the capacity of A.

A-JUMP (AJPb) (22.b) (where b may be any number 0 through 7)

This is the first instruction, to this point, in which the index registers are not used to modify the execution address. In this instruction, the 3-bit index register designator specifies the particular condition of the A register which will cause a jump in the program address. The conditions for the different possible index designations are as follows:

<u>If Index register designation is:</u>

0	Jump will occur if content of A is zero
1	Jump will occur if content of A is not zero
2	Jump will occur if content of A is positive
3	Jump will occur if content of A is negative
4	Return Jump will occur if content of A is zero
5	Return Jump will occur if content of A is not zero
6	Return Jump will occur if content of A is positive
7	Return Jump will occur if content of A is negative

Q-JUMP (QJPb) (23.b) (where b may be any number 0 through 7)

This is the same as the A-Jump except the jump occurs if the Q register is in a certain condition. The index register designator specifies the condition which will cause the jump. The various index designations with the corresponding conditions are listed below:

__If the Index register designation is:__

0	Jump occurs if Q register content is zero
1	Jump occurs if Q register content is not zero
2	Jump occurs if Q register content is positive
3	Jump occurs if Q register content is negative
4	Return Jump occurs if Q register content is zero
5	Return Jump occurs if Q register content is not zero
6	Return Jump occurs if Q register content is positive
7	Return Jump occurs if Q register content is negative

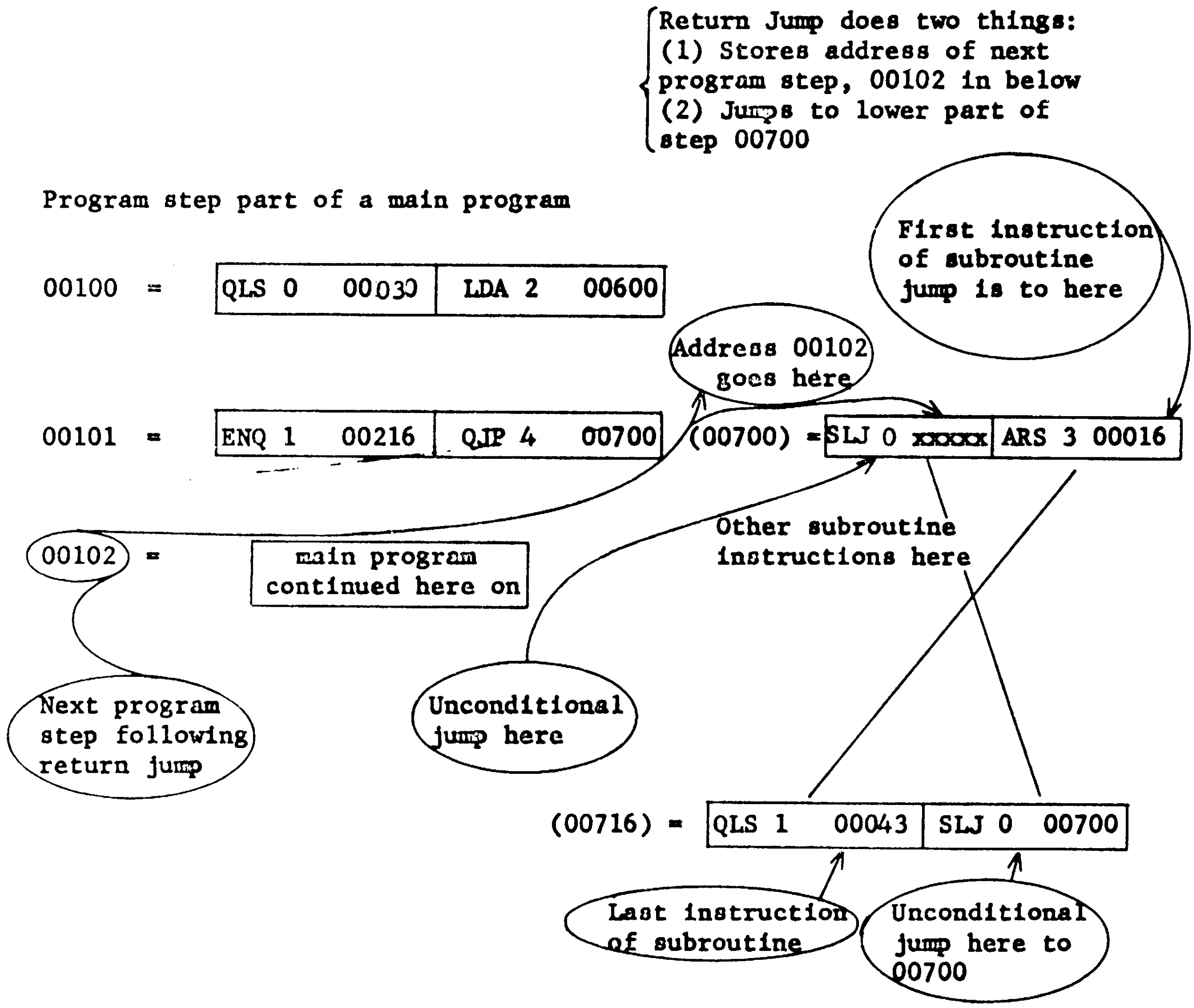
Before trying examples of these last four instructions, some of the general characteristics of jump instructions should be considered. A jump instruction causes the termination of a current program sequence and the initiation of a new sequence at a different location in storage. The program address register (P) provides the continuity between program steps. This register always contains the storage location of the program step currently being executed. Normally, the address in this register is increased by one count at the end of each program step in order to indicate to the machine the location of the next step. When a jump instruction is given, the program address register is cleared, and a new address is entered from the jump instruction. In all jump instructions the base execution address specifies the beginning address of the new program sequence.

Some of the jump instructions are conditional upon a register containing a specific value or upon the position of an operator key on the console. If the criterion is satisfied, the jump takes place to the location specified by the execution address. If the criterion is not satisfied, the program proceeds in its regular sequential order to the next instruction.

A jump instruction may appear in either position in a program step. If the jump instruction appears in the first (upper) part of the program step, and the jump is taken, the second (lower) part of that program step is never executed. If the jump instruction appears in the second (lower) part of the program step, the first (upper) part of that step is executed in the normal manner.

Return Jumps are different from normal jumps in that provision is made to "return" to the location following the place where the return jump was initiated. This operation is important in programming since one often wishes to exit from the main program to some routine and then return to continue the main program. In all such situations there must be some way for the computer to "remember" where the Return Jump originated.

On the 1604, a return jump begins a new program sequence at the second (lower) part of the program step to which the jump is made; that is, the jump is always to the lower part of the program step specified in the Return Jump instruction. At the same time the base execution address in the upper part of this same program step is replaced with the address of the location of the return jump instruction plus one. This allows the new program sequence to return to the step following the return jump instruction and resume the original program at a later time. FIG. 6 indicates the return jump sequence.



Return Jump does two things:
(1) Stores address of next program step, 00102 in below
(2) Jumps to lower part of step 00700
Program step part of a main program
00100 = QLS 0 00030 LDA 2 00600
First instruction of subroutine jump is to here
Address 00102 goes here
00101 = ENQ 1 00216 QJP 4 00700
(00700) = SLJ 0 xxxxx ARS 3 00016
Other subroutine instructions here
00102 = main program continued here on
Next program step following return jump
Unconditional jump here
(00716) = QLS 1 00043 SLJ 0 00700
Last instruction of subroutine
Unconditional jump here to 00700

FIG. 6

Return Jump Operation

The following examples are provided to afford practice on the last four instructions of this Chapter (Add, Subtract, A-Jump, and Q-Jump). (Each student is advised to work out the examples and check the given solutions.)

EXAMPLE 8. Assume the initial contents of the A register are 000 00000 021 00600.

A programmer wishes to add the contents of storage location 00200 to the contents of A. Show the instruction required if (a) no index register is to be used (b) if index register 3 (which holds 00171) is to be used.

(a)

Instruction required	ADD 0	0 0 2 0 0

(b)

Instruction required	ADD 3	0 0 0 0 7

Explanation: (a) One uses the ADD instruction (code 14.x) with the index designation equal to zero. This adds the contents of the location specified by the execution address (00200) to the A register as the problem required.

(b) If an index register is used the contents of this designated index register is added to the base execution address and this sum specifies the location of the operand which is to be added to the A register. In this case since the sum must be 00200, and since index register 3 contains 00171, it is necessary to make the execution address be 00007. (00007 + 00171 = 00200 .)

EXAMPLE 9. Assume the A register contains 203 00600 060 00400. Assume address 00300 contains 000 00000 020 00300. After the completion of instruction 150 00300 show the contents of (a) the A register (b) storage location 00300.

(a) The given instruction subtracts the contents of 00300 from the initial contents of the A register as shown below:

```
2 0 3   0 0 6 0 0   0 6 0 0 0 4 0 0     Initial contents of A
0 0 0   0 0 0 0 0   0 2 0 0 0 3 0 0
-----------------------------------
2 0 3   0 0 6 0 0   0 4 0 0 0 1 0 0     Final contents of A
```

(b) Final contents of 00300 is 000 00000 020 00300.

Explanation: (a) The A register is changed during the execution of the instruction since the arithmetic process (subtraction) takes place and the result (the difference) is left there.

(b) The subtraction does not change the quantity being subtracted from the A register. Thus the final contents of address 00300 is the same as its initial contents.

EXAMPLE 10. A programmer is generating a quantity in the Q register. At program step 00100, he wants to test the contents of Q for two conditions: (1) if the content of Q is zero, he wishes to jump to step 00600, and (2) if the content of Q is negative, he wishes to jump to step 00700. Show the instructions required to do this.

Instructions
required

(00100)	QJP 0 0 0 6 0 0	QJP 3 0 0 7 0 0

Explanation: Since he is testing Q, he uses the Q-Jump. In the upper instruction above, Q is tested for zero by placing a zero in the index designator position. If Q contains zero, the jump will go to address 00600. If Q does not contain zero, the lower instruction above will be executed. The lower instruction has a 3 in the index designator position. This causes a jump to address 00700 if the content of Q is negative. If neither condition above is satisfied, instructions of the next step (00101) are executed.

EXAMPLE 11. A random number is being generated in the A register. Each time a random negative number is generated, a jump is made to a subroutine starting at subroutine step 00300 and exiting at sub-routine step 00304. Upon exit, a return is made to the main program. Assuming the test of the contents of A is made at main program step 01000, show or explain the following:

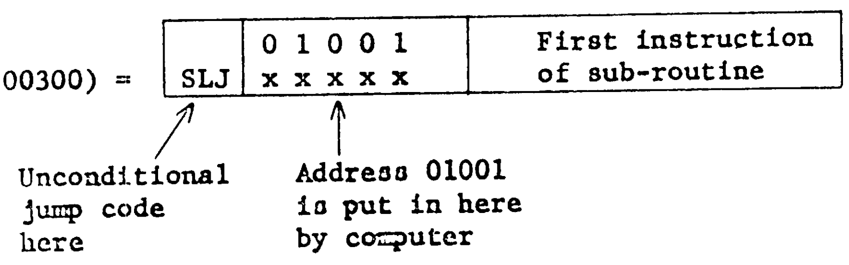
(a) Test instruction at step 01000.
(b) What happens at sub-routine step 00300?
(c) What happens at sub-routine step 00304?

(a)

(01000) = $\boxed{\text{AJP 7} \quad \text{0 0 3 0 0}}$ (This could be the upper or lower instruction of step 01000)

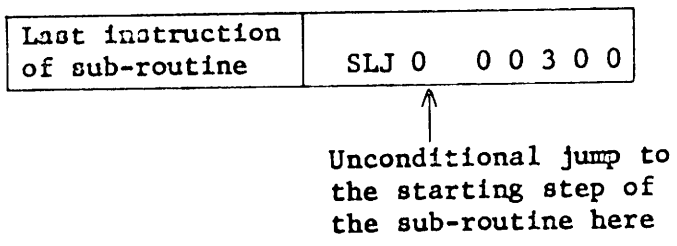
(b)

(00300) =


SLJ
0 1 0 0 1
x x x x x
First instruction of sub-routine
Unconditional jump code here
Address 01001 is put in here by computer

(c)

(00304) =


Last instruction of sub-routine
SLJ 0 0 0 3 0 0
Unconditional jump to the starting step of the sub-routine here

Explanation: (a) Index designator, 7, in the A-Jump tells the computer there is to be a Return Jump if A content is negative. The jump goes to the lower instruction of the entrance step of the sub-routine.

(b) The Return Jump automatically stores the contents of the "Program Address Register plus one" (01001 in this example) in the execution address portion of the upper instruction of the subroutine entrance step.

(c) The exit step of the subroutine should contain an unconditional jump to the upper instruction of the entrance step.

It is important to note at this time that all return jumps on the 1604 are not conditional upon the contents of the A or Q register. An unconditional return jump is available and will be explained in a later chapter.

The following exercises are provided for practice on the last four instructions presented in this chapter. Work out the solutions and check your results with those in Appendix A. The following table indicates your approximate progress in learning to program these four instructions.

Rating Table 3

If you have 5 correct answers	Excellent
If you have 3 or 4 correct answers.	Good
If you have 2 correct answers	Fair (some review)
If you have 0 or 1 correct answers.	Review thoroughly

EXERCISES 3.

a. Write an instruction which will add the contents of storage location 00030 to the contents of the A register.

b. Write program steps which will develop the result "G" in the A register, where G = (A reg.) - (B) - (C) + (D) + (E). Assume B = contents of 00041, C = contents of 00042, D = contents of 00043, and E = contents of 00044.

c. A programmer wishes to jump to address 00050 only if the contents of both A and Q are zero. Can he use the A-Jump and Q-Jump in the same program step? Explain your answer.

d. An address is generated and stored in index register 5. One wants to add the contents of this generated address to the contents of A. The A register is then tested for zero and if it is not zero, a jump occurs to program step 00105. Write one program step, of two instructions, which will accomplish this.

e. Explain what will occur if the following instruction is referenced at step 00304 and the contents of the A register are positive.

	Upper Instruction	Lower Instruction
(00304) =	AJP 6 0 0 0 1 0	

CHAPTER II

REVIEW TEST

The following questions review the twelve instructions presented in this chapter. Write answers in the space provided after each question. Solutions are given in Appendix B.

1. What happens to some bits in an "end-off" or "open-ended" shift?

   ____

2. All shifts on the 1604 are end-off shifts. (True-False)

   ____

3. What happens to the sign bits when a right shift occurs on the 1604?

   ____

4. Where does the shifting on the 1604 actually take place?

   ____

5. Is it possible to shift 200 (decimal) bit positions with one instruction 1604? (Yes or No)

   ____

6. The index designator is not significant with the shift instruction. (True-False)

   ____

7. If the index designator is 3, you add 3 to the execution address before interpreting the instruction. (True-False)

   ____

8. Storage address 00300 contains 0000000000000777. The programmer shifts the contents of 00300 right 6 bit positions with the QRS.0 instruction. (Q Right.) After storing the shifted result of Q back in 00300, show the contents of 00300.

   ____

9. Show the final contents of Q for the above exercise.

   ____

   ____

10. Using the instructions of this chapter show two methods to transfer the contents of 00050 to the contents of 01000. (Use one program step for each method.)

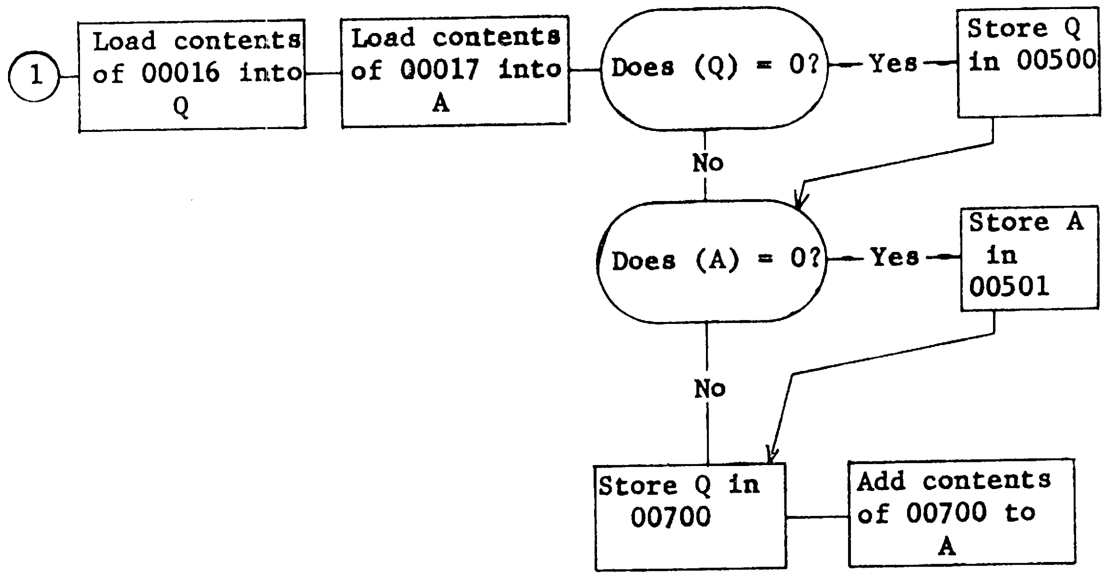
11. Why is there a STORE Q instruction as well as a STORE A instruction when each perform essentially the same function?

12. Index register 4 contains 00006. Address 00100 contains 0000000000000001. Address 00007 contains all zeros. If the LOAD A with index designation 4 is given $\boxed{\text{LDA 4 00072}}$ show the final contents of A below.

13. The instruction "STORE Q" with index designation 6 is given (STQ.6). The execution address portion of the instruction is zero. In other words, the instruction is 21 6 00000 (Code for "STORE Q" is 21.) Where will the contents of Q be stored when this instruction is executed?

14. If an index register is designated in the ADD instruction, the contents of the designated index register are added to the previous contents of the A register. (True-False)

15. Translate the following flow diagram into 1604 program steps starting with step 00100 below.



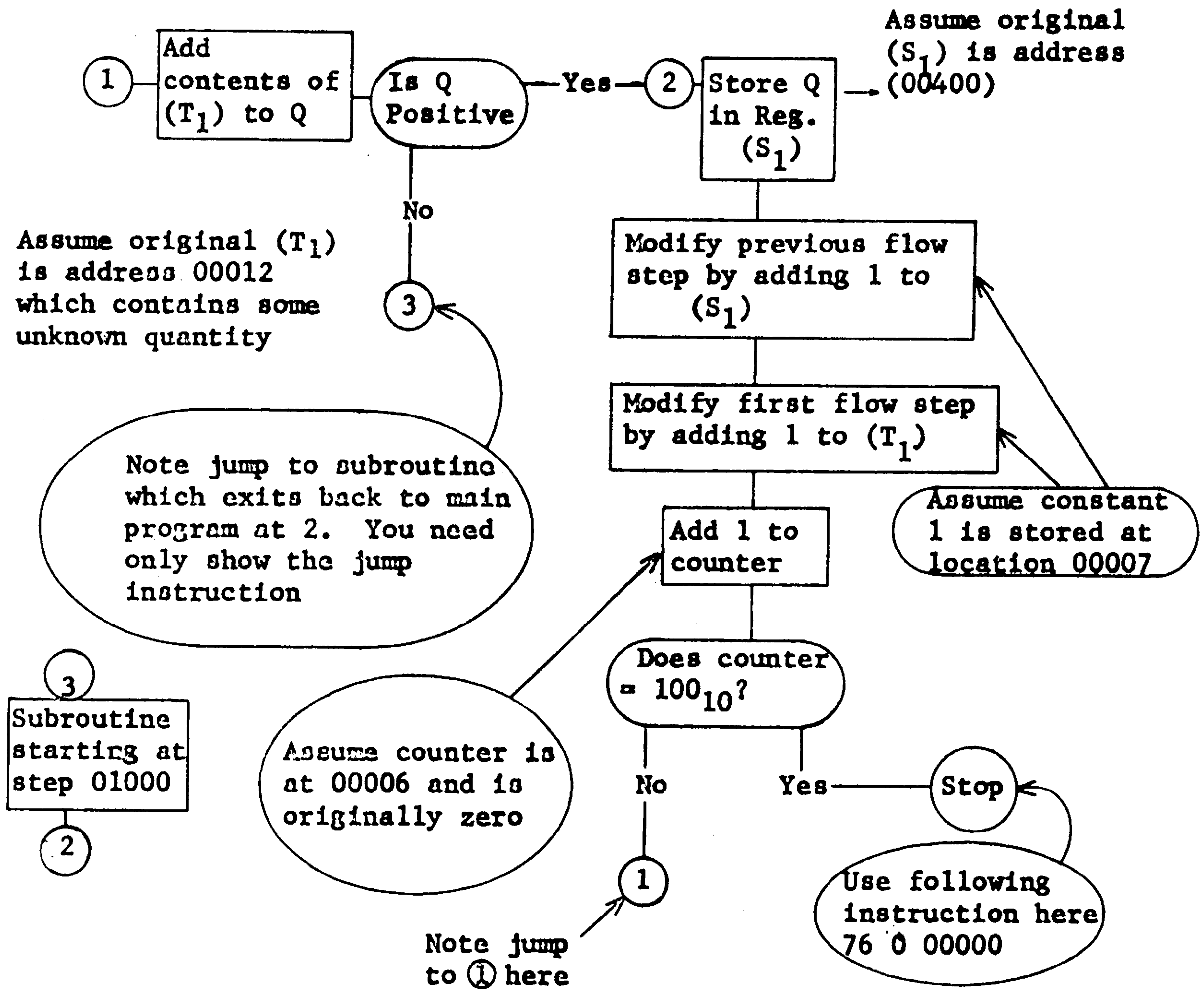
(00100) =

(00101) =

(00102) =

(00103) =

16. **Translate the following flow diagram into 1604 program steps starting with step 00100 below.**



1
Add contents of (T_1) to Q
Is Q Positive
Yes
2
Store Q in Reg. (S_1)
Assume original (S_1) is address (00400)
No
Assume original (T_1) is address 00012 which contains some unknown quantity
3
Modify previous flow step by adding 1 to (S_1)
Modify first flow step by adding 1 to (T_1)
Assume constant 1 is stored at location 00007
Note jump to subroutine which exits back to main program at 2. You need only show the jump instruction
Add 1 to counter
3
Subroutine starting at step 01000
2
Does counter = 100_{10}?
Assume counter is at 00006 and is originally zero
No
Yes
Stop
1
Use following instruction here 76 0 00000
Note jump to ① here

(00100) =

(00101) =

CHAPTER III

SECOND GROUP OF INSTRUCTIONS

SECOND INSTRUCTIONS (with Mnemonic and Numeric Codes) (where b represents the index designator and may be 0 through 7).

LONG RIGHT SHIFT	(LRS.b)	(03.b)	INCREASE A	(INA.b)	(11.b)
LONG LEFT SHIFT	(LLS.b)	(07.b)	MULTIPLY INTEGER	(MUI.b)	(24.b)
ENTER A	(ENA.b)	(10.b)	DIVIDE INTEGER	(DVI.b)	(25.b)
ENTER Q	(ENQ.b)	(04.b)	SELECTIVE JUMP	(SLJ.b)	(75.b)
LOAD A, COMP.	(LAC.b)	(13.b)	SUBSTITUTE ADDRESS(UPPER)	(SAU.b)	(60.b)
LOAD Q, COMP.	(LQC.b)	(17.b)	SUBSTITUTE ADDRESS (LOWER)	(SAL.b)	(61.b)

Of the first six instructions, two are shift instructions and four are transfer instructions. Since basic shift and transfer instructions were given in detail in Chapter II, it is reasonable to assume that these six instructions can be easily learned in one unit.

LONG RIGHT SHIFT (LRS.b)(03.b)(where b may be any number 0 through 7)

This instruction shifts the contents of the A and Q registers to the right as one 96-bit register. The A register is treated as the leftmost 48 bits and the Q register as the rightmost 48 bits. The number of bit positions to be shifted is specified by the sum of the execution address and the contents of the designated index register. The sign bit of the A register is extended as the shift is performed. The lowest order bits of the A register shift into the highest order bit positions of the Q register as the shift is performed. The lowest order bits of the Q register are discarded as they are shifted out of the right end of the Q register. Shift counts greater than 127 (decimal) are treated as shift faults and produce an indication which may be sensed by the external function instruction.

LONG LEFT SHIFT (LLS.b) (07.b) (where b may be 0 through 7)

This instruction shifts the contents of A and Q registers circularly end-around to the left as one 96-bit register. The number of bit positions shifted is specified by the sum of the execution address and the contents of the designated index register. The leftmost bits of the Q register shift into the rightmost bit positions of the A register as the shifting is performed. The bits coming off the left-end of the A register circulate end-around to the rightmost bit positions of the Q registers as the shifting is performed. Shift counts greater than 127 (decimal) are treated as shift faults and produce an indication which may be sensed by the external function instruction.

ENTER A (ENA.b) (10.b) (where b may be 0 through 7)

This instruction enters the sum of the execution address portion of the instruction and the contents of the designated index register into the A register. The A register is cleared first, and the sum of the execution address and content of the designated index register is then entered into the cleared A register as a 15 bit quantity including sign (total 15 bits). The highest order bit (or sign bit) is then duplicated in the remaining (33) higher order bits of the A register. The sketch below in FIG. 7 indicates the process.

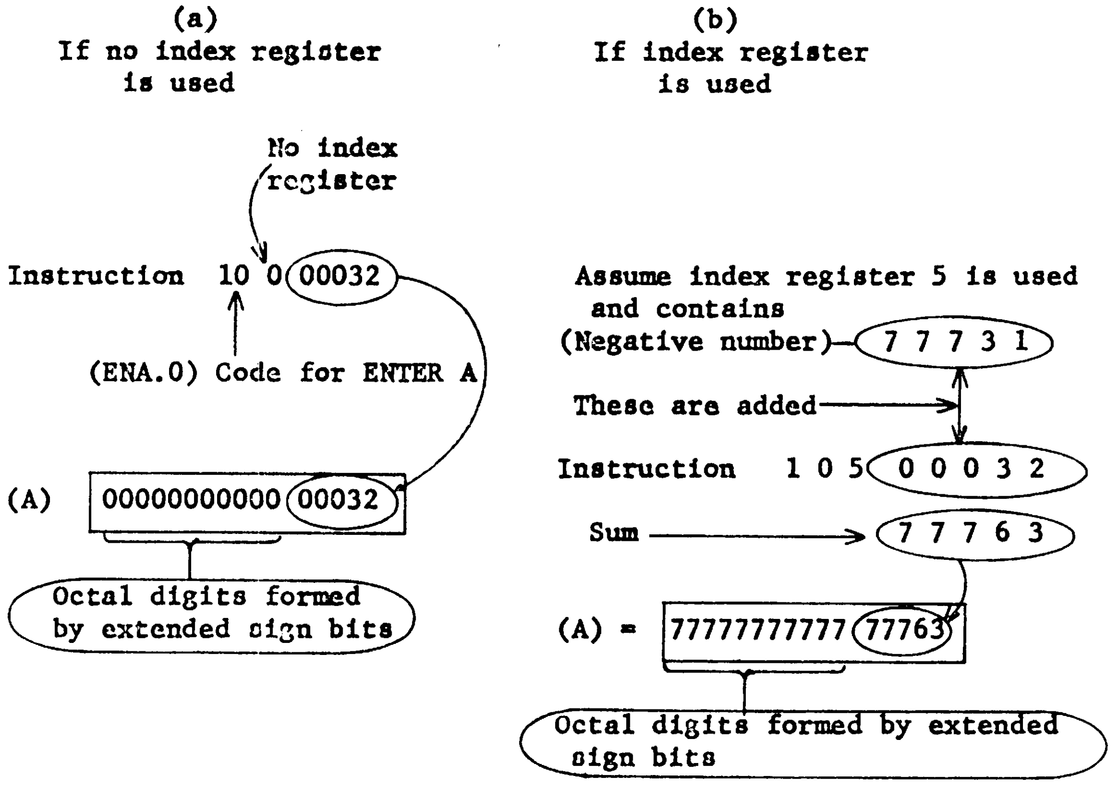

(a)
If no index register is used
(b)
If index register is used
No index register
Instruction 10 0 00032
(ENA.0) Code for ENTER A
(A) 00000000000 00032
Octal digits formed by extended sign bits
Assume index register 5 is used and contains
(Negative number) 7 7 7 3 1
These are added
Instruction 1 0 5 0 0 0 3 2
Sum 7 7 7 6 3
(A) = 77777777777 77763
Octal digits formed by extended sign bits

FIG. 7

Execution of ENTER A instruction

ENTER Q (ENQ.b) (04.b) (where b may be 0 through 7)

This is the same as the ENTER A instruction except the Q register is used. The sum of the execution address portion of the instruction and the content of the designated index register is entered into the Q register as a 15 bit quantity including sign (total 15 bits). The highest order bit (sign bit) is then duplicated in the remaining higher order bits of the Q register. FIG. 7 indicates the process except the Q register is used in place of A.

LOAD A, COMP (LAC)(13.b) (where b may be 0 through 7)

This instruction replaces the A register contents by the complement of a 48-bit operand whose location is specified by the sum of the execution address and the contents of the designated index register. After the 48-bit operand is located it is complemented and entered into the A register. A zero index designation (no index register is used) results in the 48-bit operand being found at the location specified by the execution address portion of the instruction.

LOAD Q, COMP (LQC) (17.b) (where b may be 0 through 7)

This is the same as LOAD A, COMP except the Q register is used. Then the content of the Q register is replaced by the complement of a 48-bit quantity whose location is specified by the sum of the execution address and the contents of the designated index register. A zero index designation (no index register is used) results in the 48-bit operand being found at the location specified by the execution address portion of the instruction.

The following examples should facilitate learning the previous six instructions. (Each student is advised to work out the given examples and check the given solutions.)

EXAMPLE 12. Assume the A and Q registers contain the quantities given below. A programmer wants to shift both these quantities right 39_{10} bit positions as though both made up one 96-bit register. Show the instruction used, and the final contents of A and Q after the shift is performed.

	A Register	Q Register
Given contents of A and Q in octal	$\boxed{\text{00472715 13002777}}$	$\boxed{\text{7771111100003333}}$

$$\begin{tikzcd} \text{Required instruction} & \text{LRSO 00047} & 39_{10} = 47_{8} \arrow[l, bend right] \\ & \text{Shift AQ Right Code} \arrow[u] & \end{tikzcd}$$

	A Register	Q Register
Final contents of A and Q in octal	$\boxed{\text{0000000000000004}}$	$\boxed{\text{7271513002777777}}$

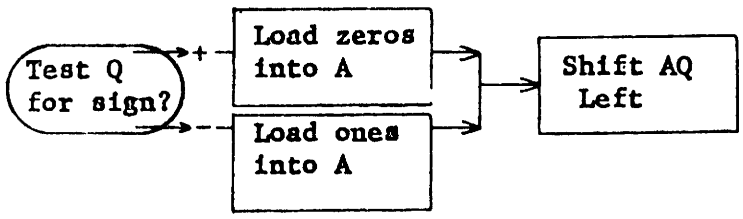
Explanation: Since one wants to shift 39 decimal bit positions, this is equivalent to a shift of 13 octal digits (since 3 bit positions equal one octal place). Since the shift is RIGHT (end-off), there are 13 octal digits discarded as they shift off the right end of Q. Also 13 low order octal digits of A go into the high order positions of Q. The sign bits (zeros) in A are extended to make the octal digits in A.

EXAMPLE 13. Assume the Q register contains a generated number. The programmer desires to double this number by shifting Q left one bit position. However, if there is a significant bit next to the sign bit, a left shift of one will "overflow" the Q register and change the sign of the number. For example, if 01101110111←→1

|←—48 bits—→|

is contained in Q, a shift of one left would give 1101110111←→10, which has changed the sign from plus to minus by the "overflow". In order to eliminate this possibility, he decides to shift A and Q as one register. Show the instructions required. (Hint: the A register must be filled with bits all of which are the same as the sign bit of the initial contents of Q.)

Required flow steps



Required steps — Jump to 00202, if Q is plus — Enter A with operand shown (all ones)

(00200) =	QJP 2 0 0 2 0 2	ENA 0 7 7 7 7 7
	Shift AQ Left once	
(00201) =	LLS 0 0 0 0 0 1	Unconditional jump to 00203 to continue
(00202) =	Enter Zeros into A ENA 0 0 0 0 0 0	Shift AQ left once LLS 0 0 0 0 0 1

(00203) = Continue program

Explanation: The first instruction tests the contents of Q for plus or minus. If plus, jump to (00202), where zeros are entered into A by 10 0 00000 and then A and Q are shifted left by 1 as though they made up one 96-bit register. If the jump at 00200 did not occur, then Q is negative and all ones are entered into A by 10 0 77777, and the AQ shift then takes place. (There are several alternative approaches which can be used to do the above.)

EXAMPLE 14. A count has been generated in index register 3. The programmer wants to know if this count is equal to the amount in the Q register. Show instructions which could be used.

Instructions required

This code stores contents of Q at 00100

This code enters the sum of contents of index register 3 and amount shown in execution address into A

```
(00300) =  | STQ 0  0 0 1 0 0 | ENA 3  0 0 0 0 0 |
```

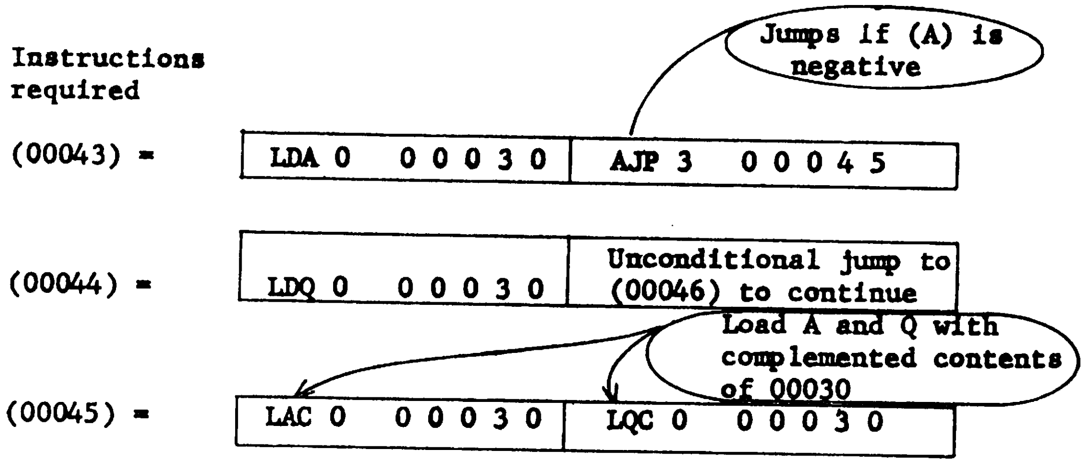
This code subtracts contents of address shown from A

This code jumps to address shown if contents of A register are zero

```
(00301) =  | SUB 0  0 0 1 0 0 | AJP 0  0 0 4 0 0 |
```

Explanation: The contents of Q are first stored at some address (in this case 00100). Then the "ENTER A" instruction with index designation 3 and execution address 00000, places in A, the contents of index register 3 plus the quantity shown in the execution address. (Contents of index reg. 3+ 00000 = contents of index register 3.) In the next step, the contents of 00100 (which now equal contents of Q) are subtracted from A (which now equals the contents of index reg. 3). The last instruction jumps to same address if the result of the subtraction is zero. If the jump occurs, one knows the contents of Q and index register 3 are equivalent. (Alternative solutions are possible here.)

EXAMPLE 15. Assume one wants to load the absolute magnitude of the quantity located at address 00030 into the A and Q registers. Show the instructions required. Hint: a test must first be made of the quantity in 00030 to see if it is positive or negative. If negative, it must be complemented before loading it into A and Q.



Instructions required

(00043) =	LDA 0 0 0 0 3 0	AJP 3 0 0 0 4 5
(00044) =	LDQ 0 0 0 0 3 0	Unconditional jump to (00046) to continue
(00045) =	LAC 0 0 0 0 3 0	LQC 0 0 0 0 3 0

(00046) = Continue program

Explanation: After the test for sign, a jump to 00045 occurs if the contents of 00030 are negative, and the contents of 00030 are complemented and loaded into A and Q. If jump does not occur then the contents of 00030 (already in A) are positive and need only be loaded into Q. (Alternative solutions are possible.)

The following exercises are provided for practice on the first six instructions of this chapter. Work out the given problems and check your answers with the solutions given in Appendix A. The number of correct answers will indicate your approximate progress by checking the Rating Table below:

Rating Table 4

If you have 5 or 6 correct answers	Excellent
If you have 3 or 4 correct answers	Good
If you have 1 or 2 correct answers	Fair - Review
If you have 0 correct answers	Complete Review

EXERCISES 4

a. Assume Q contains a positive 3, and A contains negative 7. As one 96-bit register shift A and Q right 4 bit positions. Show the instruction and the final contents of A and Q.

b. Assume Q contains all ones and A contains all zeros. As one 96-bit register shift A and Q left 24_{10} bit positions. Show the instruction and the final contents of A and Q.

c. Assume index register 5 contains 01326. Assume storage address 01111 contains 0000000000007562. Show the final contents of register A after completion of each of the following instructions.

```
         (ENA.5)
            ↓
    (1)  1 0 5   0 0 0 0 0
                              (Both instructions contain the
         (ENA.5)                "ENTER A" code)
    (2)  1 0 5   0 0 1 0 0
```

d. Mark the following True or False.

(1.) The same instruction cannot shift A and Q in two different directions. ______________________

(2.) Usually a left shift of both A and Q of one bit position is equivalent to multiplying the 96 bits in AQ by 2.

____

3. ENTER A and LOAD A instructions will do the same thing if the same execution addresses and index designators are used in each instruction.

   ____

4. LOAD Q COMP instruction will always load Q with a positive quantity. ______________________________

e. Assume register 00200 contains 0000000000007777. Show the final contents of Q after each of the following instructions:

1	04000200	(Enter Q code)
2	17000200	(Load Q COMP code)
3	16000200	(Load Q code)

The last six instructions of this chapter are probably best learned by dividing them into two groups. The first group contains three arithmetic instructions to be added to the ADD and SUBTRACT instructions of the last chapter.

INCREASE A(INA.b) (11.b) (b may be 0 through 7)

This instruction adds to A the sum of the contents of the designated index register and the execution address itself. The execution address is treated as a 15-bit quantity including sign (15 bits total). The addition is performed as if the execution address were a 48-bit quantity with the higher order bits copies of the sign bit. If overflows occur, the condition may be sensed by the external function instruction. The following sketch illustrates this instruction. See FIG. 8

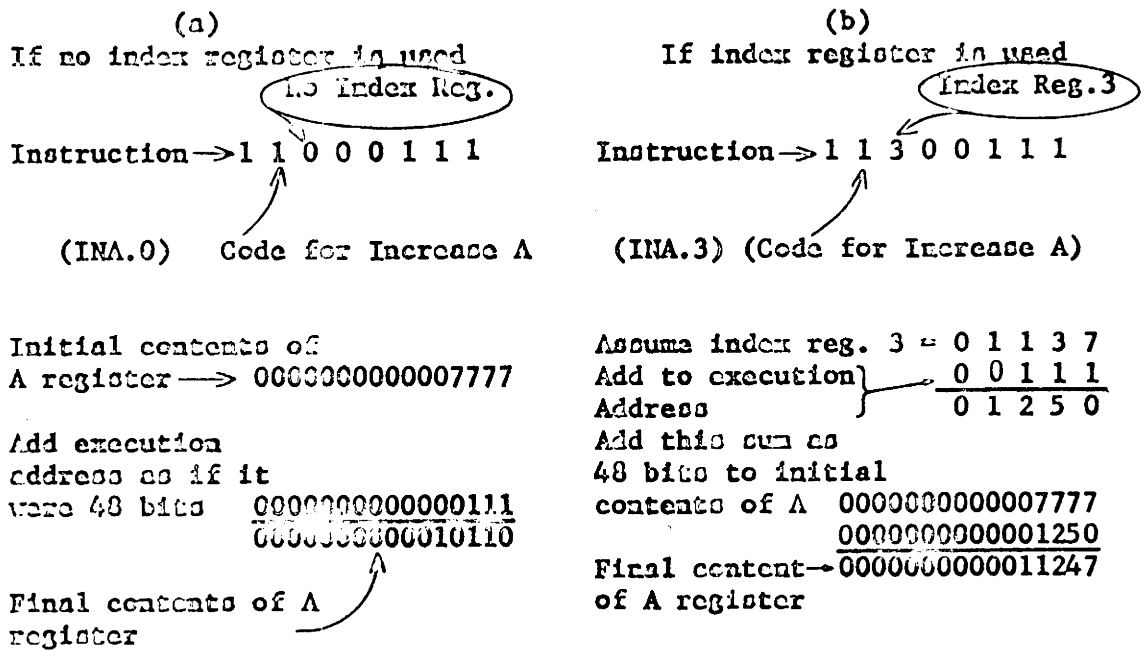

(a)
If no index register is used
No Index Reg.
Instruction→1 1 0 0 0 1 1 1
(INA.0) Code for Increase A
Initial contents of A register → 0000000000007777
Add execution address as if it were 48 bits 0000000000000111
0000000000010110
Final contents of A register
(b)
If index register is used
Index Reg.3
Instruction→1 1 3 0 0 1 1 1
(INA.3) (Code for Increase A)
Assume index reg. 3 = 0 1 1 3 7
Add to execution Address 0 0 1 1 1
0 1 2 5 0
Add this sum as 48 bits to initial contents of A 0000000000007777
0000000000001250
Final content of A register → 0000000000011247

FIG. 8

Execution of Increase A Instruction

MULTIPLY INTEGER (MUI.b) (24.b) (b may be 0 through 7)

This instruction forms a 96-bit product from two 48-bit operands. The multiplier must be loaded into the A register prior to the execution of this instruction. The sum of the contents of the designated index register and the execution address specifies the location of the multiplicand. The resulting product is contained in the QA register as a 96-bit quantity where least significant bits are in A. FIG. 9 indicates the execution of this instruction.

(a)
If no index register is used

Assume (A) = 0000000000000003
Assume (00100) = 0000000000000007
(MUI.0) (code for multiply)
Instruction $\longrightarrow$ 2 4 0 0 0 1 0 0
Multiplicand is found in 00100.
Product is 3 x 7 = 21_{10} = 25_8

Final contents of
Q Register A Register

$$\underbrace{0 \longleftrightarrow 0 \qquad 0 \longleftrightarrow 025}$$

$\boxed{\text{Sign of Product}}$

96-Bit Product

(b)
If index register is used

Assume (A) = $0 \longleftrightarrow 03$
Assume Index Register 5 = 00022
Assume (00100) = $0 \longleftrightarrow 07$
Assume (00122) = $0 \longleftrightarrow 012$
Instruction 2 4 5 0 0 1 0 0

Multiplicand is found in 00122
(since 00100 + 00022 = 00122.)
Product is 3 x 12 (octal) = 36 octal

Final contents of
O Register A Register

$$\underbrace{0 \longleftrightarrow 0 \qquad 0 \longleftrightarrow 036}$$

$\boxed{\text{Sign of Product}}$

96-Bit Product

FIG. 9

The Multiply Integer Instruction

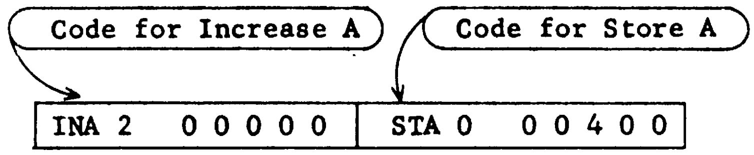
DIVIDE INTEGER (DVI.b)(25.b)(b may be 0 through 7)

This instruction divides a 96-bit integer dividend by a 48-bit integer divisor. The 96-bit dividend must be formed in the QA register, with the least significant bits in A, prior to the execution of this instruction. The 48-bit divisor is read from the storage location specified by the sum of the designated index register contents and the execution address. The quotient is formed in the A register. The remainder is left in the Q register at the end of the operation. The remainder and the initial dividend will have the same algebraic sign.

The following examples are provided to afford better understanding of the three instructions previously defined.

EXAMPLE 16. Each time a program reaches step 00050, the A register is to be increased by the amount in index register 2, and the sum is to be stored in storage address 00400. Show this program step.

(00050) =



Explanation: The first instruction increases the contents of the A register by the sum of the contents of the designated index register (2 in this case) and the amount specified in the execution address. Since the amount in execution address is zero, the A register is increased by the contents of index register 2. The second instruction stores A in storage address 00400.

EXAMPLE 17.

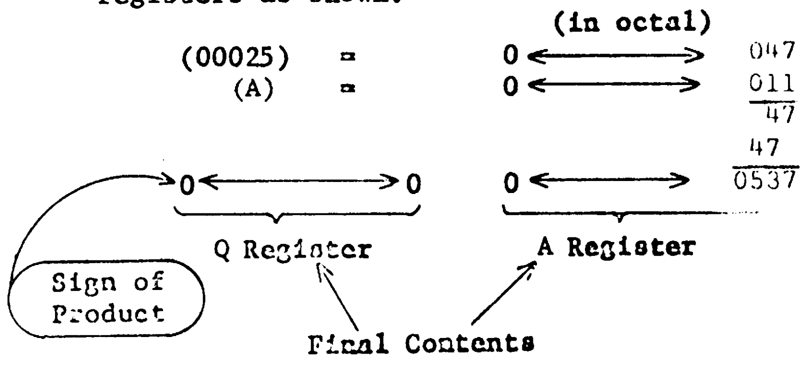
Assume (A) = 0000000000000011
Assume (00025) = 0000000000000047

After the following instruction (MULTIPLY) is executed, show the final contents of A, Q, and storage address 00025.

(MUL.0)

Instruction 2 4 0 0 0 0 2 5

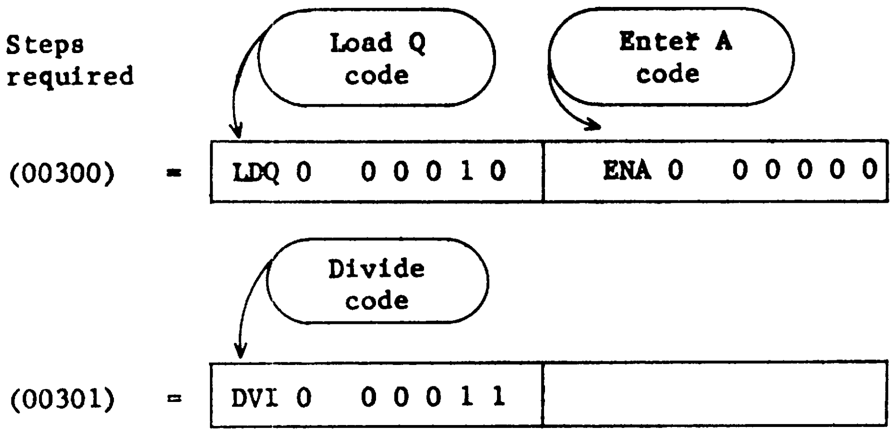
Contents of 00025 (multiplicand) is multiplied by contents of A (multiplier) and the 96-bit product is developed in the Q and A registers as shown.



Storage address (00025) will still contain 0 ⟷ 047 after completion of the multiply.

Explanation: (See description above)

<u>EXAMPLE 18.</u> One wishes to divide 2^{49} by 15. Show the steps required. Assume high order bits of 2^{49} are in storage address 00010 and 15 is in storage address 00011.



(00300) = | LDQ 0 0 0 0 1 0 | ENA 0 0 0 0 0 0 |

(00301) = | DVI 0 0 0 0 1 1 | |

<u>Explanation:</u> $2^{49} = \underbrace{10 \longleftrightarrow 0}_{\text{49 zero bits}}$

or $2^{49} = \underbrace{0000000000000002}_{\substack{\text{Assume this is}\\ \text{stored in 00010,}\\ \text{will go to Q}}} \quad \underbrace{0000000000000000}_{\substack{\text{This will go to A Reg.}\\ \text{before DIVIDE}}}$ (octal)

The first instruction loads 0000000000000002 into Q.
The second instruction loads zeros into A.
The third instruction divides by 15 in address 00011.

After the divide the quotient will be in A and the remainder will be in Q.

The following exercises are provided for practice on the three instructions just described. Solve the given problems and check your answers with the solutions presented in Appendix A. The number of correct answers will indicate your approximate progress by checking the Rating Table below:

Rating Table 5

If you have 4 or 5 correct answers	Excellent
If you have 3 correct answers.	Good
If you have 1 or 2 correct answers	Fair (Review)
If you have 0 correct answers.	Complete Review

EXERCISES 5.

a. Assume index register 6 contains 00017 and the A register contains 77777777777770135. The following "INCREASE A" instruction is given (11 6 00761). Show the contents of the A register after execution of this instruction.

b. One wants to multiply the contents of storage location 00050 (the multiplicand) by the contents of index register 2 (the multiplier). Show the program step required to do this.

c. Mark the following as True or False:

1. The multiply does not affect the magnitude of the multiplier in A. In other words, the multiplier has the same value after the multiply has been executed. ____________

2. Before the multiply instruction is referenced, it is necessary to LOAD the multiplier into the A register with a previous instruction. ____________

3. The product of two 48-bit quantities may "overflow" the 96-bit product possible in the QA register. ____________

d. Assume the Q register contains 0000000000000000. The A register contains 0000000000000020. Assume storage location 00100 contains 0000000000000003. The DIVIDE instruction 25 0 00100 is given. Show the final contents of the A register, the Q register, and storage location 00100.

Resultat 5 rest 1.
celle 100 uforandret.

e. Show program step which will "INCREASE A" by the contents of index registers 2 and the contents of the storage location whose address is given by the contents of index register 5.

The last three instructions to be described in this chapter include one instruction which is a SELECTIVE JUMP and can be used as the basis for an "unconditional jump". The other two are useful in modifying or substituting new execution addresses in the lower or upper instructions of a program step.

SELECTIVE JUMP (SLJ.b)(75.b)(where b may be 0 through 7)

This instruction causes a jump determined by specific settings of operator keys on the console of the computer. These keys are "selected" before the program starts and determine whether or not the jump will occur when the program reaches the location of the SELECTIVE JUMP instruction. The index register designator specifies which jump key is set. Thus, on this instruction, the index registers are not used. The following index designations indicate the following conditions:

<u>Index Designation</u>	<u>Condition</u>
0	[illegible] set Jump Unconditionally (no key setting)
1	Jump if jump key 1 is set
2	Jump if jump key 2 is set
3	Jump if jump key 3 is set
4	Return Jump unconditionally (no key setting)
5	Return Jump if jump key 1 is set
6	Return Jump if jump key 2 is set
7	Return Jump if jump key 3 is set

SUBSTITUTE ADDRESS (UPPER) (SAU.b)(60.b)(where b may be 0 through 7)

This instruction replaces the execution address portion of the left or upper instruction of a word, whose location is specified by the sum of the execution address and the contents of the designated index register, by the lowest order, 15 bits of the A register contents. The remaining bits of the designated word in storage are not modified by this operation. This instruction effectively inserts an address in the first instruction at the designated storage location. FIG. 10 indicates the process.

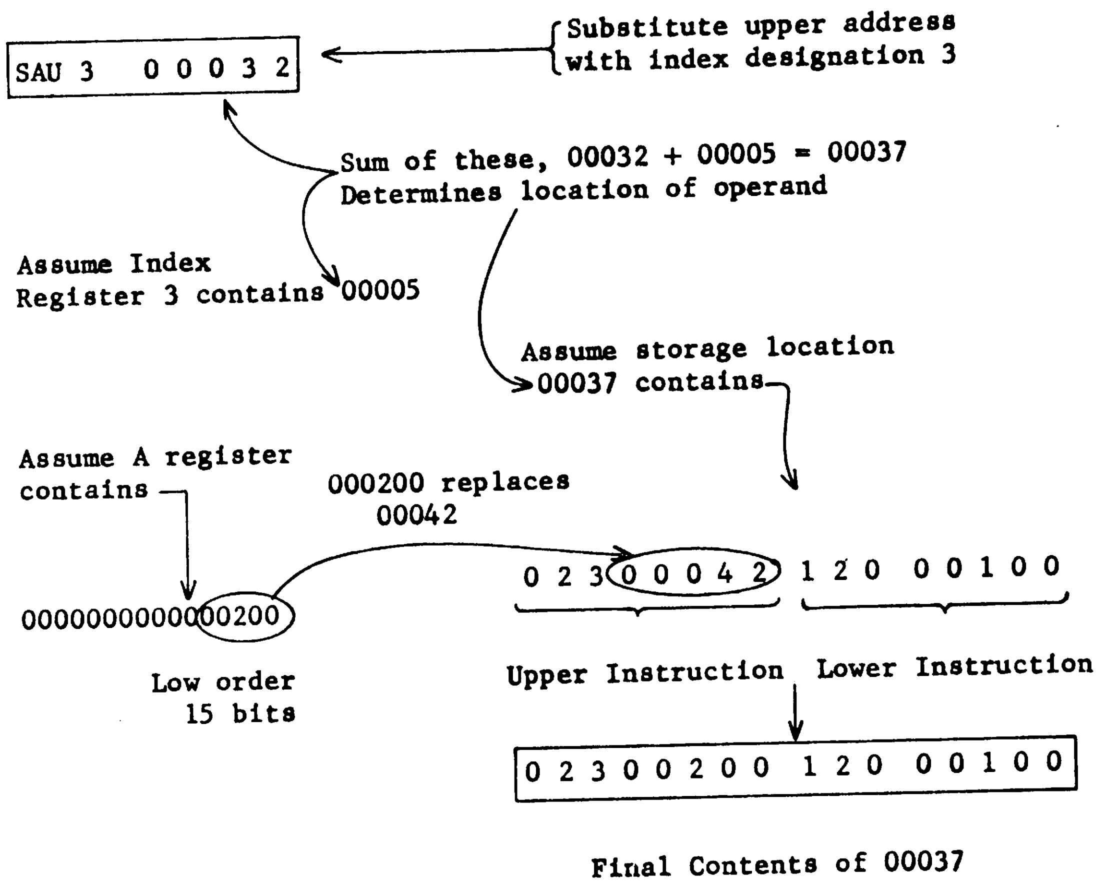

SAU 3 0 0 0 3 2
Substitute upper address with index designation 3
Sum of these, 00032 + 00005 = 00037
Determines location of operand
Assume Index Register 3 contains 00005
Assume storage location 00037 contains
Assume A register contains
000200 replaces 00042
0 2 3 0 0 0 4 2 1 2 0 0 0 1 0 0
000000000000200
Low order 15 bits
Upper Instruction
Lower Instruction
0 2 3 0 0 2 0 0 1 2 0 0 0 1 0 0
Final Contents of 00037

FIG. 10

The Substitute Upper Instruction

SUBSTITUTE ADDRESS (LOWER) (SAL.b) (61.b) (where b may be 0 through 7)

This instruction performs the same as SUBSTITUTE ADDRESS (UPPER) see FIG. 10 , except the instruction inserts an address in the lower instruction of the word whose location is specified by the sum of the execution address and the contents of the designated index register.

The following examples should facilitate learning these three last instructions of this chapter.(Each student is advised to work out the given examples and check the given solutions.)

EXAMPLE 19. When a certain program reaches step 00046 the first time, the program jumps to location 00200 where an output routine using magnetic tape is processed. The second time the program is run and location 00046 is reached, the program wants to go on in sequence with 00047 which is an output routine on punched cards. Show the instruction at 00046 and explain how it would take care of these conditions.

Required
Instruction (00046) =

SLJ 1 0 0 2 0 0	Some other instruction

Explanation: The instruction is the SELECTIVE JUMP with index designation 1. The first time, Jump Key 1 on the console would be set and the program would jump to the output routine using magnetic tape. If the second time, Jump Key 1 was not set on the console, and the program would continue with the step at 00047.

EXAMPLE 20. At location 00600, on the left side, is a "dummy" ENTER Q instruction. This dummy instruction is ENQ 0 xxxxx. Before the computer reaches this location, the program will enter an amount for the xxxxx of this instruction. Assume the amount to be entered is being generated in storage location 00300. At step 00577, the substitution is to take place. Show program step 00577.

Required
Program Step (00577) =

LDA 0 0 0 3 0 0	SAU 0 0 0 6 0 0

Explanation: The first instruction above LOADS A with the contents of register 00300. Thus A now contains 0000000000ssss (where sssss is some quantity). The second instruction tells the machine to SUBSTITUTE INTO THE LEFT INSTRUCTION OF 00600 the 15 low order bits of A. Thus sssss goes into the positions held by xxxxx and the left instruction at 00600 becomes

(00600) =

ENQ 0 s s s s s	Some Other Instruction

EXAMPLE 21. Storage location 00040 contains two generated instructions. Interchange the execution address portions of each of these instructions. Show program steps starting with 00100 which will do this.

(00100)	LDA 0 0 0 0 4 0	SAU 0 0 0 0 4 0
(00101)	ARS 0 0 0 0 3 0	SAL 0 0 0 0 4 0

Explanation: LOAD contents of 00040 into A. Substitute low order 15 bits in A (execution address of lower instruction) into upper instruction execution address. Then shift original content of 00040 (which is still in A) right 24_{10} bit-positions. Then substitute 15 low order bits of A (now the execution address of original left instruction) into right instruction execution address. This completes the interchange.

The following exercises are provided to afford practice on the last three instructions of this chapter. Work out the solutions and check results with those in appendix A. The following table indicates your approximate progress in learning to program these three instructions.

<u>Rating Table 6</u>

If you have 4 or 5 correct answers	Excellent
If you have 3 correct answers.	Good
If you have less than 3 correct answers. . . .	Review

EXERCISES 6.

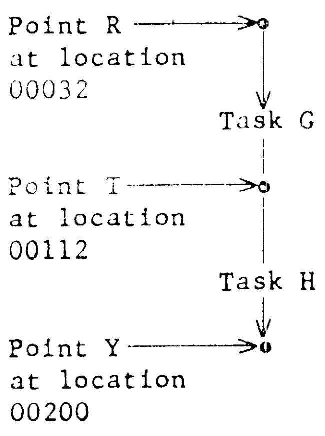
a. At location 00020, it is desired to jump to a subroutine starting at step 00072. This subroutine exits at 00105 back to the main program, continuing at 00021. Show the Return Jump at 00020, the first instruction at 00072 and the exit instruction at 00105.

b. Assume 00022 contains 1 2 3 0 0 1 0 0 0 6 4 0 0 2 0 0.
Assume 00023 contains 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1.
Show the steps, starting at 00112 which will modify (00022) to 1 2 3 0 0 0 0 1 0 6 4 0 1 0 0 0 using only the given locations above along with A and Q.

c. Assume index register 5 contains a generated number. If this number is positive, store it in the last 5 octal digit positions of location 00047 and then jump to step 00700 if jump Key 2 is set. If it is negative add to it the contents of address 00100 and jump to step 00700. (Start steps at location 00417.)

d. Increase the contents of address 00003 by the contents of index register 3 and if Jump Key 3 is set on the console, jump to program step 30. Write the program steps to do this starting at location 00050.

e. Study the following sketch.



Steps between points R and T perform TASK G. Steps between points T and Y perform TASK H. A programmer wishes to be able to do the following at different times.

1. Do TASK G and TASK H
2. Skip TASK G, Do TASK H
3. Do TASK G, Skip TASK H
4. Skip both TASKS

Write two instructions, one at Point R and one at Point T which will enable all four of these to be done at different times in a program.

CHAPTER III

REVIEW TEST

The following questions review the twelve instructions presented in this chapter. Write answers in the space provided after each question. Solutions are given in Appendix B.

1. The AQ register is really two separate registers. A and Q are treated as one register on certain instructions (True-~~False~~)

2. The fact that shifts greater than 127_{10} are treated as shift faults, stops one from shifting the AQ register the equivalent of 200_{10} positions. (True-~~False~~)

3. The "ENTER A" and "ENTER Q" instructions always lead to 15 bits being entered into A or Q and the generation of 33 other sign bits which correspond to the left most bit of the 15 bits which are entered. (True-False)

4. Assume index register 3 contains 77777. If "ENTER A" instruction 1 0 3 0 0 0 0 1 is given, what quantity will be entered into the A register?

5. Assume location 00100 contains 0000000000000003. Show the contents of the Q register after the LOAD Q COMP instruction (LQC. 0) 777- - - . . 4
   1 7 0 0 0 1 0 0

6. One wants to increase the contents of the A register by the amount contained in index register 4. Show an instruction which will do this.

7. Is there any difference in the final result between "INCREASE A" and "ADD" instructions if the execution address portions are each zero and the same index designation is used in each. In other words, is INA 2 0 0 0 0 0 the same as ADD 2 0 0 0 0 0? (Yes or No)

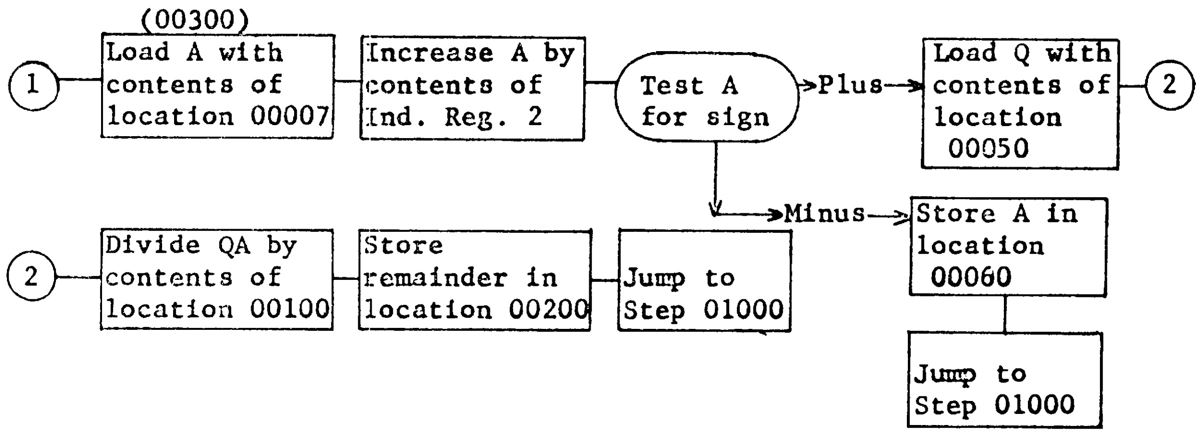
8. Where is the multiplier located when the "Integer Multiply" instruction is executed? Where is the multiplicand. Where is the product after the execution of the instruction?

(Multiplier) ____________________
(Multiplicand) ____________________
(Product) ____________________

9. If the dividend is 5 and the divisor is -3, what will the following registers contain after the DIVIDE INTEGER instruction is executed?

A Register ____________________
Q Register ____________________

10. Show program steps which will accomplish the following flow diagram. (Start with stop 00300.)


(00300)
1
Load A with contents of location 00007
Increase A by contents of Ind. Reg. 2
Test A for sign
Plus
Load Q with contents of location 00050
2
Minus
Store A in location 00060
Jump to Step 01000
2
Divide QA by contents of location 00100
Store remainder in location 00200
Jump to Step 01000

11. Write program steps starting with step 00200 which will divide the sum of the A and Q register contents by the difference obtained by subtracting the contents of Q from the contents of A.

12. An approximation to the function G can be found by the formula:

$$G = 1 - \frac{X^2}{2}$$ Assume X is a whole number

stored in index register 3. Write program steps starting with step 00100 which will compute G and store it in storage location 00077.

CHAPTER IV

THIRD GROUP OF INSTRUCTIONS

THIRD INSTRUCTIONS (with Mnemonic and Numeric Codes)(where b refers to index register designation).

STORAGE SKIP	(SSK.b)(36.b)	ENTER INDEX	(ENI.b)(50.b)
STORAGE SHIFT	(SSH.b)(37.b)	PASS	(ENI.0)(50.0)
INDEX SKIP	(ISK.b)(54.b)	INCREASE INDEX	(INI.b)(51.b)
INDEX JUMP	(IJP.b)(55.b)	LOAD INDEX UPPER	(LIU.b)(52.b)
LOAD LOGICAL	(LDL.b)(44.b)	LOAD INDEX LOWER	(LIL.b)(53.b)
ADD LOGICAL	(ADL.b)(45.b)	STORE INDEX UPPER	(SIU.b)(56.b)
SUBTRACT LOGICAL	(SBL.b)(46.b)	STORE INDEX LOWER	(SIL.b)(57.b)
STORE LOGICAL	(STL.b)(47.b)		

The third group of instructions are fifteen in number. These are probably best learned by dividing them into groups containing four, four, and seven instructions, respectively.

The first three instructions are similar in that they may be thought of as "two address" jumps. These are limited to use in the first position of a program step. The second position in the program step is normally occupied by an unconditional jump instruction. This pair of instructions then operates as a two address jump instruction. The first address of the storage skip and storage shift instructions in this program step specifies the storage location of the operand required in the jump decision. The second address specifies the destination of the jump if taken. Other combinations will be obvious to the programmer as he learns the instructions.

STORAGE SKIP (SSK.b)(36.b) (where b may be 0 through 7)

This instruction senses the sign bit of the quantity in the location designated by the sum of the execution address and the contents of the designated index register. If the designated quantity is negative, the next instruction is skipped. If the designated quantity is positive, the next instruction is executed. FIG. 11 indicates the process.

(a)
No Index Register is
Designated

Assume 00100 contains 0——— 01157
Assume following program step

(00040) 36 0 00100 75 0 00062
Operand at 00100 (0+00100=00100)
is positive.
The next instruction (75 0 00062)
is executed.
This is an unconditional jump to
step 00062.

(b)
If Index Register 2 is
Designated

Let index reg. 2 contain 00032
Let storage location 00037 contain
7——— 73[illegible]2
Assume following program step

(00040) 36 2 00005 75 0 00062
Operand at 00037 (00005+00032=00037)
is negative. The next instruction is
skipped and program continues at (00041).

FIG. 11 One use of Two-Address Jump

It is important to note that the second position need not be an unconditional jump. Thus if a single instruction is to be executed, or not executed, as a result of the sensing operation, then such an instruction may be used in the second position with the sensing instruction in the first position. Examples following these descriptions indicate these possibilities.

STORAGE SHIFT (SSH.b)(37.b) (b may be 0 through 7)

This instruction senses the sign bit of the quantity in the storage location specified by the sum of the execution address and the contents of the designated index register. If the designated quantity is negative, the next instruction is skipped. If the designated quantity is positive, the next instruction is executed. In either case the quantity in storage is shifted circularly, end-around, left one bit position. The initial contents of A and Q are not disturbed by this shifting process.

INDEX SKIP (ISK.b)(54.b) (where b may be 0 through 7)

This instruction compares the quantity in the designated index register with the base execution address. If the two are equal, then the designated index register is cleared and the next instruction in the program is skipped. If the quantity in the designated index register is not equal to the base execution address, then the quantity in the index register is increased one count and the next instruction in the program is executed. The flow diagram in FIG. 12 indicates the process. (Note the special situation when the designated index register is zero.)

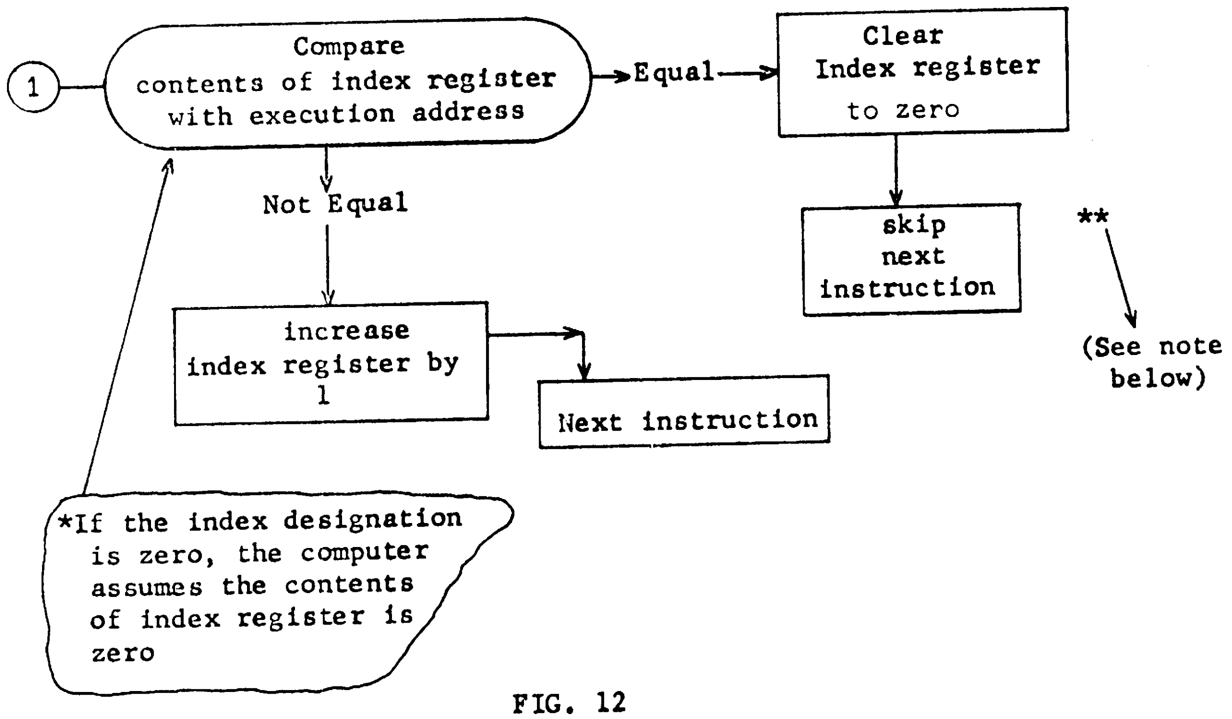

1
Compare contents of index register with execution address
Equal
Clear Index register to zero
Not Equal
skip next instruction
**
(See note below)
increase index register by 1
Next instruction
*If the index designation is zero, the computer assumes the contents of index register is zero

FIG. 12

Index Skip Instruction

**It is important to remember that the two-way sensing instructions occupy the first position (upper instruction) in the program step. In case they are used in the second position (Lower instruction) the skip will not take place which provides the two-way possibilities.

The fourth and last instruction in this group is the INDEX JUMP. It is similar to the INDEX SKIP instruction just described.

INDEX JUMP (IJP.b)(55.b) (where b may be 0 through 7)

This instruction examines the quantity in the designated index register. If this quantity is not zero, it is reduced by one count and a jump is executed to the base execution address. If this quantity is zero, then the jump is not executed and the present program sequence is continued. (Note in this instruction an index designation of zero would have little use unless it was changed during the program. A zero designation would simply have the same effect as if the contents of the index register were zero.)

The following examples should facilitate learning the previous four instructions. (Each student is advised to work out the given examples and to check the given solutions.)

EXAMPLE 22. A quantity is generated in storage location 00023. If this quantity is positive, jump to program step 00077. If negative, the program is to continue in sequence. Show a program step which will do this. (Use location 00100 for program step.)

(00100)	=	SSK 0 0 0 0 2 3	SLJ 0 0 0 0 7 7

(00101) = Continue Program

Explanation: The first instruction checks the sign of the operand at the location specified by the sum of the designated index register (0) and the execution address (00023). If this operand (at 00023) is positive, the next instruction (750 00077) is executed, which is an unconditional jump to step 00077. If the operand (at 00023) is negative, the jump instruction (750 00077) is skipped and the program continues in sequence at step 00101.

EXAMPLE 23. If the quantity contained at storage location 00200 is positive, it is desired to add it to the contents of the A register and continue in sequence. If it is negative, it is desired to continue in sequence to step 00031. Write program step 00030 which will do this

(00030)	=	SSK 0 0 0 2 0 0	ADD 0 0 0 2 0 0

(00031) = Continue Program

Explanation: The first instruction does a "STORAGE SKIP" on contents of storage location 00200 (specified by sum of execution address and contents of designated index register). If contents of 00200 are positive, the next instruction (140 00200) will ADD the contents of 00200 to the A register and continue to (00031). If the contents of 00200 are negative, the ADD instruction is skipped and the program continues. (Note: This is an example showing the use of a "possible two-address jump" without using an unconditional jump following the sensing instruction.)

EXAMPLE 24. Assume storage location 01000 contains 25 25 25 25 25 25 25 25 (octal). At step 00300 in a program it is desired to use the "STORAGE SHIFT" instruction with index designation zero and execution address 01000. (See below)

(00300)	=	SSH 0 0 1 0 0 0	Jump to 0 0 6 0 0
(00301)	=	Instruction C	Jump to 0 0 4 0 0

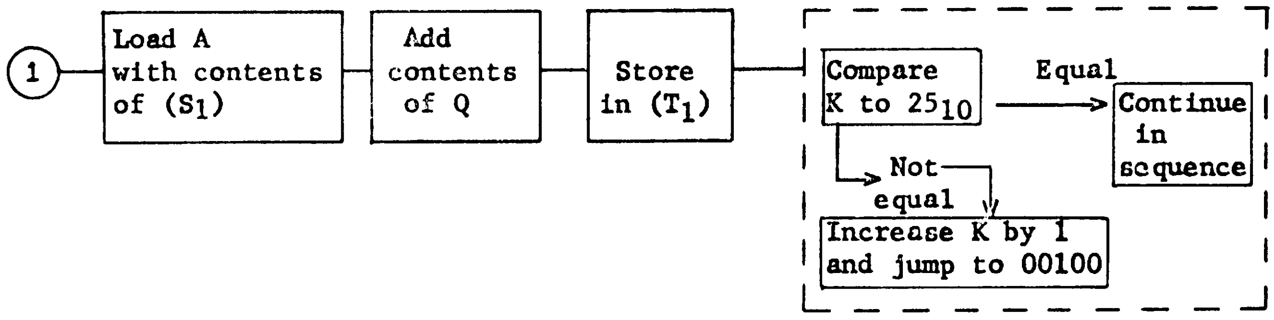
The program reference step 00300 several times. Explain what will happen in terms of the jumps shown.

Answer and Explanation:

Each time the Storage Shift instruction is executed, the contents of 01000 will shift left one bit position. Since 01000 contains alternate zeros and ones (in bits), the following will take place:

First time:	The jump to 00600 will occur
Second time:	Instruction C and jump to 00400 will occur
(Succeeding times:	Repeat the above in same order)

EXAMPLE 25. Assume index register 6 contains all zeros (00000 in octal). A programmer wants to use this index register as the counter (K) indicated below in the flow chart. Show the program step for dotted area below



Program
Step Required →

ISK 6 0 0 0 3 1	SLJ 0 0 0 1 0 0

Continue steps in sequence

Explanation: The first instruction is the "INDEX SKIP". This compares the contents of the designated index register (6) with the execution address amount (in this case $25_{10} = 31_8$). If not equal, the contents of index register 6 are increased by 1 and the next instruction, 7 5 0 0 0 1 0 0, (Unconditional jump to 00100) takes place. Each time the program loops through here the same sequence of events takes place until the contents of index register 6 equals 25_{10} and then the jump instruction is skipped and the program continues in sequence. (See Example 47 for another instance of index skip.)

EXAMPLE 26. Assume index register 3 contains 00007. A program contains a loop which involves a jump to location 00112 seven times during the program. The eighth time the program is to continue in sequence. Show a program step at (00061) which will provide the looping. (use INDEX JUMP instruction.)

(00061) =

IJP 3 0 0 1 1 2	Continue Program

Explanation: Only one instruction is required. The INDEX JUMP above checks the contents of index register 3 for zero. If not zero, it reduces the contents by one and jumps to the step specified in the execution address portion (00112 in this case). This continues 7 times until index register contents are reduced to zero. At this time (eighth time) the jump is not taken but the next instruction is executed.

The following exercises are provided for practice on the first four instructions of this chapter. Work out the given problems and check your solutions with those given in Appendix A. The number of correct answers will indicate your approximate progress by checking the Rating Table below:

Rating Table 7

If you have 4 correct answers	Excellent
If you have 3 correct answers.	Good
If you have 2 correct answers	Fair
If you have 0-1 correct answers	Complete Review

EXERCISE 7

a. When program step 00032 is reached, the contents of storage location 00050 are to be tested for sign. If positive, the jump is to go to step 00110. If negative, a jump is to be made to 00200. Using the STORAGE SKIP instruction, show program step 00032.

b. Explain what happens if a conditional jump with jump key 2 not set, follows the STORAGE SKIP instruction as shown below:

	Storage Skip	Conditional Jump
(00052) =	SSK 0 0 0 1 0 0	SLJ 2 0 0 2 0 0

c. In storage location 00040 there is an octal digit (0 through 7). The programmer wants to determine if the digit is odd or even by use of the STORAGE SHIFT instruction. Write program steps that would do this. (Hint: First shift the right-most bit of the octal digit to the left-most bit position and then use the STORAGE SHIFT instruction.)

d. In a certain program, if the contents of index register 5 equal the number 100_{10}, the program jumps to step 00200. If the contents of index register 5 do not equal 100_{10}, the program jumps to step 00050. Write instructions which will do this, using the INDEX SKIP instruction.

The second four instructions of this chapter include those instructions known as "logical instructions". These are the instructions generally used in programming situations that extract or mask portions of words.

LOAD LOGICAL (LDL.b)(44.b)(where b may be 0 through 7)

This instruction loads the A register with the bit-by-bit logical product of the Q register contents and the quantity in storage located by the sum of the execution address and the contents of the designated index register. This logical product utilizes the following rules:

1 X 0 = 0
0 X 1 = 0
0 X 0 = 0
1 X 1 = 1

ADD LOGICAL (ADL.b)(45.b)(where b may be 0 through 7)

This instruction adds to the contents of the A register, the logical product of the Q register contents and the quantity in storage whose location is given by the sum of the execution address and the contents of the designated index register. This a normal add for the selected bits of the operand, the non-selected bits of the operand being interpreted as zeros.

SUBTRACT LOGICAL (SBL.b)(46.b)(where b may be 0 through 7)

This instruction subtracts from the contents of the A register the logical product of the Q register contents and the quantity in storage whose location is specified by the sum of the execution address and the contents of the designated index register. This is a normal subtraction for the selected bits of the operand, the non-selected bits of the operand interpreted as zeros.

STORE LOGICAL (STL.b)(47.b)(where b may be 0 through 7)

This instruction stores the logical product of the A register and Q register contents at the storage location which specified by the sum of the execution address and the contents of the designated index register. Neither the A nor the Q register initial contents are modified by this instruction.

It is important to note that in the first three instructions, involving the logical products, the "extractor" can be placed in either the Q register or in the operand at the specified storage location. In the STORE LOGICAL instruction, the "extractor" may be either in the Q register or the A register.

The following examples should facilitate learning the previous four instructions. (Each student is advised to work out the given examples and to check the given solutions.)

EXAMPLE 27. It is desired to examine the right-most nine bits of the contents of storage location 00017. The extractor is placed into Q and the LOAD LOGICAL instruction is executed. Show the instruction, the contents of Q, and the contents of the A register after the execution of the instruction.

INSTRUCTION REQUIRED LDL 0 0 0 0 1 7 ⟵ This loads A with logical product of Q and contents of 00017

Extractor in Q ⟶ 0 0 0 0 0 0 0 0 0 0 0 0 0 7 7 7

The nine "one Bits" here will extract the last nine bits of contents of 00017

Right nine bits of 00017

Final contents of A ⟶ 0 0 0 0 0 0 0 0 0 0 0 0 0 X X X

Explanation: The logical product of contents of Q and contents of 00017 will give the right nine bits of the contents of 00017. These nine bits (which make 3 octal digits) are loaded into the A register.

EXAMPLE 28. Storage location 00100 contains eight six-bit codes. Each code designates the number of times the Q register contents are to be shifted. The A register contains the "dummy" SHIFT Q LEFT instruction as the lower, and an unconditional jump to 00200 as the upper instruction. (7 5 0 0 0 2 0 0 0 6 0 0 0 0 0 0). It is desired to pick off the right six-bit code in address 00100, add this code to the dummy shift in A, and send the result to some other step in sequence. Show the program steps to do this, starting with step 00010. (Hint: save the initial contents of Q while using Q to extract six bits from 00100.)

```
A         = | SLJ  0  0 0 2 0 0      | ALS  0  0 0 0 0 0 |

              Store Q in                 Load Q with
               00050                      Extractor
(00010)   = | STQ  0  0 0 0 5 0      | LDQ  0  0 0 0 6 2 |
                  Add Logical Product     Shift A Left 24₁₀ bit positions
                  to Dummy in A           to put Dummy in  left part.

(00011)   = | ADL  0  0 0 1 0 0      | ALS  0  0 0 0 3 0 |
                  Store Dummy             Reload Q with saved
                  at Step 00013           initial contents
(00012)   = | STA  0  0 0 0 1 3      | LDQ  0  0 0 0 5 0 |

                                          Unconditional Jump

(00013)   = | ALS  0  0 0 0 X X      | SLJ  0  0 0 2 0 0 |
```

Explanation: Q is first saved. Then the extractor is loaded into Q. (Extractor here is in 00062 and should equal 0000000000000077). The third instruction adds the right six code-bits of (00100) to the dummy shift instruction in A. This is then shifted left 24_{10} positions to put the instruction on the upper portion of the word. The word is then sent to step 00013. The Q register is re-loaded with its initial contents.

EXAMPLE 29. Assume storage location 00055 contains $0000000000000X_1X_2X_3$ (where X_1, X_2, X_3 are generated octal digits). It is desired to store $0000000000000X_1X_20$ in storage location 00056. Show the program steps, starting at 00477 which will do this. (Assume extractor is in 00100 and show this extractor.)

	Load Q with Extractor	Load A with contents of 00055 ($X_1X_2X_3$)
(00477) =	LDQ 0 0 0 1 0 0	LDA 0 0 0 0 5 5

	Subtract Logical Product from $X_1X_2X_3$	Store in 00056
(00500) =	SBL 0 0 0 0 5 5	STA 0 0 0 0 566

where extractor in storage location 00100 is 0000000000000007

Explanation: The extractor, which will pick off X_3, is loaded into the Q register. The quantity X_1, X_2, X_3 is then loaded into the A register. At step 00500, the SUBTRACT LOGICAL instruction subtracts from A ($X_1X_2X_3$) the logical product (X_1) which results from the logical multiply of the contents of 00055 by the extractor in Q. This leaves X_1X_2 0 in A and this is stored in storage location 00056.

EXAMPLE 30. Assume A contains 16 octal digits which are unknown to the programmer. It is desired to pick off the left-most octal digit and store it at location 00312. Show how this can be done with the STORE LOGICAL instruction. (Hint: first load Q with the correct extractor - show this extractor.)

	Load Q with Extractor	Store Logical Product of A and Q at 00312
(c)	LDQ 0 0 0 0 1 4	STL 0 0 0 3 1 2

where extractor at 00014 is 7000000000000000

Explanation: Since the left-most digit is desired, the extractor contains three one bits (octal 7) in the left-most octal position. This extractor is first placed in Q. The STORE LOGICAL instruction then stores the logical product of the 16 octal digits in A and the extractor in Q at location 00312. At the end of the operation, 00312 will contain X000000000000000, where X is the left-most octal digit. If this is desired to be in the far right position, the appropriate shift must be executed. What shift would be needed?

The following exercises are provided for practice on the previous four instruction of this chapter. Work out the given problems and check your solutions with those presented in Appendix A. The number of correct answers will indicate your approximate progress by checking the Rating Table below:

Rating Table 8

If you have 4 correct answers	Excellent
If you have 3 correct answers	Good
If you have 2 correct answers	Fair
If you have 0 or 1 correct answers .	Complete Review

EXERCISES 8

a. Storage location 01012 contains eight 6-bit characters as shown:

```
(01012)    K  K  K  K  K  K  K  K
           1  2  3  4  5  6  7  8
```

The left most bit of each character contains parity data. If there is a 1 in any one of these eight positions, the whole word is to be cleared. Write program steps which will make the test and clear out the word if necessary. Show any extractors used.

b. At program step 00111 the following ADD LOGICAL and STORE A instructions are given as shown.

```
(00111) = | 4 5 3  0 0 0 0 0 | 2 0 0  0 0 1 0 0 |
```

Assuming the A register contains 1200000000000000, the Q register contains 0070000000000000, and index register 3 contains 00111, show the final contents of the A register after completing step 00111.

c. Write program steps which will reduce the quantity in storage location 00712 by thesum of the left-most and right-most octal digits of that quantity.

d. Answer the following questions:

1. Can a logical product equal the extractor?

2. Can a logical product exceed in magnitude both of the factors? Explain.

3. If the content of either A or Q is positive, is it possible for the LOGICAL STORE instruction to store a negative quantity? Explain.

The last seven instructions in this chapter are held together by a common bond. Each deals with an index register in some relationship. For this reason, these instructions can probably be learned as a group.

ENTER INDEX (ENI.b)(50.b) (where b may be 1 through 7)

> This instruction replaces the contents of the designated index register with the value of the base execution address. No storage reference is made in this instruction. Thus if index register 3 contains 00007 and instruction "5 0 3 0 1 1 1 1" is given, then 01111 will replace 00007 in index register 3. (Note: in this instruction "b" cannot be zero - it has a special meaning when used in this particular instruction - see below.)

PASS (Do Nothing) (ENI.0) (50.0)

> This is a special code of the last instruction. When 5 0 0 is used as the first 3 octal digits in an instruction the machine knows that this portion of the step is not used - that is, it is to be passed by. The program thus proceeds to the next instruction. This PASS can be used in either half of a program step.

INCREASE INDEX(INI.b)(51.b)(where b may be 0 through 7)

> This instruction adds the value of the base execution address to the contents of the designated index register. (A zero index designation has little meaning here but if used the contents of the index register are considered to be zero by the computer.)

LOAD INDEX(UPPER)(LIU.b) (52.b)(where b may be 0 through 7)

> This instruction replaces the contents of the designated index register with the upper address of the word in the storage whose location is given by the base execution address. This instruction thus extracts the address from the upper instruction of a word at a designated storage location and enters this value into a particular index register. A zero index register designation has no meaning here and should not be used. If used, the machine uses up some time in referencing storage and then continues with the next instruction. This is not a fault.

LOAD INDEX (lower) (LIL.b) (53.b) (b may be 1 through 7)

This is similar to the previous instruction except the contents of the designated index register is replaced by the lower address of the word in the storage whose location is given by the base execution address. A zero index register designation has no meaning and should not be used. If used, it will skip to the next instruction but time is wasted by the machine making storage references involved. No fault will occur.

STORE INDEX (upper) (SIU.b) (56.b) (b may be 0 through 7)

This instruction stores the contents of the designated index register in the address portion of the upper instruction of the word whose location is given by the base execution address. The other bits at the specified storage location are not modified. A zero index register designation has no meaning, but if used, zeros are inserted in the address portion of the upper instruction of the word whose location is specified by the base execution address and the other bits are not modified by the insertion.

STORE INDEX (lower) (SIL.b) (57.b) (where b may be 0 through 7)

This instruction is similar to the previous instruction except the contents of the designated index register are stored in the address portion of the lower instruction of the word whose location is specified by the base execution address. A zero index register designation has little meaning, but if used, zeros are inserted into the address portion of the lower instruction of the word whose location is specified by the base execution address. Regardless, the other bits at the specified storage location are not modified by the insertion.

The following examples should facilitate learning the previous seven instructions (six and the PASS code). Each student is advised to work out the given examples and to check the given solutions.

EXAMPLE 31. At step 00021 the programmer wants to clear index registers 1 and 2. Show the program step with the necessary instructions.

(00021) =	ENI 1 0 0 0 0 0	ENI 2 0 0 0 0 0

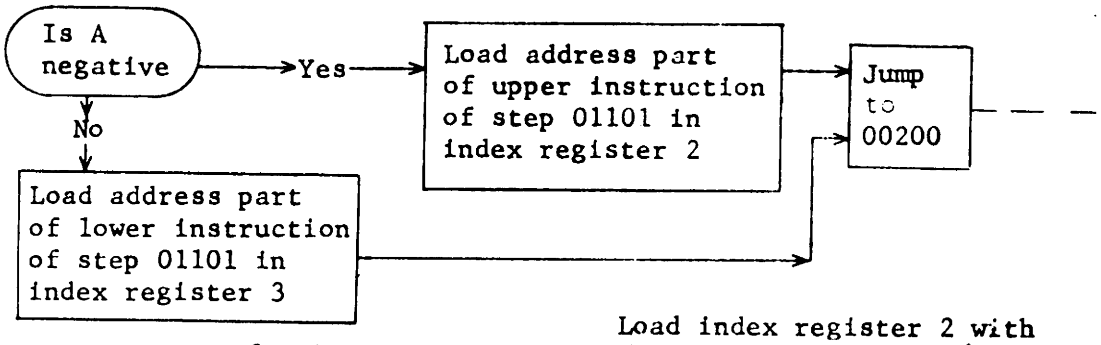
Explanation: Each instruction is the ENTER INDEX. The value in each execution address portion of each instruction is entered into the designated index register. Since the value is zero in each case, index registers 1 and 2 will contain zeros after the above step is performed.

EXAMPLE 32. At program step 00172, it is desired to increase the quantity in index register 5 by 10_{10} and decrease the quantity in index register 2 by 10_{10}. Show program step 00172.

(00172) =	INI 5 0 0 0 1 2	INI 2 7 7 7 6 5

Explanation: Two INCREASE INDEX instructions are required. The first instruction adds 10_{10} (which is 00012 in octal) to the contents of index register 5. The second instruction adds negative 10_{10} (which is 77765 in octal) to the contents of index register 2. The addition of negative 10_{10} is the equivalent of subtracting a positive 10_{10}.

EXAMPLE 33. Write program steps for the following flow diagram (Start at step 00100)


Is A negative
Yes
No
Load address part of upper instruction of step 01101 in index register 2
Load address part of lower instruction of step 01101 in index register 3
Jump to 00200

	If A is positive jump to	Load index register 2 with address part of upper instruction of 01101
(00100) =	AJP 2 0 0 0 1 0 2	LIU 2 0 1 1 0 1
	Unconditional jump	Pass
(00101) =	SLJ 0 0 0 2 0 0	ENI 0 0 0 0 0 0
	Load index register 3 with lower side of 01101	Unconditional jump to 00200
(00102) =	LIL 3 0 1 1 0 1	SLJ 0 0 0 2 0 0

Explanation: After the test on the contents of A, the "LOAD INDEX (upper)" or the "LOAD INDEX (lower)" instructions place the address portions of the instructions of 01101 in the respective index registers. Note the use of the PASS instruction at Step 00101.

EXAMPLE 34. When the program reaches step 00362, it is desired to shift the A register contents left and the Q register contents right. The number of left shifts is to be determined by the contents of index register 4 and the number of right shifts by the contents of index register 5. This is done by the following two program steps.

	Store contents of index register 4 in upper address of 00362	Store contents of index register 5 in lower address of 00362
(00361)	SIU 4 0 0 3 6 2	SIL 5 0 0 3 6 2
	A Left shift	Q Right shift
(00362)	ALS 0 0 0 0 0 0	QRS 0 0 0 0 0 0

Is it possible to accomplish the same thing by using program step 00362 only above? Show step 00362

Although the following exercises are provided for practice on the last seven instructions of this chapter, some general review is also provided on using the instructions up to this point. Check your solutions with those given in Appendix A. The number of correct answers will indicate your approximate progress by checking the Rating Table below:

Rating Table 9

If you have 5 correct answers	Excellent
If you have 3 or 4 correct answers.	Good
If you have 2 correct answers	Some Review
If you have 0-1 correct answers	Complete Review

EXERCISES 9

a. A trucking concern charges a flat rate plus a zone rate. Zone rates are stored in index registers 1 through 4. Minimum distances have no zone rate. Assuming the flat rate is stored in index register 6, write steps for the total charge to a company which required service for 1 minimum distance, and 1 distance for each of the zones 1 through 4 (total of 5 distances). (Try to do this by using "STORE INDEX" and "INCREASE INDEX" instruction.)

b. Write program steps which will count the number of index registers, 1 through 6, which contain negative quantities. (Start with step 00154.)

c. Write program steps which will modify the upper instruction of step 00026 by adding an octal digit (0-7) which is contained in index register 2 to the operation code bits (left 3 octal digits) of the instruction.

d. Write program steps, which will increase the contents of index register 6 by the sum of the contents of index register 4 and the contents of Q.

e. A subroutine consisting of 42_{10} steps is coded with consecutive addresses starting with 00000. A programmer wants to store these 42_{10} steps at consecutive locations starting with 00377. Use is made of index register 3 to do this. Explain or show how it can be done.

CHAPTER IV

REVIEW TEST

The following questions review the fifteen instructions presented in this chapter. Write answers in the spaces provided after each question. Solutions are given in Appendix B.

1. The "two-way jump" instructions such as STORAGE SKIP, STORAGE SHIFT, and INDEX SKIP should be used as which instruction in a program step? (Upper or lower)

   ____

2. The STORAGE SKIP instruction senses the sign bit of a quantity in a specified storage location. If it is plus, what action is taken?

   ____

3. What is the main difference between the STORAGE SKIP and STORAGE SHIFT instructions?

   ____

4. If the content of index register 3 is 00001, and the INDEX SKIP instruction 5 4 3 0 0 0 0 1 is executed, what will occur?

   ____

5. Assume the content of index register 4 is 00002. If the INDEX JUMP instruction 5 5 4 0 0 1 0 0 is given, how many times will a jump to 00100 take place?

   ____

6. In trying to extract bits from a word, name three places where the extractor may be stored.

   ____

   ____

7. Show the rules of multiplication which govern a logical product. ____________________

8. If one wishes to save the bits in a word, he should store zeros in the corresponding positions of the extractor. (True-False) ____________________

9. Most of the logical instructions destroy the original contents of the A and Q registers. (True-False)

____

10. If one wanted to extract the four right-most bits of a word, what would the extractor be? ______________________

11. Can the logical product be zero? (Yes or No)

____

12. Show the PASS instruction. ______________________

____

13. The PASS instruction can not be used in the second or lower position of a program step (True-False)

____

14. The ENTER INDEX instruction and the INCREASE INDEX instruction make use of the value specified by the execution address rather than referencing a storage location. (True-False)

____

15. Write program steps which will find the sum of the octal digits contained in index register 5 and deliver the sum to index register 3.

CHAPTER V

FOURTH GROUP OF INSTRUCTIONS

FOURTH INSTRUCTIONS (with Mnemonic and Numeric Codes)

MULTIPLY FRACTIONAL	(MUF.b)(26.b)	EQUALITY SEARCH	(EQS)(64.b)
DIVIDE FRACTIONAL	(DVF.b)(27.b)	THRESHOLD SEARCH	(THS)(65.b)
SELECTIVE STOP	(SLS.b)(76.b)	MASKED EQUALITY	(MEQ)(66.b)
		MASKED THRESHOLD	(MTH)(67.b)
SELECTIVE SET	(SST.b)(40.b)	REPLACE ADD	(RAD)(70.b)
SELECTIVE CLEAR	(SCL.b)(41.b)	REPLACE SUBTRACT	(RSB)(71.b)
SELECTIVE COMPLEMENT	(SCM.b)(42.b)	REPLACE ADD ONE	(RAO)(72.b)
SELECTIVE SUBSTITUTE	(SSU.b)(43.b)	REPLACE SUBTRACT ONE	(RSO)(73.b)

The first three of the above are similar to other instructions which have previously been described. However, they involve unique features which make them different to program.

MULTIPLY FRACTIONAL (MUF)(26.b)(where b may be 0 through 7)

This instruction forms a 96-bit product from two 48-bit operands. All quantities in this operation are treated as fractions with the binary point immediately to the right of the sign bit. (The sign bit here is the single left-most bit of the register.) The multiplier must be loaded into the A register prior to the execution of the instruction. The multiplicand is read from the storage location specified by the sum of the execution address and the contents of the designated index register. The product is formed in the AQ register and the multiplier is discarded in the multiplication process. The following examples indicate the process.

Multiply 1/8 by 1/4

Assume 1/8 is stored at address (00100) as $\overset{\longmapsto 48 \text{ bits} \longleftarrow\!\!\mid}{0.0010 \longleftrightarrow 0}$

Assume 1/4 is stored in register A as $\overset{\longmapsto 48 \text{ bits} \longleftarrow\!\!\mid}{0.0100 \longleftrightarrow 0}$

The product in the AQ register will then be:

$0.00 \quad 00100 \longleftrightarrow 0$

96 possible bits

The correct answer then will be

0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (in octal)

Multiply 5/8 by 1/4

```
                     |<——48 bits——>|
     5/8 =            0.1010<————>0

                     |<——48 bits——>|
     1/4 =            0.010<—————>0
```

Product in AQ will be:

```
                     |<——96 bits———>|
                      0.001010<————>0
```

Expressed in octal the product is

```
                      0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

DIVIDE FRACTIONAL (DVF) (27.b)(where b may be 0 through 7)

This instruction divides a 96-bit dividend by a 48-bit divisor. All quantities involved in this operation are treated as fractions with the binary point immediately to the right of the left-most bit (sign bit). The 96-bit dividend must be loaded into the AQ register prior to the execution of this instruction. The 48-bit divisor is read from the storage location specified by the sum of the execution address and the contents of the designated index register. At the end of the operation, the quotient is left in the A register and the remainder is left in the Q register. (Note this is just the reverse of the DIVIDE INTEGER.) The quotient and remainder bear the same algebraic sign.

Example   Divide 1/8 by 1/4

Assume AQ contains 0.0010<——————>0 = 1/8

Assume storage location 00100 contains 0.010<————>0 = 1/4

DIVIDE INSTRUCTION   DVF 0 00100

The quotient in A will be 0.10<————>0 = 1/2
and the remainder in Q is zero.

Expressed in octal the quotient is

```
                      2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

SELECTIVE STOP (SLS)(76.b)(where b may be 0 through 7)

This instruction causes the program to stop on specified conditions of operator lever keys on the console. This is similar to the SELECTIVE JUMP settings discussed previously. The index registers are not used for address modification in this instruction. The three-bit index designator specifies which stop key is sampled in determining the stop decision. A jump to the base execution address occurs regardless of the stop decision as shown below:

IF INDEX DESIGNATION IS	COMPUTER ACTION IF CORRECT STOP KEY IS SET	COMPUTER ACTION UPON RESTART OR IF STOP KEY IS NOT SET
0	Stops unconditionally	Jump to Execution address
1	Stops if stop key 1 is set	Jump to Execution address
2	Stops if stop key 2 is set	Jump to Execution address
3	Stops if stop key 3 is set	Jump to Execution address
4	Stops unconditionally	Return jump to Execution address
5	Stops if stop key 1 is set	Return jump to Execution address
6	Stops if stop key 2 is set	Return jump to Execution address
7	Stops if stop key 3 is set	Return jump to Execution address

SELECTIVE SET (SST)(40.b)(where b may be 0 through 7)

This instruction sets individual bits in the A register to "one" where there are corresponding ones in the word at the storage location address specified by the sum of the execution address and the contents of the designated index register. This is a bit-by-bit function and does not involve normal addition. FIG. 13 indicates the procedure:

(a) IF NO INDEX REGISTER IS DESIGNATED	(b) IF INDEX REGISTER IS DESIGNATED
Instruction in Octal 4 0 0 0 0 1 5 0	Instruction in Octal 4 0 3 0 0 1 5 0
Assume (A) = 1 1 1 1 1 $0 \longleftrightarrow 0000$	Assume index register 3 = 00005
Assume (00150) = $000000 \longleftrightarrow 0111$	Sum of execution address and contents of index register 3 is 0 0 1 5 5
The three one bits in (00150) will set the corresponding bits in A to ones, so that (A) becomes	Operand is found at assume (00155) = $0 \longleftrightarrow 01110000$
(A) = 1 1 1 1 1 $0 \longleftrightarrow 0$ 1 1 1	Assume (A) = $1 \longleftrightarrow 10000000$
	The three one bits in (00155) will set one bits into corresponding positions in A (A) becomes $1 \longleftrightarrow 11110000$

FIG. 13

Setting one bits in A Register

SELECTIVE CLEAR (SCL)(41.b)(where b may be 0 through 7)

This instruction clears individual bits of the A register where there are ones in corresponding bit-positions in the quantity in storage whose location is specified by the sum of the execution address and the contents of the designated index register. FIG. 13 indicates the same procedure except zeros are set in (A) instead of ones.

SELECTIVE COMPLEMENT (SCM)(42.b)(where b may be 0 through 7)

This is the same as the SELECTIVE SET instruction except this instruction complements individual bits in the A register where there are ones in corresponding bit positions in the quantity in storage specified by the sum of the execution address and the designated index register. FIG. 13 can also be used to indicate the process, except the ones in the control quantity complement the corresponding ones in A.

SELECTIVE SUBSTITUTE (SSU)(43.b)(where b may be 0 through 7)

This instruction substitutes portions of an operand into the A register using the Q register as a mask. This action may be considered in two steps. First individual bits in A are cleared where there are corresponding ones in the Q register. Second, these same (cleared) individual bits in A are replaced with the corresponding bit values from the storage location specified by the sum of the execution address and the contents of the designated index register.

The following examples should facilitate learning the previous seven instructions. (Each student is advised to work out the examples and to check the given solutions.)

EXAMPLE 35. Assume storage location (00111) contains the fraction, 1 2 3 3 0 0 0 0 0 0 0 0 0 0 0 0. A programmer wants to square this quantity, store the result in storage location (00112), and stop. Write program steps to do this and show the contents of storage location (00112) and the AQ register after the MULTIPLY FRACTIONAL is executed.

	Load Multiplier in A	Fractional Multiply
(00050)	LDA 0 0 0 1 1 1	MUF 0 0 0 1 1 1
	Store A in 00112	Unconditional Stop
(00051)	STA 0 0 0 1 1 2	SLS 0 0 0 2 0 0

The product in the AQ register will then be 000011011001001110110010 0 0
which is expressed in octal as 0331166200000000

EXAMPLE 36. It is desired to use the DIVIDE FRACTIONAL instruction to divide 324 (octal) by 51 (octal). Assume the divisor (51) is stored at address (00042). Assume (324) is stored at address (00040). Show the program steps, the format in which the 324 and 51 should be stored, and the final contents of the A and Q registers after the DIVIDE.

Store 324 in (00040) as $\longrightarrow$ 1520000000000000

Store 51 in (00042) as $\longrightarrow$ 2440000000000000

	Load A .	Zeros to Q
(00100) =	LDA 0 0 0 0 4 0	ENQ 0 0 0 0 0 0

	Divide Fractional	
(00101) =	DVF 0 0 0 0 4 2	

Final Contents of A The Quotient	Final Contents of Q The Remainder
2 4 5 3 5 5 2 1 1 2 7 3 2 4 2 2	0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 6

EXAMPLE 37. The first time a program reaches step 01005, the program is to jump to step 00200. The second time when step 01005 is reached the program is to stop. Explain how this could be accomplished using a SELECTIVE STOP instruction.

Explanation: There are several ways this might be done. A SELECTIVE STOP with index designation 1 could be used and the stop key 1 selected in the console. When the program reached step 01005 the computer would stop but by depressing the START button it could jump to 00200. The second time it would also STOP. Another method might be to use the SELECTIVE STOP with a designation, but a different stop key setting, for example,76 2 00200 with stop key 1 is set. The first time, there would be no stop and the program would jump to 00200. Before reaching stop 01005 the second time a modification of 01005 (7 6 2 0 0 2 0 0) to 7 6 1 0 0 2 0 0 could be made and this time the STOP would occur.

Explain what will occur if program steps (00050) and (00051) are given and stop key 3 is the only key setting on the console.

(00050) =	SLS 1 00100	SLS 2 00200
(00051) =	SLS 3 00300	SLS 0 00100

<u>EXAMPLE 38.</u> Assume the A register contains eight 6-bit characters. If the content of Q is positive, it is desired to set the left-most bit of each of the eight characters to one. If the content of Q is negative, the left-most bit of each of the eight characters is to be set to zero. If the sign of the contents of address 00011 is used to set the ones and zeros, show the program steps and the contents of address 00011.

	Test O for positive	Set bits in (A) corresponding to one bits in 00011 to zeros
(00050) =	QJP 2 0 0 0 5 2	SCL 0 0 0 0 1 1
	Jump Exit	Pass
(00051) =	SLJ 0 0 0 0 5 3	ENI 0 0 0 0 0 0
	Set bits in (A) to ones	Pass
(00052) =	SST 0 0 0 0 1 1	ENI 0 0 0 0 0 0

```
                                          (in bits)
Contents of address 00011 ——→100000 100000 100000....... etc.
     (In octal) ————————→4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0
```

<u>Explanation:</u> First Q is tested for positive content with Q-JUMP. If positive, jump occurs to 00052 where "ones" are set in A by the pattern in address 00011. If Q is negative, jump does not occur, and "zeros" are set in A by pattern in 00011. Note "exit" jump needed in step 00051. Why?

EXAMPLE 39. The numeric code for the Q-Jump with zero index designation is 2 3 0. The numerical code for the A-Jump with zero index designation on 2 2 0. At step 00117 in a program the Q JUMP, "2 3 0 0 0 1 5 0", is given, followed by a PASS instruction. On the second time through this step, change the Q-Jump to the A-Jump " 2 2 0 0 0 1 5 0". The third time, it is to be returned to its original form, etc. Each time through this step, it is desired to alternate between a Q-Jump and an A-Jump. Assume the necessary modification is done at step 00150 by using the SELECTIVE COMPLEMENT instruction. Show this instruction and the contents of the storage address containing the necessary "change pattern".

	Load (A) with step 00117	Complement bits in A by pattern of ones in address 00200 (see below)
(00150) =	LDA 0 0 0 1 1 7	SCM 0 0 0 2 0 0
(00151) =	STA 0 0 0 1 1 7	Next Instruction

Where (00200) contains (in bits) $\longrightarrow$ 0000010 $\longleftrightarrow$ 0
or(in octal) $\longrightarrow$ 0100000000000000

```
                       Q-Jump          Pass
Explanation:  First 230  00150 500   00000 is LOADED into A.
```
"SELECTIVE COMPLEMENT" then changes the 230 part of instruction to 220. The next time it will be reset to 230, etc. Then A is stored in 00017.

EXAMPLE 40. Assume storage location 00100 contains the following:

```
140 00600   200 00300
```

It is desired to change this to the following:

```
142 00600   200 00426
```

To do this, patterns are set up in Q and address 00050. The "SELECTIVE SUBSTITUTE" instruction used. Show the steps necessary and the contents of Q and address 00050.

	Load Pattern into	Load Instructions to be modified into A
(00022) =	LDQ 0 00051	LDA 0 00100

	Selective Substitute	Store Modified Instructions
(00023) =	SSU 0 00050	STA 0 00100

Contents of 00051 = contents of Q = 00700000 00000777

Contents of 00050 $\longrightarrow$ 00200000 00000426

Explanation: The pattern of sevens in Q clears the corresponding octal digits in the A register. The octal digits in 00050 then replace these corresponding cleared positions in A. The modified instruction is then returned to 00100 by "STORING A".

The following exercises are provided for practice on the first seven instructions of this chapter. Some general review may also be included in the exercises. Check your solutions with these given in Appendix A. The number of correct answers will indicate your approximate progress by checking the Rating Table below:

Rating Table 10

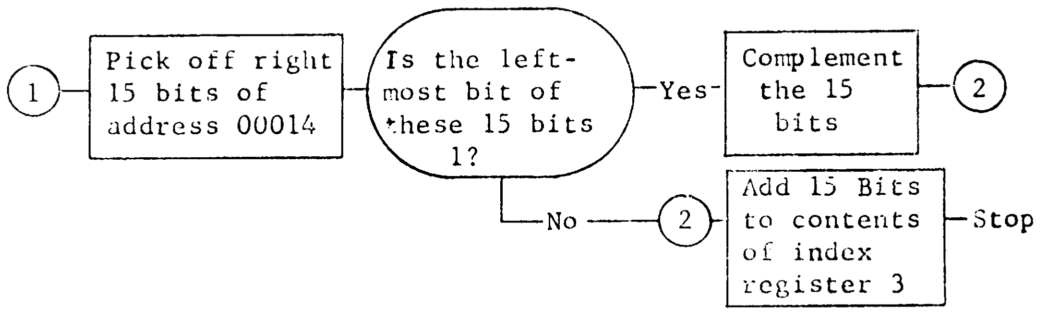
If you have 5 correct answers	Excellent
If you have 4 correct answers	Good
If you have 3 correct answers	Some Review needed
If you have 0-2 correct answers	Complete Review

EXERCISES 10.

a. Storage address 00025 contains a tax rate whose range is between 1.5% and 2.5%. A company has found that 37.5% (stored in address 00026) of this tax rate is a good barometer of business trend in general. The procedure is to calculate this barometer, round it to three digits and subtract the result from 1. Program this, starting at step 00050 and using the given address contents above.

b. Starting at storage address 00011, there are 16_{10} quantities stored. Each negative quantity is to have its right-most 3 bits replaced by one bits. Each positive quantity is to have its right-most 3 bits replaced by zero bits. Show program steps starting at step 00200 to do this. Show contents of all addresses used as constants.

c. Program the following flow chart:


1
Pick off right 15 bits of address 00014
Is the left-most bit of these 15 bits 1?
Yes
Complement the 15 bits
2
No
2
Add 15 Bits to contents of index register 3
Stop

d. Write a program which will count the non-significant octal zeros contained in storage location 00100.

e. Starting at step 00300 show a program which will exchange the contents of index register 4 with the right-most 15 bits contained in storage location 00100.

The next four instructions are similar in that they are search instructions and they are extensions of the "two address jump" instructions. These instructions are also limited to use in the upper position of program steps. The lower position in such a program step is normally an unconditional jump instruction. Thus the first address (in the upper instruction) specified the location of the operands required in the jump decision; whereas, the second address (in the lower instruction) specifies the destination of the jump if taken. If these search instructions are used in the second position of program steps, no skip will take place which removes most of the power of the instruction.

EQUALITY SEARCH (EQS)(64.b)(where b may be 0 through 7)

This instruction searches a list of operands beginning at the search address specified by the sum of the execution address and the contents of the designated index register minus one and continuing back toward the execution address until an operand is found that is equal in value to the contents of the A register or until the operand at the execution address has been searched. Before each operand is searched, the search address is reduced one count at a time (by reducing the index register contents) until the equality is found or the base execution address is reached. If an operand is found that is equal to the contents of the A register, the search is terminated and the next instruction is skipped. If no operand in the list is equal to the contents of the A register, then the next instruction is executed.

It is important to note that if an index register designation of zero is used, the search will only be made on one operand - - that is the operand located at the base execution address.

Usually the instruction which follows the search instruction is a jump. When the search criterion is met, this instruction is skipped. The programmer often desires to know the address of the register containing the value which equals the contents of the A register. This is easily accomplished by adding the reduced count in the designated index register to the base execution address.

The following flow chart of the SEARCH procedure is outlined in FIG. 14. This should further indicate the procedures followed by the computer while executing this type of an instruction. Also note that one stores the exact number for the number of searches to be made since the preliminary reduction made by the machine adjusts for this.

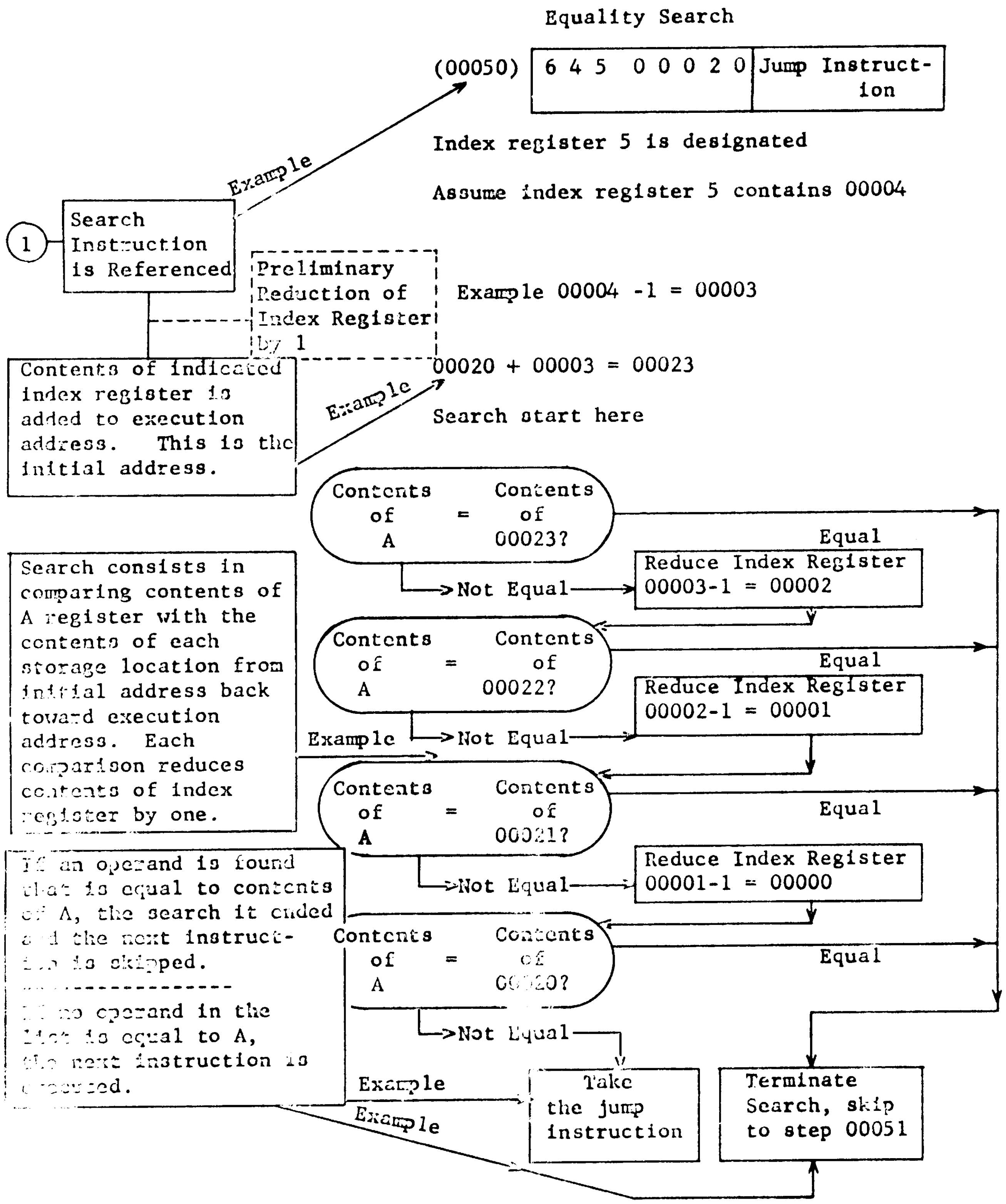

Equality Search
(00050)
6 4 5 0 0 0 2 0
Jump Instruct-ion
Index register 5 is designated
Assume index register 5 contains 00004
Example
1
Search Instruction is Referenced
Preliminary Reduction of Index Register by 1
Example 00004 -1 = 00003
Contents of indicated index register is added to execution address. This is the initial address.
Example
00020 + 00003 = 00023
Search start here
Contents of A = Contents of 00023?
Equal
Not Equal
Reduce Index Register 00003-1 = 00002
Contents of A = Contents of 00022?
Equal
Not Equal
Reduce Index Register 00002-1 = 00001
Search consists in comparing contents of A register with the contents of each storage location from initial address back toward execution address. Each comparison reduces contents of index register by one.
Example
Contents of A = Contents of 00021?
Equal
Not Equal
Reduce Index Register 00001-1 = 00000
If an operand is found that is equal to contents of A, the search it ended and the next instruction is skipped.
If no operand in the list is equal to A, the next instruction is executed.
Contents of A = Contents of 00020?
Equal
Not Equal
Example
Take the jump instruction
Example
Terminate Search, skip to step 00051

FIG. 14

Execution of a Search Instruction

THRESHOLD SEARCH (THS)(65.b)(where b may be 0 through 7)

This instruction is similar to the EQUALITY SEARCH except it searches a list of operands for one which is greater than the value in the A register. If an operand is found which is greater than the value in the A register, the search is terminated and the next instruction is skipped. If no operand in the list is greater than the value in the A register, the next instruction is executed.

The search begins at the address specified by the sum of the execution address and the contents of the designated index register minus one and continues back toward the execution address value. Before each storage address is searched, the contents of the index register is reduced one count.

If a zero index register designation is used, then the computer searches just one address, viz., the one whose location is given by the execution address part of the instruction. FIG. 14 indicates the search process with everything being the same except the comparison is made to see if any operand is greater than the value in the A register rather than for equality. Also note that the programmer stores the exact number of searches he desires in an index register since the preliminary reduction takes care of the necessary adjustments.

MASKED EQUALITY (MEQ)(66.b)(where b may be 0 through 7)

This instruction searches a list of operands starting at the location specified by the sum of the execution address and the contents of the designated index register. If an operand is found such that the logical product of the operand and the contents of the Q register is equal to the contents of the A register, the search is terminated and the next instruction is skipped. If no operand in the list meets this requirement, the next instruction is executed. As in the case of the previous two search instructions, the search begins at the location specified by the sum of the execution address and the contents of the designated index register minus one and continues backward through consecutive storage location toward the execution address value. Before each search is made, the contents of the index register is reduced by one count. The search continues until an operand satisfying the requirement is met, or until the operand in the base execution address has been searched.

A zero index designation infers that no index register is being used and as a result only one storage location is searched, viz., the one located at the base execution address. FIG. 14 can be used to trace this instruction using the criterion "MASKED EQUALITY" instead of the "EQUALITY" criterion shown therein.

MASKED THRESHOLD (MTH)(67.b)(where b may be 0 through 7)

This instruction is the same as the MASKED EQUALITY instruction except the search is made on the criterion of trying to find an operand such that the logical product of the operand and the contents of the Q register is greater than the contents of the A register. If such an operand is found, the search is ended and the next instruction is skipped. If no such operand is found the next instruction is executed. The search begins at the location specified by the sum of the execution address and the contents of the designated index register minus one and continues backward through consecutive storage locations whose addresses are above the base execution address. Before each address is searched, the contents of the designated index register is reduced one count until the contents become zero (this will be the situation when the base execution address has been reached and searched).

A zero index designation will result in just one stored location being searched - that one designated by the base execution address. FIG. 14 can be used as indicative of the process followed except the MASKED THRESHOLD criterion should be substituted for the EQUALITY criterion shown therein.

The following examples should facilitate learning the four previous search instructions. (Each student is advised to work out the given examples and to check the given solutions.)

<u>EXAMPLE 41.</u> There are 100_{10} constants stored in consecutive locations starting at address 00100. Show steps starting at 00011 needed to count the number of zero quantities in these addresses.

	Enter zeros into index register 2		Pass		
(00011) =	ENI 2	0 0 0 0 0	ENI 0	0 0 0 0 0	
	Enter zeros into A		Enter 100_{10} into index register 3		
(00012) =	ENA 0	0 0 0 0 0	ENI 3	0 0 1 4 4	
	Equality Search		This jump will occur if equality is not found, it will be skipped if equality is found		
(00013) =	EQS 3	0 0 1 0 0	SLJ 0	0 0 0 1 7	
	Increase index register 2 by one count		Index jump index register 3 is examined for zero. If not zero, reduce index register 3 by 1,		
(00014) =	INI 2	0 0 0 0 1	IJP 3	0 0 0 1 6	Jump to 00016
	Pass		Jump to exit		
(00015) =	ENI 0	0 0 0 0 0	SLJ 0	0 0 0 1 7	
	Increase Index register 3 by one count		Continue search at (00013)		
(00016) =	INI 3	0 0 0 0 1	SLJ 0	0 0 0 1 3	

(00017) = Exit. When exit is reached the count of the number of zeros will be in Index Register 2.

<u>Explanation:</u> Zeros are first put into index register 2 which is used as a counter for the number of zero quantities found. The number of storage locations to be searched is entered into index register 3. (Note this is 144 in octal since $100_{10} = 144_8$.) At step 00013, the EQUALITY SEARCH instruction is given. The next instruction (75 000017) will be executed as long as no equality is found. Thus if all 100 addresses do not contain zero, a jump to 00017 will be made. If a zero is found, this jump is skipped and at step 00014, the contents of index register 2 (the counter) is increased by 1.

The INDEX JUMP instruction is then used to examine the contents of index register 3 for zero (which would mean all 100 locations had been searched before a zero was found). If the contents of index register 3 are not zero, the INDEX JUMP instruction reduces it by 1. Thus it is necessary to add 1 to the register before jumping back to continue the search. For this reason, the instruction at step 00016 is used to increase the index register 3 by 1.

EXAMPLE 42. A programmer desires to search 50_{10} storage addresses starting with address 00200 to find the first one of these addresses containing the quantity 7. He wants to know the address of the first one of these locations which contains a 7. Show program steps starting at 00100.

	Enter 7 into (A)	Enter 50_{10} into Index Register 6
(00100) =	ENA 0 0 0 0 0 7	ENI 6 0 0 0 6 2
		If no address contains 7, jump to exit
(00101) =	EQS 6 0 0 2 0 0	SLJ 0 0 0 1 0 3
	Enter A with 200 + (B^6)	Store Result in 00050
(00102) =	ENA 6 0 0 2 0 0	STA 0 0 0 0 5 0

Explanation: First 7 is put into A and $50_{10} = 62_8$ is put into index register 6. The search instruction is then executed. When the first address containing 7 is found, step 00102 is executed. At this time the sum of the contents of index register 6 and the execution address (00200) of the SEARCH instruction will give the address of the first location containing a 7. The ENTER A INSTRUCTION, of step 00102, puts the sum of 200 and the contents of index register 6 into the accumulator and from here it is stored in location 00050.

EXAMPLE 43. There are 1000_{10} tax accounts stored in consecutive storage locations starting at 00126. Write program steps, starting at 00010 which will transfer those accounts greater than 9450_{10} into consecutive addresses starting at location 03000.

	Enter 1000_{10} into index register 4	Pass
(00010) =	ENI 4 0 1 7 5 0	ENI 0 0 0 0 0 0
	Enter 9450_{10} into the A register	Pass
(00011) =	ENA 0 2 2 3 5 2	ENI 0 0 0 0 0 0
	Threshold Search	If no accounts are found that are greater than 9450_{10}, jump to exit at step 00017
(00012) =	THS 4 0 0 1 2 6	SLJ 0 0 0 0 1 7
	Pass	Pass
	Enter A with 126 + (B^4)	Substitute generated "address" into the address portion of upper instruction at step 00014
(00013) =	ENA 4 0 0 1 2 6	SAU 0 0 0 0 1 4
	Load A with contents of address (xxxxx)	Store at locations starting at 03000
(00014) =	LDA 0 (x x x x x)	STA 0 (0 3 0 0 0)
	Load previous program step into A register	Increase A by 1 - which modifies storage location (03000) of previous step
(00015) =	LDA 0 0 0 0 1 4	INA 0 0 0 0 0 1
	Modified step sent back to original location	Jump back to start search again
(00016) =	STA 0 0 0 0 1 4	SLJ 0 0 0 0 1 2

Explanation: After finding the addresses of the locations containing quantities greater than 9450_{10}, the contents of these addresses must then be transferred to addresses starting at 03000. This is the reason for the right instruction at 00013 and step 00014. Another method, referred to as Indirect Addressing, is an alternative to perform this operation. Indirect addressing is described in the "Operational and Technical Characteristics" manual.

EXAMPLE 44. Starting at location 00500 there are 500_{10} quantities stored in consecutive addresses. The right-most bit of each of these quantities is reserved for parity data. If any one of these contain a parity designation of 1, the 500_{10} addresses are to be cleared. Show program steps starting at 00071.

	Enter extractor in Q register	Enter comparator in A register
(00071) =	ENQ 0 0 0 0 0 1	ENA 0 0 0 0 0 1
	Enter search number into index register	Enter 500_{10} into index register 3 for a counter for index jump (see 00076)
(00072) =	ENI 2 0 0 7 6 4	ENI 3 0 0 7 6 3
	Start Masked Equality Search	If no parity is found in all of the locations jump to exit at (00077)
(00073) =	MEQ 2 0 0 5 0 0	SLJ 0 0 0 0 7 7
	Enter zeros in a register	Clear address register contents starting at (00500)
(00074) =	ENA 0 0 0 0 0 0	STA 0 (0 0 5 0 0)
	Load previous step into A register	Increase A by 1 which increases storage designation of lower instruction of step 00074
(00075) =	LDA 0 0 0 0 7 4	INA 0 0 0 0 0 1
	Store modified step back in original location	Index jump based upon contents of index register 3
(00076) =	STA 0 0 0 0 7 4	IJP 3 0 0 0 7 4

Explanation: The extractor entered into Q picks off the right-most bit (parity designation in this example) and compares it with contents of the A register. If equality is found (indicating an address contains parity 1) step 00074 is taken. Steps 00074 and 00075 clear the addresses starting at 00500. The INDEX JUMP at (00076) controls the number of times (500) the clearing takes place.

Note that program step 00075 modifies the previous program step by adding a 1 to it. There is a "neater" method of modifying by using an index register. For example, step 00074 could have been LDA 0 00000 STA 5 00500 where index register 5 contains zero initially. Then this index register can be increased by 1 before looping back. Write the above program to include this method.

<u>EXAMPLE 45.</u> 60_{10} consecutive locations starting at address 00100 contain either eight alphabetic characters of 6 bits each, or eight numeric characters of 6 bits each. The formats for numerical and alphabetic characters are shown below: (where symbol "x" may be a 0 or 1).

Each numeric character is of the form 00xxxx (6 bits)
Each alphabetic character is of the form 01xxxx (6 bits)

Numerical and alphabetic characters will not appear in the same storage address.

Show program steps which will store in address 00200 the address of the first alphabetic location found.

```
                Enter extractor        Enter zeros into
                into Q Register        A register

(c)     =   | ENQ 0   0 0 0 2 0  | ENA 0   0 0 0 0 0 |

                Enter search count
                60₁₀, into index register 5    Pass

(c+1)   =   | ENI 5   0 0 0 7 4  | ENI 0   0 0 0 0 0 |

                 Start Masked           If no locations contain alphabetic
                 Threshold Search       characters jump to exit at (c+5)

(c+2)   =    | MTH 5   0 0 1 0 0  | SLJ 0   (c+5) |

                 Enter contents of index  Increase contents of A by 100
                 register 5 into A        to get address of first alpha-
                 register                 betic location
(c+3)   =    | ENA 5   0 0 0 0 0  | INA 0   0 0 1 0 0 |

                 Store address in
                  (00200)                  Pass

(c+4)   =    | STA 0   0 0 2 0 0  | ENI 0   0 0 0 0 0 |


(c+5)   =    Continue Program
```

The following exercises are provided for practice on the four SEARCH instructions just described. Some general review is also included in these exercises. Check your solutions with these given in Appendix A. The number of correct answers will indicate your approximate progress by checking the Rating Table below:

Rating Table 11

If you have 4 correct answers	Excellent
If you have 3 correct answers	Good
If you have 2 correct answers	Fair
If you have 0-1 correct answers	Review

EXERCISES 11

(a) 50_{10} consecutive addresses starting with 00100 contain accounts "C", 50_{10} consecutive addresses starting with 00200 contain accounts "D". If any of accounts "C" equal accounts "D" store a 1 in register 00032 and stop. Write program steps for this, starting at step 00400.

(b) Use SEARCH instructions to do the following: if the contents of address 00200 is greater than the contents of address 00201, jump to step 00400; if the contents of address 00200 equals the contents of address 00201, jump to step 00500.

(c) Given the following portion of a program. Explain what this is attempting to do.

(00050)	ENI 6 0 0 1 4 4	ENA 6 0 0 0 0 0
(00051)	THS 6 0 0 1 0 0	SLJ 0 0 0 0 6 0
(00052)	ENQ 0 0 0 0 0 0	STQ 0 0 0 5 0 0

(d) Explain what would occur in the following two separate program steps. (Treat each program step as a separate occurrence.)

I.	(00700)	MTH 3 0 0 0 5 0	SLJ 0 0 0 7 0 0
II.	(00105)	LDA 0 0 0 0 5 0	EQS 3 0 0 2 0 0

The last four instructions of this chapter are the REPLACE ADD and REPLACE SUBTRACT instructions. In programming one is often concerned with modifying an instruction and replacing it in its original location. Tl .s is the primary function performed by these instructions. They are described below.

REPLACE ADD (RAD)(70.b)(where b may be 0 through 7)

This instruction replaces the quantity whose location is given by the sum of the execution address and the contents of the designated index register, by its original value plus the contents of the A register. The resultant sum is also left in the A register.

Thus if A contains 0000000000000003 (in octal) and if address 00100 contains 0000000000000005, then RAD 0 00100 will replace the above contents of 00100 with 0000000000000010 (since 3+5 = 10 in octal). This same result will also be left in the A register.

REPLACE SUBTRACT (RSB)(71.b)(where b may be 0 through 7)

This is the same as the previous instruction except the process is subtraction. This instruction replaces the quantity whose location is given by the sum of the execution address and the contents of the designated index register, by its original value minus the contents of the A register. The resultant difference is also left in the A register.

REPLACE ADD ONE (RAO) (72.b)(where b may be 0 through 7)

This instruction replaces the quantity whose location is given by the sum of the execution address and the contents of the designated index register with its original value plus one. The resultant sum is also left in the A register after the execution. For example, if A contains 0000000000000010 (in octal), and address 00100 contains 0000000000000006; then "RAO" 0 00100 will replace the contents of address 00100 with 0000000000000007 (6+1 = 7) and this same result (0000000000000007) will be left in the A register.

REPLACE SUBTRACT ONE (RSO) (73.b)(where b may be 0 through 7)

This is the same instruction as the previous one except 1 is subtracted rather than added to the original contents of the address whose location is given by the sum of the execution address and the contents of the designated index register. The resultant difference is also left in the A register after the execution. Using the example given above (see REPLACE ADD ONE), if the instruction had been "RSO" 0 00100, the contents of 00100 would have been 0000000000000005 (6-1 = 5) and this same result would have been left in the A register.

The following examples should facilitate learning the REPLACE instructions, which have just been described. Check each example by working through to your solution and then comparing it with the given solution.

EXAMPLES 46-47. Add the contents of 100_{10} consecutive addresses starting at 00300 to the contents of the 100_{10} consecutive addresses starting at 00500 and store the sums in 100_{10} consecutive addresses starting at 01000. Program this starting at step 00011 by two methods (46) without USING INDEX REGISTERS FOR MODIFICATION AND (47), USING INDEX REGISTERS FOR MODIFICATIONS.

EXAMPLE 46.

		Enter 99_{10} into index register 3 for counter	Pass
(00011)	=	ENI 3 0 0 1 4 3	ENI 0 0 0 0 0 0
		Load A with contents of address (00300)	Add contents of (00500)
(00012)	=	LDA 0 (0 0 3 0 0)	ADD 0 (0 0 5 0 0)
		Pass	Store in (01000)
(00013)	=	ENI 0 0 0 0 0 0	STA 0 (0 1 0 0 0)
		Add 1 to contents of step 00013 and replace	Add 1 to contents of 00012 and replace. (This modified the (00600) above.)
(00014)	=	RAO 0 0 0 0 1 3	RAO 0 0 0 0 1 2
		Add contents of 00050 to contents of A	Store in 00012. (This will modify the (00300) above)
(00015)	=	ADD 0 0 0 0 5 0	STA 0 0 0 0 1 2
		Index Jump back to 00012 for 99 times	Where (00050) = 00000001 00000000
(00016)	=	IJP 3 0 0 0 1 2	Program continues

Note: The only use of an index register was as a counter and not to modify. Location (00050) = 0000000100000000

EXAMPLE 47. (Same as previous example except using index register to modify instructions.)

```
          Enter zero into
          index register 4        Pass
(00011) | ENI 4   0 0 0 0 0 | ENI 0   0 0 0 0 0 |

          Load A with contents   Add the contents of the address
          of "00500 + contents   located by the sum of 00300 and
          index register 4"      contents of index register 4
(00012) | LDA 4   0 0 5 0 0 | ADD 4   0 0 3 0 0 |

          Store in location
          given by (01000) +          Pass
          contents index register 4
(00013) | STA 4   0 1 0 0 0 | ENI 0   0 0 0 0 0 |

          Index Skip checks           Jump back to
          contents of index register  step 00012
          4
(00014) | ISK 4   0 0 1 4 3 | SLJ 0   0 0 0 1 2 |
```

Explanation: Index register 4 contains a zero to start. This means that 00100, 00300, and 01000 will be used as the location of the operands in steps 00012 and 00013 the first loop through. However, step 00014 (INDEX SKIP) checks the contents of index register 4 against the execution address part of the instruction (99_{10}) and if not equal adds 1 to index register 4 and takes the next instruction (JUMP). The second loop through, 00101, 00301, and 01001 will be used as the location of the operands in steps 00012 and 00013 since index register 4 now has a 1 in it, etc., until the INDEX SKIP at step 00014 sees the index register 4 contains 99_{10} and skips the jump. (Why is 99_{10} instead of 100_{10} used?)

EXAMPLE 48. Examine the contents of the A register for sign. If positive, increase the contents of 00100 by 1 and decrease the contents of 00101 by 1 and then jump to step 00250. If negative program a selective stop with a restart at 00200. (Start the program steps at 00200.)

```
          If (A) is negative      Replace Add 1 to
          jump to step            contents of 00100
           00202
(00200) | AJP 3   0 0 2 0 2 | RAO 0   0 0 1 0 0 |
          Replace Subtract 1      Jump to step
          to contents of 00101    00250
(00201) | RSO 0   0 0 1 0 1 | SLJ 0   0 0 2 5 0 |

          Stop with restart         Pass
         at step 00200
(00202) | SLS 1   0 0 2 0 0 | ENI 0   0 0 0 0 0 |
          (Where stop key 1 is set on the computer console)
```

Explanation: The REPLACE ADD ONE and the REPLACE SUBTRACT ONE instructions are used when the content of A is positive. Other steps above are self-explanatory.

EXAMPLE 49. It is desired to add to the contents of address 00050 the contents of consecutive addresses 00060 to 00067 inclusive. The total sum is then to be stored in index register 3. Show the program steps starting at 00712 to do this.

	Enter zero into index register 6	Pass
(00712)	ENI 6 0 0 0 0 0	ENI 0 0 0 0 0 0
	Load "A" with (00060) + index register 6	(00050) + (A) (00050)
(00713)	LDA 6 0 0 0 6 0	RAD 0 0 0 0 5 0
	Index Skip checks index register 6	Jump to 00713
(00714)	ISK 6 0 0 0 0 7	SLJ 0 0 0 7 1 3
	Store sum in "A"	Load sum into index register 3
(00715)	STA 0 0 0 1 0 0	LIL 3 0 0 1 0 0

Explanation: Index register 6 is first loaded with a zero. The A register is then loaded with the contents of the location whose address is the sum of 00060 and the contents of index register 6. Thus the first time, contents of address"00060 + 0" = "contents of 00060" goes to A; the next time, contents of "00060 + 1" = "contents of 00061" goes to A, etc. Then a REPLACE ADD instruction replaces the contents of 00050 by the sum of A and the contents of 00050. Each time the contents of 00050 are replaced by the sum of its current contents and the contents of the address then in A. This continues until contents of index register 6 are equal to 00007 in INDEX SKIP instruction at step (00714). Then at step (00715) the current sum (still in A) is entered into index register 3. If the sum exceeded 15 bits, the total in index 3 would not be correct. Assume that the total didn't exceed 15 bits.

Although the following exercises are provided for practice on the REPLACE instructions, some general review is included on the instructions presented up to this point. Check your solutions with these given in Appendix A. The number of correct answers will indicate your approximate progress by checking the Rating Table below:

Rating Table 12

If you have 5 correct answers	Excellent
If you have 3 or 4 correct answers.	Good
If you have 2 correct answers	Fair
If you have 0-1 correct answers	Review

EXERCISES 12

(a) There are 100_{10} positive quantities stored in consecutive locations starting at address 00201. Find the largest of these and store it at address 00077. Start program at step 03001.

(b) A company uses the following formula to compute "Withholding Tax". (Gross Weekly Pay - \$13 (Number of Dependents)x18% = Tax. Program the tax computation starting at step 00101.

Assume:	Gross Weekly Pay is stored in address 00022=xxx.xx
	Number of dependents stored in address 00024 = x.
	Constant 18 is stored in address 00025 = .18.
	Constant 13 is stored in address 00027 = 13.
	Tax (rounded to nearest cent) stored in 00030 = xx.xx

(c) The third octal digit from the right end of each of 100_{10} consecutive registers starting at 00312 contains a code of one octal digit. Write a program to determine the number of codes which are greater than 3. Start program at step 00011 and store result in address 01000.

(d) Write a program starting at step 00312 which will interchange the first and last of the 16 octal digits stored in each of 300_{10} consecutive addresses starting at address 02000.

(e) Each of 50_{10} consecutive registers starting at 00011 contain 2 separate positive numerical quantities, each of 24 bits. Write a program starting at step 01000 which will compare the two quantities in each and order them so that the larger of the two quantities will always be stored in the left 24 bits of each address.

CHAPTER V

REVIEW TEST

The following questions review the fifteen instructions presented in this chapter. Write answers in the spaces provided after each question. Solutions are given in Appendix B.

1. The MULTIPLY FRACTIONAL is similar to the MULTIPLY INTEGER in that the multiplier must be loaded into the Q register prior to execution of the instruction. (True-False)

   ____

2. Where is the product formed in the MULTIPLY FRACTIONAL instruction? (What Registers?)

3. Assume one of the factors used in the MULTIPLY FRACTIONAL is stored as 0100 $\longleftrightarrow$ 011 (in binary). How will the computer treat this quantity during the process (show where binary point will be considered)?

4. What is the largest positive fractional quantity which can be stored directly in its absolute form without scaling and be used by the MULTIPLY FRACTIONAL to get a correct result?

5. In the DIVIDE FRACTIONAL instruction, the dividend must be loaded into the AQ register prior to execution of the instruction. (True-False)

   ____

6. In the DIVIDE INTEGER instruction, the quotient is left in Q and the remainder of the division process is left in the A register. In the DIVIDE FRACTIONAL, the quotient and remainder are left in the same locations. (True-False)

   ____

7. The SELECTIVE STOP instruction, with stop key 3 set on the console, will not stop the computer program unless the index designation of 3 is used in the instruction word. (True-False) ______________________

8. A programmer desires to set corresponding bits in the A register to ones where there are ones in the storage location whose address is 00100. Show the instruction which will be used.

____

9. Using the same conditions given in 8, show the instruction which will set zeros in A instead of ones.

____

10. If the EQUALITY SEARCH instruction is given with an index register designation of zero, what will occur?

____

11. What kind of an instruction generally follows a SEARCH instruction?

____

12. What happens when the SEARCH condition is satisfied?

____

13. What happens if the SEARCH condition is not satisfied?

____

14. After a SEARCH, if one wants to find the address of the location which met the search condition, what must be done?

____

15. Name or explain the four types of SEARCH instructions possible with the 1604.

____

____

____

____

16. Show the instruction which can be used to find if any of the contents of 50_{10} consecutive addresses starting at address 00062 exceed the contents of the A register. Use index register 5 to hold the search count. Show the instruction and the contents of index register 5.

Instruction: _______________

Contents of:

Index register 5: _______________

17. What is the difference between an ADD instruction and a REPLACE ADD instruction?

____

18. Given A = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7

(00100) = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6

Show contents of A and (00100) after execution of the REPLACE SUBTRACT instruction:

```
(RSB)
7 1 0   0 0 1 0 0
```

A Register ______________________________________

(00100) ______________________________________

19. Using the conditions given in 18, show contents of A and 00100 after execution of the REPLACE ADD ONE instruction:

```
(RAO)
7 2 0   0 0 1 0 0
```

A Register ______________________________________

(00100) ______________________________________

20. There are 1000_{10} quantities stored in consecutive addresses starting ad address 00201. Write a program which will advance each quantity three memory locations.

CHAPTER VI

FIFTH GROUP OF INSTRUCTIONS

FIFTH GROUP OF INSTRUCTIONS (with Mnemonic and Numeric Codes) (where b refers to index register designation)

FLOATING ADD	(FAD)	(30.b)	SCALE A	(SCA)	(34.b)
FLOATING SUBTRACT	(FSB)	(31.b)	SCALE AQ	(SCQ)	(35.b)
FLOATING MULTIPLY	(FMU)	(32.b)			
FLOATING DIVIDE	(FDV)	(33.b)			

The first four of these instructions are related to the 1604 floating-point format. The last two instructions will probably be used frequently to "pack" and "unpack" to or from floating-point format. To understand these instructions thoroughly it becomes necessary to know the floating-point format of words. The first part of this chapter is devoted to this subject.

Words packed in floating-point format make it possible to add, subtract, multiply, and divide, and have the computer automatically position the binary point between the integral and functional parts of the results. This feature removes the necessity of the programmer having to continually scale and rescale as the problem progresses.

In order for the computer to do this, it is necessary for all numbers to be expressed in a particular format (floating-point format) before the floating-point instructions are executed. This means that if floating-point instructions are to be used with many factors, a conversion routine (from binary to floating-point) will probably be used on the input parameters. Likewise, after the results are determined (in floating-point format), an output conversion routine (from floating-point to binary) will probably be used. The 1604 floating point format is shown in FIG. 15.

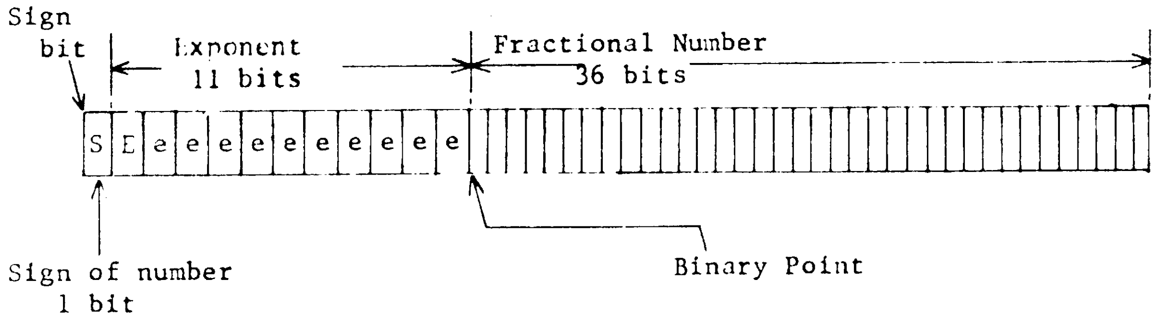


FIG. 15

1604 Floating Point Format

Within the machine floating point numbers are represented in a form similar to that used in "scientific" notation, that is, a coefficient, or fraction, multiplied by a number raised to a power. Since the machine uses binary numbers only, the numbers are multiplied by powers of 2.

$F*2^E$: F = fraction

E = exponent

$10.5625_{10} = 1010.1001_2$

$.105625*10^2 = .10101001*2^4$

In the 48-bit machine word the fraction is in the lower 36 bit positions. The exponent in the next upper 11 bit positions and the 48th bit position is the sign of the fraction.

48	47 37	36 1
S	Exponent	Fraction

In order to represent positive and negative exponents, the exponent is biased by (A) 10 000 000 000_2 (2000_8) if positive, and by (B) 01 111 111 111_2 (1777_8) if negative. 2000_8 as an exponent means 2^0.

(A)	2001	+1	(B)	1776	-1
	2002	+2		1775	-2
	2003	+3		1774	-3
	-			-	
	-			-	
	-			-	
	3776	+1776		0001	-1776
	3777	$+1777_8$		0000	-1777_8

The range of floating point numbers in the machine is $F*2^{1777_8}$ to $F*2^{-1777_8}$

($F*2^{1023}$ to $F*2^{-1023}$) = ($K*10^{309}$ to $K*10^{-309}$)

The number 10.001_2 would be represented in the machine in floating point form as 2002 420000000000_8. This means the fractional part, converted to binary, $.1000100 \rightarrow 0_2$ is to be multiplied by 2^2, or the binary point is to be moved two places to the right; $10.00100 \rightarrow 0_2$.

To convert a number from decimal to machine floating binary, first convert the number to binary, then move the binary point to the left of the highest order non-zero binary digit keeping account of the number of places.

1. Convert 10.5625_{10} to machine floating binary:

(A) $10_{10} = 1010._2$

(B)

$.5625$		From (A) and (B)
$\times 2$		$10.5625_{10} = 1010.1001_2$
$\underline{1}.1250$	$.1$	
$\times 2$		
$\underline{0}.2500$	$.10$	$= .101010010 * 2^4$
$\times 2$		The machine exponent
$\underline{0}.5000$	$.100$	is 2004_8
$\times 2$		
$\underline{1}.0000$	$.1001$	The fraction as octal is .52200000000
$.5625_{10} =$	$.1001_2$	The floating point machine word is 2004522000000000_8

2. Convert $.125_{10}$ to machine floating point binary:

$.125$		
$\times 2$		
$\underline{0}.250$	$.0$	The machine exponent is 1775_8
$\times 2$		
$\underline{0}.500$	$.00$	The fraction as octal is $.40000000000_8$
$\times 2$		
$\underline{1}.000$	$.001$	The floating point machine word is 1775400000000000_8

$.125_{10} = .001_2$

$= .1 * 2^{-2}$

In the machine a negative number, both for fixed point and floating point, is represented as the complement of the positive number.

.125 in floating point is 1775400000000000_8

-.125, then, would equal 6002377777777777_8

and

10.5625 in floating point is 2004522000000000_8

-10.5625, then, would equal 5773255777777777_8

Arithmetic performed by the computer with operands packed in the floating-point format follows the normal algebraic rules. In addition and subtraction the terms are expressed with like exponents; the coefficients are then added or subtracted to obtain the coefficient of the answer, its exponent being the exponent of the terms.

The product of two floating-point operands is a quantity whose coefficient is the product of the coefficients of the terms, and whose exponent is the sum of the operand exponents. Similarly, a floating-point quotient is a quantity whose coefficient is the quotient of the dividend coefficient divided by the divisor coefficient, with an exponent which is the difference of the exponents of the terms.

The computer performs floating-point arithmetic by separating the exponents from the coefficients and performing the necessary operations on the two portions independently. The results of the two independent actions are assembled to obtain the final answer.

Before the final result is assembled the computer inspects the fraction to determine if it is properly expressed for assembly. The most-significant bit of the fraction must be a "1", if not it is shifted until a "1" appears in this position, the exponent is adjusted to maintain the magnitude of the entire quantity. This operation is the "normalize" routine.

Since the result of a floating-point instruction is expressed within the limits of the A register, the residue of the computation in Q must be inspected to determine if it is great enough to cause the fraction to be increased by one when expressed to the nearest 2^0 position. This is the "round" routine. Both normalize and round are performed automatically during the course of the execution of a floating-point instruction.

Returning to the six instructions at the start of the chapter, let us examine how these use the floating point formats.

FLOATING ADD (FAD) (30.b)(where b may be 0 through 7)

This instruction forms the sum of two 48-bit quantities which are packed in floating-point format. An operand is read from the storage location specified by the sum of the execution address and the contents of the designated index register and is added to the previous contents of the A register. The result is normalized and rounded in the A register at the end of the operation. The Q register contains the residue from the rounding operation at the end of the sequence.

FLOATING SUBTRACT (FSB)(31.b)(where b may be 0 through 7)

This instruction subtracts an operand in floating-point format from the previous contents of the A register, also in floating-point format. The operand is read from the storage location specified by the sum of the execution address and the contents of the designated index register. The result is normalized and rounded in the A register. The residue from the rounding operation is left in the Q register at the end of the sequence.

FLOATING MULTIPLY (FMU) (32.b)(where b may be 0 through 7)

This instruction forms the product of an operand in floating-point format with the previous contents of the A register, also in floating-point format. The operand is read from the storage location specified by the sum of the execution address and the contents of the designated index register. The result is normalized and rounded in A. The residue from the rounding operation is left in the Q register at the end of the sequence.

FLOATING DIVIDE (FDV)(53.b)(where b may be 0 through 7)

This instruction forms the quotient of two 48-bit quantities in floating-point format. The dividend must be loaded into the A register prior to the execution of this instruction. The divisor is read from the storage location specified by the sum of the execution address and the contents of the designated index register. The quotient is normalized and rounded in the A register at the end of the operation. The residue from the rounding operation is left in the Q register at the end of the operation.

The following example indicates how the floating-point add functions:

Example: Add 56_8 and 122_8 in floating point.

56_8 = .101 110. (Exponent = + 6)

floating-point format = 0 10 000 000 110.101 110$\longleftrightarrow$0

octal floating-point format for 56_8 is, 2006.560000000000

122_8 = 00.1010010. (Exponent = +7)

floating-point format is 0 10 000 000 111.1010010$\longleftrightarrow$0

octal floating-point format for 122_8 is, 2007.510000000000

Assume the first of these numbers (56_8) is in the A register, and the second (122_8) is in memory location 00100.

The following instruction is given:

$\boxed{\text{FAD 0 00100}}$

The following addition, normalizing, and rounding takes place: Since the exponents are different, one fraction must be shifted to make the exponents equivalent, before the fractional parts of the two floating-point words are added. The general procedure in this regard is to shift the smaller number right by difference of the two exponents.

In the example, the two exponents (2006 and 2007) are compared and subtracted. The difference is 1. The number with the smaller exponent is then shifted right 1 bit. This is shown below:

56_8 = 2006.560000000000

122_8 = 2007.510000000000

2007 - 2006 = 1 = difference in exponents, the larger exponent is stored (in this example, 2007 is stored). Shift the number of the smaller exponent right 1 bit.

```
The number of the smaller exponent is .560000000000 (in octal)
Shift right 1, gives                   .270000000000 (in octal)
                                       .270000000000
Adding the two numbers                 .510000000000
                                      ______________
Octal sum of 2 fractional number is   1.000000000000
```

This sum must be normalized, which theoretically means moving the binary point to the left of the most-significant "1". In the example it is necessary to move the binary point one place to the left which means the stored exponent (2007) must be increased by 1, the number of shifts required.

Shifted octal sum is .400000000000

And adding 1 to the stored exponent gives: 2007+1 = 2010

The sum is then rounded by inspecting the bit immediately to the right of the 36 bits of significance in the sum. If there is a "1" the 36th bit is changed; otherwise, the rounding has no effect. In the example, the rounding does not change the sum. (In machine rounding provision must be made for the case when rounding might possibly develop a carry all the way through the number.)

The final result in A is 2010.400000000000

Check: $56_8 + 122_8 = 200_8 = 2^8\ (.4_8) = 128_{10}$

The preceding example indicates the large number of internal sequences required to execute a floating-point instruction. Although the machine process is more complex, the example indicates some of the problems the machine must solve inorder to carry through a floating-point instruction.

The last two instructions of this chapter are explained with the floating-point instructions since they involve scaling and they enable the programmer to simulate the normalizing process which takes place automatically in the floating-point processes.

SCALE A (SCA) (34.b) (where b may be 0 through 7)

This instruction shifts the quantity in the A register circularly to the left until the most-significant "1" is immediately to the right of the sign bit. The base execution address is reduced by the number of bit positions shifted. The shift is terminated if the base execution address becomes zero before the normalizing is completed. In any event, the reduced base execution address is then entered into the designated index register.

SCALE AQ (SCQ) (35.b)(where b may be 0 through 7)

This instruction shifts the quantity in the AQ register circularly to the left until the most-significant "1" is immediately to the right of the sign bit. The base execution address is reduced by the number of bit positions shifted. The shift is terminated if the base execution address becomes zero before the normalizing operation is completed. In any event, the reduced execution address is then entered into the designated index register.

Either of these instructions can be used to indicate the scaling of numbers or results of arithmetic processes. Each instruction is so executed that the number of shifts to normalize is left in the designated index register. As shown in later examples this quantity is used to determine the exponent of a floating-point number.

As an example of the use of one of these instructions, the SCALE A instruction is used to indicate the scaling of the number in the following example.

Example: The quantity 402_8 is in storage location 00100. Normalize and show the contents of the designated index register after the quantity is normalized.

Given: Storage location 00100 contains:

$$\overbrace{0 \longleftrightarrow 0}^{39\text{ bits}} \quad \overbrace{100\ 000\ 010}^{9\text{ bits}} \text{ (in binary)}$$

After loading this into A, the SCALE A instruction is executed.

Instructions are:

LDA 0 00100 SCA 3 00057

The SCALE A instruction shifts A left, and reduces the execution address (00057) by 1 on each left shift. The shift is terminated if the execution address becomes zero before the normalizing is completed. The reduced execution address is then entered into the designated index register (3 above).

In the example, there will be 38_{10} shifts required to put the contents of 00100 in normalized form as shown below:

$$010\ 000\ 001\ 00 \overset{38\text{ bits}}{\longleftrightarrow} 0 \text{ (normalized)}$$

Since 38_{10} shifts were required, the execution address (0057_8) is reduced by $38_{10} = 46_8$ $00057_8 - 00046_8 = 00011_8$. This reduced amount is automatically entered into the index register 3. Thus at the end of the execution, index register 3 contains:

$$\overbrace{0 \longleftrightarrow 001001}^{15 \text{ bits}} \qquad \text{(in binary)}$$

It should be noted that this reduced amount is exactly equivalent to the true floating-point exponent of the base 2.

$402_8 = .100\ 000\ 010.$

9 shifts

Thus $402_8 - 2^9\ (0.402_8)$

The above example indicates a method which can be used by the programmer to pack binary numbers into the floating-point format. This chapter is concluded with an example "packing routine" for positive whole numbers with positive exponents.

The following routine packs a positive binary whole number with a positive exponent in floating-point format and leaves it in Q.

(C)	=	LDA 0 SL	AJP 3 NEG.	Load first word into A , if negative, jump.
(C+1)	=	SCA 1 *2057	ALS 0 1	Normalize, then shift left 1.(see note below)
(C+2)	=	STA 0 T	ENA 1 0	Store normalized number, enter floating point exponent into A.
(C+3)	=	LDQ 0 T	LRS 0 14	Load normalized number into Q, shift to floating point position in Q.

*Note by using 2057 in the execution address part of the SCALE A instruction, the sign of the exponent is contained along with the exponent after the normalizing has taken place. Thus in the above example, if the normalizing involved 50_8 shifts, then

$$2057_8 - 50_8 = 2007 = \underset{\text{sign of number}}{010}\ \underbrace{000\ 000\ 111}_{\text{exponent}} \qquad \text{(in binary)}$$

EXERCISES 13.

a. Change the following to 1604 floating-point format

(1) 31_8
(2) -156_{10}
(3) 0.4_8
(4) -0.5028

b. Change the following floating-point numbers to octal equivalents

(1) 2010.612000000000

(2) 5772.317777777777

c. Assume S1 contains a number which can be expressed in 1604 floating-point format as a positive number with positive exponent. Write a program which will "pack this" number in 1604 floating-point format.

CHAPTER VII

SIXTH GROUP OF INSTRUCTIONS

A description of the input-output section is a prerequisite to a discussion of Group Six instructions. Communication of data between the computer and external equipment is carried out in either of two modes of operation. The first and most frequently used method is buffering. The most important feature of buffer communication is that individual words are exchanged independently from the main computer program. Once the main program prepares for and initiates a buffer operation, no further main-program attention is required. Computation continues and is suspended for brief periods whenever individual words are taken from or entered into the storage section of the computer.

The second method of communication is the high-speed transfer, used in exchanging data between computers or other special external equipment. Transfer communication is accomplished under main program control, and thus computation may not continue independently during the transfer of words, as was the case for buffering. For this reason, the transfer instructions are used only with equipment capable of handling data at a very rapid rate.

There are six 48-bit channels for use in buffering. Channels 1, 3, and 5 are for input, while channels 2, 4, and 6 are for output. The high-speed transfer uses channel 7 for both input and output. The external equipment at the 1604 console (reader, punch, and typewriter) are connected to buffer channels 1 and 2. Channels 3, 4, 5, and 6 may provide connection to a 1604 magnetic tape system and to the 1605 adaptor and thereby to several external IBM equipments or other peripheral devices.

Buffer operations involve the manipulation of blocks of data, where a block may range from one word to several thousand. The size of a block is determined by specifying the initial and terminal storage address + 1 which is to be used.

Each buffer channel operates asynchronous with the computer program and the other channels. All channels are treated with equal priority by a scanner which sweeps sequentially through all buffer controls every 3.2 microseconds. If a channel requires processing, the scanner stops while the word is transferred and then resumes scanning to the next channel. In the event that all channels should simultaneously request action the last would be processed in a maximum of 240 microseconds. This time limits the rate of communications to a 5 KC word rate per channel if all channels are used simultaneously.

The detail sequencing of a buffer operation is coordinated by a special buffer control section in the central computer. The buffer control utilizes a special address in storage for each communication channel. These special addresses (00001 - 00006) contain the current storage address for buffering data and a terminal address for stopping the transfer at the end of the block of data. Only the upper and lower address portions of these special storage locations are treated as significant by the buffer control. The terminal address, which is one greater than the address of the last word transferred, must be entered in the lower address position of the control word prior to the initiation of the buffer action. This entry may be made by either a substitute lower address (61) or by a simple store instruction since the quantity appears in the lowest 15 bits of the word in storage.

The storage address of the first word in the block to be buffered is entered into the appropriate control word by the external function instruction (74) which initiates the buffer operation. This address is entered much like a substitute upper address instruction (60) and does not alter the other bits in the control word.

Each word transferred during a buffer operation requires the execution of three storage references. The first reference reads the appropriate control word from the special storage address and extracts from the upper address position from the current storage location for the word data being buffered. The address is then used for the second storage reference which transfers the data to (or from) the central storage unit. A third reference is then made to replace in the upper address position the current buffer address with its initial value plus one. The current and terminal addresses are compared and if they are equal the buffer operation is terminated.

The status of a buffer operation may be monitored by the computer program during the buffer operation. This is accomplished by a normal reference to the appropriate control word and a comparison of the current buffer address with the terminal address. The buffer operation is completed when the current address is equal to the terminal address.

SIXTH GROUP OF INSTRUCTIONS (With Mnemonic and Numeric Codes)(where "b" represents the index designator which can be of any number 0 through 7)

EXTERNAL FUNCTION	(EXF.b)	(74.b)
INPUT TRANSFER	(INT.b)	(62.b)
OUTPUT TRANSFER	(OUT.b)	(63.b)

These instructions are used to transfer information into the computer from some external equipment or out of the computer to some external equipment. The first instruction is a buffer instruction and is used with most of the input-output equipment. The last two instructions utilize a special communication channel and are provided primarily to facilitate the high-speed transfer of information betwen the memories of two or more 1604 computers or other similar data transfer requirement.

EXTERNAL FUNCTION (EXF.b)(74.b)(where b may be 0 through 7)

This instruction has eight sub-instructions which are used to control the transfer of information between the computer and input-output equipments. The index registers are not used for address modification in this instruction. The index designator is used to specify which one of the eight sub-instructions is to be performed.

The sub-instructions and the operation performed by each are:

74.0	Select external equipment
74.1	Activate communication channel one
74.2	Activate communication channel two
74.3	Activate communication channel three
74.4	Activate communication channel four
74.5	Activate communication channel five
74.6	Activate communication channel six
74.7	Sense external condition

Execution of a 74.1 through 74.6 sub-instruction initiates the buffering of a block of data between the computer and a previously selected external equipment. The base execution address of such an instruction designates the starting address in computer storage. Thus, if the current instruction is 74 2 05000, an output buffer is initiated on channel 2. The first word to be sent out is taken from address 05000. During the execution of 74.2, the initial address for the buffer, in this example 05000, is stored at the upper address position of storage address 00002, the terminal address plus one having previously been entered into the lower address position at 00002.

Thus if 100 (octal) words are to be buffered out, beginning at address 05000, then the terminal address is 05100. This address quantity, 05100, was entered in the lower position at 00002 by a 61 instruction that has been executed prior to the 74.2. As a result the last word to be buffered is taken from address 05077.

A 74.0, or external select instruction, provides for transmitting control information or codes to the control section of an external equipment. Such codes cause the equipment to prepare for the transfer of data to or from the computer. The computer may sense the status of the external equipment by means of the 74.7 external sense instruction. For both the 74.0 and the 74.7 instructions the base execution address is used to specify the channel, equipment, and condition.

The interpretation of the various digits of the execution address for a 74.0 or 74.7 instruction is shown in Figure 16. The upper three bits are interpreted in the computer to determine the channel that is to receive the remaining 12 bits. The latter bits are called the "external select" code for 74.0 and the "external sense" code for 74.7. The various select and sense codes used with external equipment appear at the end of this chapter.

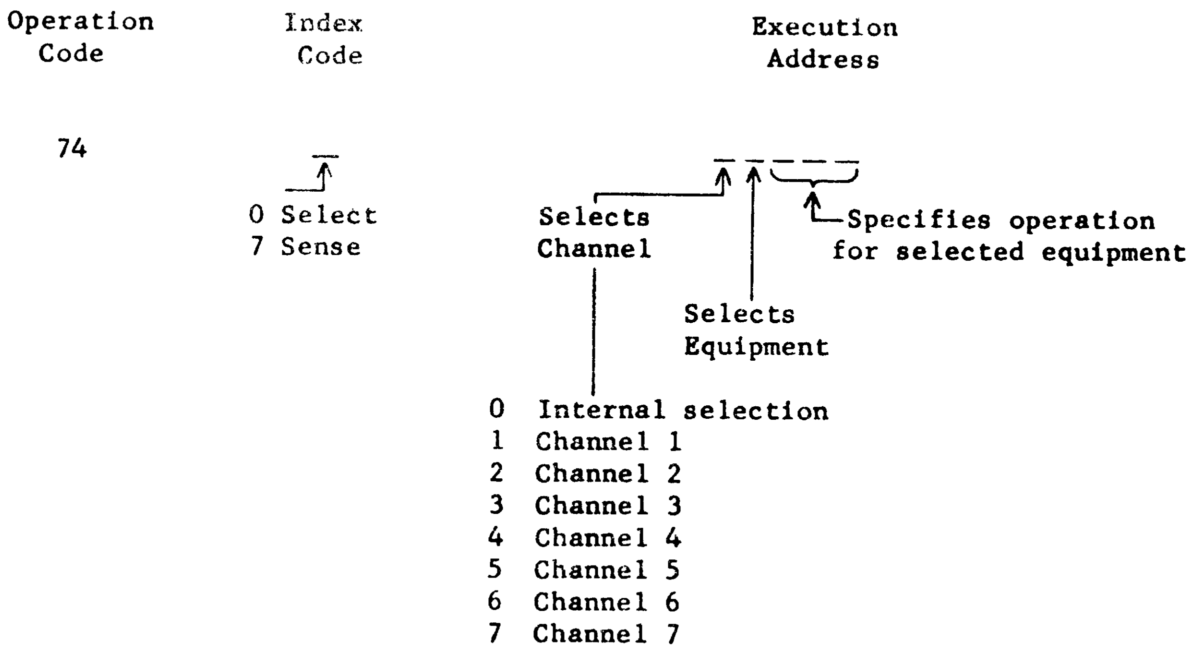


Figure 16. Interpretation of Octal Digits in External Function Instruction

The case of a 74.0 or 74.7 instruction when the upper octal digit of the execution address is 0, is a special one. This involves no communication with external equipment, but instead provides for examining computer fault conditions. An example of 74.0 external select instruction is the following

74 0 11200

Execution of this instruction sends the code 1200 (the lower four octal digits of the execution address) to the equipment connected to channel 1 at the 1604 console. The code 1200 selects the paper tape reader which is connected to channel 1. As a result, the reader is prepared to begin operation as soon as a 74.1 instruction (activate communication channel 1) is executed.

The 74.7 instruction, which senses a computer fault or a condition in an external equipment, is always used as the upper instruction. This follows from the fact that the lower instruction is skipped on the basis of the results of sensing the presence or absence of the specified condition. For each specified condition two sense codes are provided. They are identical except for the lowest digit. The code with an even digit in the lowest position senses the presence of the condition, that is, a positive response results when the condition does not exist. The code with an odd digit in the lowest position senses the absence of the condition and a positive response results. In either case a positive response causes the next, that is, the lower, instruction to be skipped.

A typical use of the 74.7 instruction is in sensing whether a tape unit in the 1604 magnetic tape system is ready to read. The instruction format is as follows

74 7 320j0

Execution of this instruction sends the code 20j0 to tape unit j of the 1604 tape system. If this tape unit is ready to read, a positive response is returned to the computer. As a result, the next, that is, the lower, instruction is skipped. If the tape unit is not ready to read, a negative response is sent back and the computer executes the lower instruction.

An alternative method for sensing the same condition uses the following format:

74 7 320j1

The difference in the lowest digit means that if tape unit j is ready to read, a positive response is sent back. If it is not ready a negative response is sent to the computer.

The 74.7 instruction with an even sense code is quite typically used with a jump as the lower instruction. The jump would enter a routine for handling the specified condition. The jump instruction is skipped unless the condition sensed by the 74.7 is present. Such a use of an even external sense code and a jump as the lower instruction is convenient for cases where several conditions are to be sensed in a successive manner. The 74.7 with an add sense code affords another option which is useful in other circumstances.

The individual external equipments interpret the equipment selection digit in an external function. Each equipment recognizes its own code. The remaining nine bits of the external control cable are then sampled by the selected external device. The external equipment is continually monitoring these control lines. No resume signal is sent back to the computer indicating receipt of the information. However, an external sense code causes the equipment to produce a responding signal on the sense return line when the condition specified by the code is present.

INPUT TRANSFER (INT.b)(62.b)(where b may be 0 through 7)

This instruction transfers a block of data from an external equipment into the central computer storage. This transfer utilizes a special communication channel of 48 lines and is intended for a very rapid transfer of data between the memories of two or more 1604 computers or other similar data transfer situations. The number of words transferred is specified by the contents of the designated index register. The words are stored in consecutive locations beginning at the location specified by the base execution address.

The first word received is stored at the last address. For each word transferred the designated index register is reduced by one. Thus the index register is cleared at the end of the transfer. If the index code is zero, one word is transferred. If the contents of the designated index register are zero, no transfer takes place.

OUTPUT TRANSFER (OUT.b)(63.b)(where b may be 0 through 7)

This instruction transfers a block of data from the central computer storage to an external equipment. This transfer utilizes a special communication channel of 48 lines and is intended for very rapid transfer of data between the memories of two or more 1604 computers or other similar data transfer situations. The number of words transferred is specified by the content of the designated index register. The words are located in a consecutive list beginning at the location specified by the base execution address. The first word transferred out is obtained from the last address. For each word transferred the designated index register is reduced by one. Thus the index register is cleared at the end of the transfer. If the index code is zero, one word is transferred. If the contents of the designated index register are zero, no transfer takes place.

EXAMPLE

Write a program which will load 100 (octal) words from paper tape and store it in consecutive addresses beginning at address 200. Assume that words are to be loaded in the assembly mode. Begin program at address 00050. (The external select and sense codes used with 74.0 and 74.7 sub-instructions appear at the end of the chapter.)

```
00050    74 7 11220        75000050

00051    10 0 00300        20000001

00052    74 7 11201        76000052

00053    74 0 11200        74100200

00054    74 7 11220        75000054

00055    76 0 00050        00000000
```

EXPLANATION.

If this is a routine in the middle of a program, then at the time the routine is to be executed some piece of input-output equipment on channel 1 may be receiving information; so the first program step tests channel 1 to ensure that information is not being buffered at this time. If information is being buffered an unconditional jump back to the sense instruction is executed, and this step will be repeated until all the information has been buffered. When this happens, the lower portion of step 00050 is skipped and step 00051 is executed. This step loads A with the last address plus one at which the information will be stored, and the lower instruction of step 00051 stores this address in the lower portion of address 00001.

Step 00052 then senses the paper tape reader to ensure that the reader is in the assembly mode. If it is not, the lower portion of Step 00052 is executed, which is an unconditional stop so that the operator may throw the switch which will put the reader in the assembly mode. If the reader is already in the assembly mode the stop instruction is skipped and step 00053 is executed.

The upper portion of step 00053 selects the paper tape reader. The lower portion then selects the channel by means of the index designator and the first address at which to store the incoming information by means of the base execution address. These are all the steps that are necessary to activate the buffer. One could now do computation or set up a buffer on a different channel. However, in this case we chose to simply load in the words and stop the computer. Since the buffering action will cease if the computer is stopped, we must test to determine when all 100 (octal) words have been loaded. This is accomplished by the same method as explained in the beginning of this example. Once the end of the buffer has been sensed, the program executes step 00055 wh'ch stops the computer.

EXAMPLE

Using the high speed transfer instruction, write the step for transferring 50 (octal) words from an external equipment to memory. Store the last word at address 00300. Where will the first word be stored?

```
     01000              50 1 00050          62 1 00300
```

EXPLANATION

Index register one is loaded with the number of words to be transferred. The first word will be stored in the address which is the sum of the base execution address of the input transfer instructions and the content of the designated index register. However, since the index register is reduced one before this transfer takes place, the first word will be stored at address 00347, the next word at 00346, etc. The last word will be transferred when the index register one equals zero so that it will be stored at address 00300.

USE OF 74.7 INSTRUCTION

The 74.7 external function instruction may be used in either the upper or lower positions. When used in the upper position, a 74.7 is a skip instruction. That is, the lower instruction is skipped if the condition given by the EF code is present, but the lower instruction is executed if the condition given by the EF code is not present. In the first case, the 74.7 "exits" to the next pair of instructions. In the second case, the 74.7 "half exits" to the lower instruction.

When the 74.7 is used in the lower position, it is not a skip instruction. Instead the 74.7 is simply executed repeatedly until the condition given by the EF code occurs. At this time the 74.7 exits to the next pair of instructions. Until the condition given by the EF code is present, the 74.7 simply half exits to repeat itself. A 74.7 in the lower position is therefore a means of awaiting the occurrence of a specified condition.

Throughout the attached list of sense codes, the term "exit" and "half exit" are used with the meaning indicated above. An exit is performed if the stated condition is present; if not present, a half exit is performed.

Any 1xxxx select code clears all previous selections on channel 1. Likewise, any 2xxxx select code clears all previous selections on channel 2.

EXTERNAL FUNCTION CODES

The attached lists present external function codes for the following:

CODE	CONDITION
0xxxx	Internal
1xxxx	Channel 1
2xxxx	Channel 2

Internal Select

0x010	- Select interrupt on channel 1 inactive
0x011	- Remove selection
0x020	- Select interrupt on channel 2 inactive
0x021	- Remove selection
0x030	- Select interrupt on channel 3 inactive
0x031	- Remove selection
0x040	- Select interrupt on channel 4 inactive
0x041	- Remove selection
0x050	- Select interrupt on channel 5 inactive
0x051	- Remove selection
0x060	- Select interrupt on channel 6 inactive
0x061	- Remove selection
0x07x	- Clear arithmetic faults
0x100	- Select interrupt on arithmetic fault
0x101	- Remove selection
01xxx	- Start clock
02xxx	- Stop clock

Internal Sense

0x010	Channel 1 active
0x011	Channel 1 inactive
0x020	Channel 2 active
0x021	Channel 2 inactive
0x030	Channel 3 active
0x031	Channel 3 inactive
0x040	Channel 4 active
0x041	Channel 4 inactive
0x050	Channel 5 active
0x051	Channel 5 inactive
0x060	Channel 6 active
0x061	Channel 6 inactive
0x110	Divide fault
0x111	No divide fault
0x120	Shift fault
0x121	No shift fault
0x130	Overflow fault
0x131	No overflow fault
0x140	Exponent fault
0x141	No exponent fault

Channel 1 Select

1110x	Keyboard entry
1114x	Keyboard entry and interrupt on carriage return
1120x	PT reader
1121x	PT reader and end-of-tape indicator
1122x	PT reader and interrupt on end-of-tape

Channel 2 Select

2110x	Print assembly mode
2111x	Print character mode
2120x	Punch assembly mode
2121x	Punch character mode
2124x	Turn punch motor off

Channel 1 Sense

11100	Keyboard carriage return
11101	No keyboard carriage return
11140	Keyboard lower case
11141	Keyboard upper case
11200	PT reader, end-of-tape
11201	PT reader, no end-of-tape
11210	PT reader, assembly mode
11211	PT reader, character mode

Channel 2 Sense

212x0	Punch, end-of-tape
212x1	Punch, no end-of-tape

APPENDICES

APPENDIX A

RESULTS OF EXERCISES

EXERCISES I (solutions)

a. $\boxed{\text{QRS 0 \quad 0 0 0 1 6}}$

Alternative Solution:
An Index designation other than zero may be used if the octal sum of the contents of the designated index register and the execution address equals the desired shift count.

b. 0 0 0 0 0 2 2 0   0 0 0 0 0 4 3 6

c. $\boxed{\text{QRS 0 \quad 0 0 0 0 7}}\boxed{\text{QLS 0 \quad 0 0 0 1 4}}$ (See alternative solution above)

d. $\boxed{\text{ALS 0 \quad 0 0 0 5 3}}$ (See alternative solution above)

since no bits are to be discarded, one must use equivalent left shift. (48 - 5 = 43 decimal shifts required.)

e. No. Since the end-off shift discards some bits, the two results can not be the same.

EXERCISES 2 (solutions)

a. This is the LOAD A instruction. Since index register 3 is designated, the instruction will load the contents of address 00015 (00010 + 00005) into the A register. This address is the sum of the execution address (00010) and the contents of index register 3 (00005).

b.

LDQ 0	0 1 0 0 0	STQ 0	0 1 0 0 1

The upper instruction loads the contents of address 01000 into Q. The lower instruction then stores the contents of Q at location 01C01. (Alternative solutions are possible by using index registers whose contents added to the contents of other execution addresses would give the execution addresses above.) Since the A register was not to be disturbed, the Q instructions were used.

c. The instruction given STA 2 0 0 0 2 0 stores the contents of the A register in a location equivalent to the sum of the execution address (00020, above) and the contents of the designated index register (2). Since index register 2 contains 77767, and execution address is 00020, the location of the operand is equal to the sum below.

```
address  (00020)          00020
contents of index
   register 2             77767
                        -------
one's complement     1    00007
with end-around
carry in this                 1
case, to give ------    -------
                          00010
```

Therefore the given instruction will store the contents of the A register in address 00010.

d. When LOAD instructions are given, the initial contents of A or Q are destroyed since this register is cleared before the LOAD operation is executed.

When STORE instructions are given, the initial contents of A or Q are not disturbed by the execution of the instruction.

EXERCISES 3 (solutions)

a.

ADD 0	0 0 0 3 0

b.

(K) =	SUB 0	0 0 0 4 1	SUB 0	0 0 0 4 2
(K+1) =	ADD 0	0 0 0 4 3	ADD 0	0 0 0 4 4

c. Yes.

00100	AJP 1	0 0 1 0 1	QJP 0	0 0 0 5 0

d.

ADD 5	0 0 0 0 0	AJP 1	0 0 1 0 5

e. The upper instruction is a Return Jump since the index designator 6 calls for a Return Jump if A contents are positive. The location of the Return Jump instruction (00304) is increased by 1 (to 00305) and this goes to the execution address portion of the upper instruction at step 00010. A jump then occurs to the lower instruction of step 00010.

EXERCISES 4 (solutions)

a. INITIAL CONTENTS of A and Q were given as:

```
                A Register                          Q Register
     7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 0   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3
                                     (Shift A and Q Right 4)
Instruction Required -------  | LRS 0   0 0 0 0 4 |

                         A Register            Q Register
Final Contents  )
of A and Q      )     7777777777777777      4000000000000000
```

b. INITIAL CONTENTS of A and Q were given as:

```
            A Register                       Q Register
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
                                 (LLS.0)  (Shift A and Q left 24₁₀ places)
Instruction Required ———> 0 7 0   0 0 0 3 0
                                          Note 24₁₀ = 30 (octal)
```

Final Contents
of A and Q

```
            A Register                       Q Register
     0000000077777777                 7777777700000000
```

c. Given: Index Register 5 contains 0 1 3 2 6
Address 01111 contains 0000000000007562

(1.) $\boxed{\text{ENA 5} \quad 0\ 0\ 0\ 0\ 0}$ After this ENTER A instruction, the contents of A will be:

0 0 0 0 0 0 0 0 0 0 0 0 1 3 2 6

(2.) $\boxed{\text{ENA 5} \quad 0\ 0\ 1\ 0\ 0}$ After this ENTER A instruction, the contents of A will be:

0 0 0 0 0 0 0 0 0 0 0 0 1 4 2 6

d.
1. TRUE (Two instructions would be needed)
2. TRUE (Unless overflow takes place)
3. FALSE (One uses contents of execution address, other uses execution address as operand)

4. FALSE (It can just as well send a negative to Q)

e.
1. After $\boxed{0\ 4\ 0 \quad 0\ 0\ 2\ 0\ 0}$ Q will contain:
0000000000000200

2. After $\boxed{1\ 7\ 0 \quad 0\ 0\ 2\ 0\ 0}$ Q will contain:
7777777777770000

3. After $\boxed{1\ 6\ 0 \quad 0\ 0\ 2\ 0\ 0}$ Q will contain:
0000000000007777

EXERCISES 5. (solutions)

a. Add the execution address 00761
and the contents of index register 6. 00017
Extend sign bits of sum → 01000

```
Add the execution address . . . . . .      00761
and the contents of index register 6.      00017
Extend sign bits of sum  ----------------> 01000

                    <-0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

"Increase A" by this amount
                      7 7 7 7 7 7 7 7 7 7 7 7 0 1 3 5
                      -------------------------------
Contents of A will be 7 7 7 7 7 7 7 7 7 7 7 7 1 1 3 5
```

b. (00100)

ENA 2	0 0 0 0 0	MUI 0	0 0 0 5 0

↑Code for Enter A ↑Code for Integer Multiply

First enter contents of index register 2 into A (the multiplier).
Then multiply using contents of 00050 as multiplicand.

c. 1. False (Final content of A is part of the product.)

2. True (Otherwise one would not know the multiplier)

3. False (Product of two 48 bit factors cannot overflow 96 bits)

d. The dividend is contained in Q and A and is given:

Q Register	A Register
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0

The given instruction divides the contents of 00100 into this.
Address 00100 contains 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3.

After the divide, the following quantities will be left in the A and Q registers and address 00100.

FINAL CONTENTS

```
              Q Register                                A Register
              0000000000000001                          0000000000000005
                    ↑                                         ↑
Sign of       Remainder                   Sign of          Quotient
Dividend                                  Quotient
Address 00100    0000000000000003 (unchanged)
```

e.	(00040) =	INA 2	0 0 0 0 0	ADD 5	0 0 0 0 0

The first instruction "increases A" by the contents of index register 2. The second instruction adds to A the contents of the address located at index register 5. Note the execution addresses above are each zero which reduces each instruction to a function of the contents of the index registers involved. However, INCREASE A uses the contents of the index register as the operand; whereas, ADD uses the contents of the index register as the address of the operand.

EXERCISES 6. (solutions)

a.	(00020) =	SLJ 4	0 0 0 7 2	Some other instruction
	(00072) =	SLJ 0		First Instr. of Sub-Routine
	(00105) =	Last Instr. of Sub-Routine		SLJ 0 0 0 0 7 2

Explanation: The SELECTIVE JUMP with index designation 4 is an Unconditional Return Jump which stores contents of Program Address Register plus one (00021) in the upper instruction at 00072 and the jump goes to the lower instruction at 00072. The exit at 00105 jumps back to 00072 whence another jump goes back to 00021.

b.	(00112) =	LDA 0	0 0 0 2 3	SAU 0	0 0 0 2 2
	(00113) =	ALS 0	0 0 0 1 1	SAL 0	0 0 0 2 2

The first instruction LOADS A with 0000000000000001. The second instruction SUBSTITUTES the low order 15 bits of A (00001) into the execution address portion of the upper side of 00022. Then A is shifted left nine places to become 0000000000001000. Again the low order 15 bits of A (01000) are SUBSTITUTED into the address portion of the lower side of 00022.

c.

(00417) =	ENA 5	0 0 0 0 0	AJP 2	0 0 4 2 1
(00420) =	ADD 0	0 0 1 0 0	SLJ 0	0 0 7 0 0
(00421) =	SAL 0	0 0 0 4 7	SLJ 2	0 0 7 0 0

Contents of Index Register 5 are first sent to A. The A register is tested for positive contents. If positive jump goes to 00421; if not to 00420. At 00420 contents of 00100 are added to A and jump goes to 00700. At (00421), the contents of A are stored at 00047 and jump goes to 00700 if Jump Key 2 is set.

d.

(00050) =	ENA 3	0 0 0 0 0	ADD 0	0 0 0 0 3
(00051) =	STA 0	0 0 0 0 3	SLJ 3	0 0 0 3 0

e.

(00032) =	SLJ 1	0 0 1 1 2	Some Other Instr.
(00112) =	SLJ 2	0 0 2 0 0	Some Other Instr.

If neither key is set, it will perform TASK G and TASK H. If Jump Key 1 is set and Jump Key 2 not set, it will skip TASK G and do TASK H. If Jump Key 1 is not set and Jump Key 2 is set, it will do TASK G and skip TASK H. If Jump Key 1 is set and Jump Key 2 is set, it will skip both tasks.

EXERCISES 7. (solutions)

a.

(00032) =	SSK 0	0 0 0 5 0	SLJ 0	0 0 1 1 0
(00033) =	SLJ 0	0 0 2 0 0	Some Other Instr.	

b. Given Jump Key 2 not set

(00052) =	SSK 0	0 0 1 0 0	SLJ 2	0 0 2 0 0

The first instruction tests the contents of 00100 for sign. If negative, the next instruction (7 5 2 0 0 2 0 0) is skipped. If positive, the next instruction is interpreted but since Jump Key 2 is not set, thejump will not occur to 00200. Therefore, in either case the instruction 7 5 2 0 0 2 0 0 is skipped until Jump Key 2 is set on the console.

Load A w/contents of 00040

Shift A Right 1 bit place to put last bit in left-most position

c.

(00013) =	LDA 0	0 0 0 4 0	LRS 0	0 0 0 0 1
			Pass	
(00014) =	STQ 0	0 0 0 5 0	ENI 0	0 0 0 0 0
(00015) =	SSH 0	0 0 0 5 0	SLJ 0	0 0 2 0 0
(00016) =	Continue			

The STORAGE SHIFT at step (00015) tests the sign of the last bit of the unknown digit. If positive (even in this case) the next instruction jumps to 00200. If negative (odd in this case) the instruction at 00016 is the next step to be performed.

d.

(c) =	ISK 5	0 0 1 4 4	SLJ 0	0 0 0 5 0
(c+1) =	SLJ 0	0 0 2 0 0	Some Other Instr.	

The first instruction tests the contents of index register 5 against the number in the execution address (100_{10}). If not equal, the jump to 00050 occurs. If equal, the next instruction is skipped and the jump to 00200 occurs.

THIS PAGE HAS BEEN DELETED.

EXERCISES 8 (solutions)

a.

		Load Q with extractor	Load A with logical product of Q and contents of 01012
(c)	=	LDQ 0 0 0 0 5 4	LDL 0 0 1 0 1 2
		Jump if content of A is zero	Enter zeros into A register
(c+1)	=	AJP 0 (c+k)	ENA 0 0 0 0 0 0
		Store the zeros in A in 01012	
(c+2)	=	STA 0 0 1 0 1 2	

(Alternative solutions are possible)

Extractor at 00054 40 40 40 40 40 40 40 40

b.

```
Given:         (A)     =  1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
               (00111) =  4 5 3 0 0 0 0 0 2 0 0 0 0 1 0 0
               (Q)     =  0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0

Index Reg. 3        =     0 0 1 1 1
```

Final Contents of A $\longrightarrow$ $\boxed{1\ 2\ 3\ 0\ 0\ 0\ 0\ 0\quad 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}$

Explanation: The given instruction at (00111) forms a logical product of the operand located at the sum of the execution address (00000) and the contents of index register 3 (00111), and the contents of Q. Thus the logical product formation takes place between contents of Q and contents of 00111. This gives the result 0030000000000. When this is added to A, the final result is obtained. The second instruction in the step does not affect the contents of A - it stores A at 00100.

c. Given (00712) contains X_1 X_2 X_3 . . . . X_{16} where each X is an octal digit of 3 bits. The problem is to reduce this quantity by the sum of X_1 and X_{16}.

	Load A	Load Q with Extractor
(c) =	LDA 0 0 0 7 1 2	LDQ 0 0 0 0 5 0
		where (00050) = 0000000000000007
	Store Log. Product of Q and A	Shift A Register left 3 places
(c+1) =	STL 0 0 0 0 5 1	ALS 0 0 0 0 0 3
		00051 now contains 000000000000000X 16
	Store Log. Product of Q and A	Reload A with Original Quantity
(c+2) =	STL 0 0 0 0 5 2	LDA 0 0 0 7 1 2
		00052 now contains 000000000000000X_1
	Subtract X_{16}	Subtract X_1
(c+3) =	SUB 0 0 0 0 5 1	SUB 0 0 0 0 5 2
	Store Reduced Amount in 00712	
(c+4) =	STA 0 0 0 7 1 2	Continue

d. (1) Yes.

(2) No. A bit-by-bit product cannot exceed the number of bits in each of the factors.

(3) No. Since the positive factor will generate a positive bit in sign of product regardless of other factor.

EXERCISES 9. (solutions)

a. Given: Zone rates in index registers 1 through 4. Flat rate in index register 6.

	Store index register 1	Store index register 2
(00100) =	SIL 1 0 0 1 0 2	SIU 2 0 0 1 0 3
	Store index register 3	Store index register 4
(00101) =	SIL 3 0 0 1 0 3	SIU 4 0 0 1 0 4
	Store index register 6	Enter index register 5
(00102) =	SIL 6 0 0 1 0 4	ENI 5 (0 0 0 0 0)
	Increase index register 5	Increase index register 5
(00103) =	INI 5 (0 0 0 0 0)	INI 5 (0 0 0 0 0)
	Increase index register 5	Increase index register 5
(00104) =	INI 5 (0 0 0 0 0)	INI 5 (0 0 0 0 0)
(00105) =	Continue Program	

Explanation: The first five instructions store the contents of index register 1, 2, 3, 4, and 6, respectively, in consecutive instructions starting with the lower instruction of step 0 0 1 0 2. The next instruction enters index register 5 with the contents of index register 1. The last four instructions increase the contents of register 5 by the contents of index registers 2, 3, 4, and 6, respectively. The desired rate is then found in index register 5.

b.

Clear A

(00154)=	ENA 0 0 0 0 0 0	ENI 0 0 0 0 0 0
(00155)=	ENQ 1 0 0 0 0 0	QJP 7 0 0 1 6 4
(00156)=	ENQ 2 0 0 0 0 0	QJP 7 0 0 1 6 4
(00157)=	ENQ 3 0 0 0 0 0	QJP 7 0 0 1 6 4
(00160)=	ENQ 4 0 0 0 0 0	QJP 7 0 0 1 6 4
(00161)=	ENQ 5 0 0 0 0 0	QJP 7 0 0 1 6 4
(00162)=	ENQ 6 0 0 0 0 0	QJP 7 0 0 1 6 4
(00163)=	SLJ 0 0 0 2 0 0	ENI 0 0 0 0 0 0
(00164)=	SLJ 0 (0 0 0 0 0)	INA 0 0 0 0 0 1
(00165)=	SLJ 0 0 0 1 6 4	

Explanation: The A-register is cleared to use as a counter. The contents of each index register are entered into the Q-register. Q is then tested to see if it contains a negative number. If Q is negative, a return jump is made to a subroutine which adds one to the A-register (counter). After all index registers have been tested, the A-register will contain the number of negative index registers.

c.

		Enter Index register 2 into (A)	Shift (A) Left 39_{10} bit positions
(c)	=	ENA 2 0 0 0 0 0	ALS 0 0 0 0 4 7
		Add Instr. at 00026	Store Modified Instruction at 00026
(c+1)	=	ADD 0 0 0 0 2 6	STA 0 0 0 0 2 6

Explanation: The first instruction enters the contents of index register 2 into (A). (A) now appears as:

$$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \underbrace{0\ 0\ 0\ 0\ X}\ \left\{\begin{array}{l}\text{These octal digits}\\ \text{come from index}\\ \text{register 2}\end{array}\right.$$

(A) is now shifted left 13 (octal) or 39_{10} bit positions.
(A) now contains

0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0

The contents of (00026) are then added to A which modifies the upper instruction by adding the digit (X) to the operation code bits (left nine bits) of the upper instruction. The result is then stored in 00026.

d.

	Enter Index Register 6 in (A)	Increases (A) by contents of Index Register 4
(c) =	ENA 6 0 0 0 0 0	INA 4 0 0 0 0 0
	Store (Q) in 00050	Add contents of 00050 (which=Q) to contents of (A) Register
(c+1) =	STQ 0 0 0 0 5 0	ADD 0 0 0 0 5 0
	Store (A) at 00050	Loads Index Register 6 with Lower 5 Octal Digits of (00050)
(c+2) =	STA 0 0 0 0 5 0	LIL 6 0 0 0 5 0

Explanation: The first instruction enters contents of index register 6 into the A register. The second instruction INCREASES A by the contents of index register 4. The contents of Q are sent to (00050). Then (00050) is added to A. (A) now contains the sum of index register 6 and 4 and the contents of Q. This sum is stored in 00050. Address 00050 now contains:

$$0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,\underbrace{x\,x\,x\,x\,x}_{\text{Sum of Q, index register 6, and index register 4}}$$

The last instruction, LOAD INDEX LOWER, loads index register 6 with the LOWER order 15 bits (5 octal digits) of 00050.

e.

(00600)	ENI 3 0 0 0 5 1	ENI 0 0 0 0 0 0
(00601)	LDA 3 0 0 0 0 0	STA 3 0 0 3 7 7
(00602)	IJP 3 0 0 6 0 1	Continue program

Explanation: First index register 3 is set to total number of steps minus one. Then the subroutine is transferred to the new location, starting with the last step and ending with the first.

Alternative Solution for problem of Exercise 9e.

Assume index register 3 contains	0 0 3 7 7
Assume index register 2 contains	0 0 0 0 0

		Store index register 2 in upper 00601	Store index register 3 in lower 00601
(00600)	=	SIU 2 0 0 6 0 1	SIL 3 0 0 6 0 1
		Dummy Load	Dummy Store
(00601)	=	LDA 0 (0 0 0 0 0)	STA 0 (0 0 0 0 0)
		Index Skip	Pass
(00602)	=	ISK 2 0 0 0 5 1	ENI 0 0 0 0 0 0
		Index Skip	Unconditional Jump Back to (00600)
(00603)	=	ISK 3 0 0 4 5 0	SLJ 0 0 0 6 0 0

Two index registers are used and are each increased by one in the INDEX SKIP instructions in (00602) and (00603). Also note the PASS following the first INDEX SKIP. This works here since the record INDEX SKIP instruction at (00603) acts as the true barometer as to when to exit. Otherwise the PASS at (00602) would not be able to be used. Why can't the second INDEX SKIP INSTRUCTION be moved into the right side of 00602 and eliminate the PASS?

EXERCISES 10 (solutions)

a. Assume contents of (00025) = XXX0000000000000

Assume contents of (00026) = 1400000000000000

(Note 37.5% is equivalent to 0.3 in octal) which is 140$\longleftrightarrow$0 in fractional format

(00050)	LDA 0 0 0 0 2 6	MUF 0 0 0 0 2 5
(00051)	LRS 0 0 0 0 4 6	STA 0 0 0 1 0 0
(00052)	ENA 0 0 0 0 0 1	SUB 0 0 0 1 0 0
(00053)	STA 0 0 0 1 0 0	Continue

Explanation: First the multiplier is entered into the A-register. Then the Multiply Fractional is performed. The result is shifted so that it is in the lower part of A and rounding off is performed if it is necessary.

b. Mask (00070) = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7

Start (00200)	ENI 1 0 0 0 1 7	ENI 0 0 0 0 0 0
(00201)	LDA 1 0 0 0 1 1	AJP 2 0 0 2 0 3
(00202)	SST 0 0 0 0 7 0	SLJ 0 0 0 2 0 4
(00203)	SCL 0 0 0 0 7 0	ENI 0 0 0 0 0 0
(00204)	STA 1 0 0 0 1 1	IJP 1 0 0 2 0 1
(00205)	Continue program	

Explanation: Each quantity to be tested is entered into the A-register and tested to see whether it is positive. If the quantity is positive, a jump is made to step 00203 where the lower three bits of the quantity are cleared. If the quantity is negative, the lower three bits are set at step 00202. In either case, the quantity is stored at its original location and an index jump is made to test the next quantity.

c.

```
                Load Q with                 L(Q) (00014)
                extractor in 00070          to A register

(00100) =   | LDQ 0   0 0 0 7 0  |  LDL 0   0 0 0 1 4  |

                Store 15 bits               Shift 15 bits in A
                in 00071                    to leftmost position

(00101) =   | STA 0   0 0 0 7 1  |  ALS 0   0 0 0 4 1  |

                Jump if A                   Complement upper
                is positive                 15 bits of A

(00102) =   | AJP 2   0 0 1 0 3  |  SCM 0   0 0 0 7 3  |

                Shift 15 bits to            Add contents of
                original position           index register 3 to A

(00103) =   | ALS 0   0 0 0 1 7  |  INA 3   0 0 0 0 0  |

                                            Enter index
                Store A                     register 3

(00104) =   | STA 0   0 0 0 72   |  LIL 3   0 0 0 7 2  |

                Stop

(00105) =   | SLS 0   0 0 1 0 0  |                     |
```

Extractor in 00070 is 0000000000077777
For complementing 00073 is 7777700000000000

Explanation: The steps above contain descriptions of the functions performed.

d. Assume 00200 contains 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
(This is to be used as an extractor.)
Assume 00201 contains 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
(This is to be used as a counter.)
Assume 00202 contains 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 = 16_{10}

(00050)	ENI 1 0 0 0 0 0	ENA 0 0 0 0 0 0	Clear index; clear A
(00051)	LDQ 0 0 0 1 0 0	ENI 0 0 0 0 0 0	(00100) $\longrightarrow$ Q; Pass
(00052)	LLS 0 0 0 0 0 3	AJP 1 0 0 0 5 5	One digit $\longrightarrow$ A; jump if A $\neq$ 0
(00053)	ISK 1 0 0 0 1 7	SLJ 0 0 0 0 5 2	Skip if all digits tested; jump
(00054)	ENI 1 0 0 0 2 0	ENI 0 0 0 0 0 0	Enter 16_{10} into ind. reg. 1; pass
(00055)	Exit		

Count will be in index register 1

Explanation: Index register 1 and the A-register are cleared. The number to be tested is entered into the Q register. One octal digit is shifted into A. If A is not zero, a jump is made to the exit. If A is zero, this is a non-significant zero, so the count is increased by one and a jump is made to shift the next octal digit into A and test it. If the number to be tested contained sixteen zeros, index register 1 would be cleared to zero. That is why, at step 00054, index register 1 is set to 20_8.

e.

	Enter Contents of Ind. Reg. 4 into (A)	Replace Contents of Ind. Reg. 4 by 15 Rightmost bits of 00100
(00300) =	ENA 4 0 0 0 0 0	LIL 4 0 0 1 0 0

Substitute Low Order 15 Bits
in (A) into Rightmost 15 Bits of 00100

(00301) = $\boxed{\text{SAL 0 0 0 1 0 0}}$

Explanation: Although the above program is short, there are one or two critical conditions which must be satisfied. First, in order to exchange parts of two addresses one part must be sent somewhere else before the exchange begins. In this case the contents of index register 4 are first sent to the A- register. To transfer the contents

e. (Continued)

of A to the contents of 00100 without disturbing the other bits requires an instruction similar to the SUBSTITUTE ADDRESS instruction above (SAL 0 0 0 1 0 0). Note that a "STORE A" instruction cannot be used since it would erase the other bits in 00100. The "LOAD INDEX LOWER" instruction above (LIL 4 0 0 1 0 0) replaces the contents of index register 4 by the 15 rightmost bits of the contents of 00100.

EXERCISES 11 Given the following:

```
a.  00100 ⎫                               00200 ⎫
    00101 ⎪      Contain 50₁₀             00201 ⎪      Contain 50₁₀
      .   ⎪                                 .   ⎪
      .   ⎬                                 .   ⎬
      .   ⎪      Accounts                   .   ⎪      Accounts
      .   ⎪        "C"                      .   ⎪        "D"
    00161 ⎭                               00261 ⎭
```

If any of Accounts "C" = Accounts "D", put a 1 in (00032)

```
                     Enter 50₁₀ into     Load Contents of (00100)
                     Index Reg. 3        into A Register
(00400)    =        |ENI 3   0 0 0 6 2|LDA 0  (0 0 1 0 0)|

                     Equality Search     If Account "C" in "A" Does not equal
                     Starting at 00200   any Account in "D", Jump to (00404).
(00401)    =        |EQS 3   0 0 2 0 0|SLJ 0   0 0 4 0 4 |

                     Enter 1 into        Store Q in Reg. 00032
                     Q Register          (This puts a 1 there)
(00402)    =        |ENQ 0   0 0 0 0 1|STQ 0   0 0 0 3 2 |

                     STOP                PASS
(00403)    =        |SLS 0   0 0 4 0 0|ENI 0   0 0 0 0 0 |
                                           Increase "A" by 1.  This
                     Put Stop 00400        increases the Account "C"
                     into A Register       storage location Reference by 1.
(00404)    =        |LDA 0   0 0 4 0 0|INA 0   0 0 0 0 1 |

                     Store "A" in 00400.        Subtract Contents
                     This modified the step of (00050) from "A"
(00405)    =        |STA 0   0 0 4 0 0|SUB 0   0 0 0 5 0 |
```

Storage address 00050 contains, 503 00062 120 00162.
This will be the quantity tested, in "A", when all accounts are searched.

```
                     Jump Back to 00400
                     if "A" is not zero
(00406)    =        |AJP 1   0 0 4 0 0|Continue Program  |
```

b.

```
                 Put Contents of
                 (00201) into "A" Reg.        Pass
(c)     =   | LDA 0  0 0 2 0 1 | ENI 0  0 0 0 0 0 |

                 Threshold Search, if
                 (00200) > (A) →(c+2)        Jump to (c+3), if (00200) is not
                                                            greater than (A)
(c+1)   =   | THS 0  0 0 2 0 0 | SLJ 0   c+3 |

                 Jump to (00400). Since
                 (00200) > (00201)           Pass
(c+2)   =   | SLJ 0  0 0 4 0 0 | ENI 0  0 0 0 0 0 |

                 Equality Search, if
                 (00200) = (A)  (c+y)
                                                  Jump to some location
(c+3)   =   | EQS 0  0 0 2 0 0 | Jump Exit |      for program following
                                                  the condition that
                 Jump to (00500), Since           (00200) is less than
                 (00200) = (A) = (00201)   Pass   (00201)
(c+4)   =   | SLJ 0  0 0 5 0 0 | ENI 0  0 0 0 0 0 |
```

<u>Explanation:</u> Since SEARCH instructions must appear in the upper instruction of a program step, the first two pass instructions above are needed. Also note that the index register designators in the SEARCH instructions above are zero. This will result in the contents of <u>one</u> address being compared to the contents of the A register.

c. The portion of the program given is attempting to accomplish the following:

1. The number 100_{10} is ENTERED into index register 6.
2. The contents of index register 6 (100_{10}) is put into "A"
3. A Threshold Search is made of 100_{10} consecutive storage locations starting with the contents of storage locations 00100 to see if any of these 100 storage locations contain quantities greater than 100_{10} (which is in "A").
4. If none of the 100 storage locations contain quantities greater than 100, a jump goes to step 00060. If a storage location is found whose content is greater than 100 (in A), then at step 00052, zeros are entered into Q and then into storage location 00500.

In summary, this program is trying to determine if any of the 100 consecutive storage locations starting at address 00100 contain quantities greater than 100; and if so, to store a zero in storage location 00500.

d. Part I. Given (00700)=

MTH 3 0 0 0 5 0	SLJ 0 0 0 7 0 0

Here is a MASKED THRESHOLD SEARCH followed by a jump to the same address containing the search instructions. As long as the logical product of (Q) and the contents of the storage locations starting at address 00050 is greater than the contents of "A", this jump is skipped; however, if the criterion is not met, then this jump back to the search is made. If the jump instruction is reached, the search is complete and the index register contains zero. This step (2 instructions) would be endlessly repeated, but no search would take place in the first half. The danger lies in the fact that if the criterion is not met, a non-terminating loop is formed.

Part II. Given (00105)=

LDA 0 0 0 0 5 0	EQS 3 0 0 2 0 0

Here the EQUALITY SEARCH is programmed in the lower instruction of a program step. When this happens, the next instruction is never skipped and in effect one would always go to the next step at 00106 regardless of the equality criterion being met. The only way one would know whether the criterion was met would be to look at the contents of index register 3. If index register 3 contains a zero, the criterion was not met; if index register 3 contains any number other than zero, the criterion was met.

EXERCISES 12.

(a)

	Set index	Pass
(03001) =	ENI 3 0 0 1 4 3	ENI 0 0 0 0 0 0
	One word to A	Pass
(03002) =	LDA 3 0 0 2 0 1	ENI 0 0 0 0 0 0
	Ship if (M) >(A)	Jump if (A)> (M)
(03003) =	THS 3 0 0 2 0 1	SLJ 0 0 3 0 0 5
	Jump to test new (A)	Pass
(03004) =	SLJ 0 0 3 0 0 2	ENI 0 0 0 0 0 0
	Store greatest quantity	Stop
(03005) =	STA 0 0 0 0 7 7	SLS 0 0 3 0 0 1

<u>Explanation:</u> The first step sets an index equal to one less than the total number of quantities to be examined. At 03002 the last quantity in the list is entered into the A register. At 03003 the remainder of the list is searched to determine if there is a quantity greater than the quantity in the A-register. If such a quantity is found, a jump is made to 03002 where the greater quantity is entered into the A-register. The search is then resumed at the point where it found the greater quantity. When the entire list has been exhausted, a jump is made to 03005. At this time, A contains the greatest quantity in the list and this quantity is stored at 00077.

(b) Given: $\left[\text{Gross Pay} - \$13\ (\text{No. of Deps.})\right] 18\% = \text{Tax}$

Assuming Gross Pay stored at 00022, No. of Deps. stored at 00024, Constant 18 stored at 00025, Constant 13 stored at 00027, and TAX to be stored at 00030.

```
                 No. of Deps.—> A        Multiply by 13.
                 as Multiplier           Product XXX. (Max.) in "A"

(00101) =      | LDA 0   0 0 0 2 4  | MUI 0   0 0 0 2 7  |
                 Shift Product
                 2 Octals                Store Product      (00020) =
                                                               XXX.00 (Max.)
(00102) =      | ALS 0   0 0 0 0 6  | STA 0   0 0 0 2 0  |

                 Gross Pay —>(A)         Gross Pay - $13 (No. of Deps) —>(A)

(00103) =      | LDA 0   0 0 0 2 2  | SUB 0   0 0 0 2 0  |

                 Store Difference        Constant 18 to A as
                                         Multiplier
(00104) =      | STA 0   0 0 0 2 1  | LDQ 0   0 0 0 2 5  |

                 Multiply

(00105) =      | MUI 0   0 0 0 2 1  | LRS 0   0 0 0 0 6  |

                                         Round off if necessary

(00106) =      | QJP 2   0 0 1 0 7  | INA 0   0 0 0 0 1  |

                 Store result

(00107) =      | STA 0   0 0 0 3 0  |                    |
```

(c)

```
            Enter Zeros into              Enter 100₁₀ into
            Ind. Reg. 5                   Ind. Reg. 3
(00011) -  | ENI 5    0 0 0 0 0          | ENI 3    0 0 1 4 4 |

            Enter Extract                 Enter Code Comparator
            Pattern in Q                  in A Reg.
(00012) =  | ENQ 0    0 0 7 0 0          | ENA 0    0 0 3 0 0 |

            Masked Threshold              Jump to 00016

(00013) =  | MTH 3    0 0 3 1 2          | SLJ 0    0 0 0 1 6 |

            Add 1 to contents of          Pass
            00100 and Replace
(00014) =  | RAO 0    0 0 1 0 0          | ENI 0    0 0 0 0 0 |
                           Where initial contents of (00100) = counter =
                                                          0 <------> 0
            Index Skip                    Jump Back to (00012)
(00015) =  | ISK 5    0 0 1 4 3          | SLJ 0    0 0 0 1 2 |


(00016)=   | Continue Program            |                    |
```

<u>Explanation:</u> A critical point in the above program is the use of the "REPLACE ADD ONE" instruction at step 00014. Keep in mind that this instruction destroys the original contents of A (in this case the comparator). Thus the jump back must go to step 00012 where the comparator is entered into A. If some other plan is used for the counter then it may be possible to jump back to step 00013 instead of 00012. (Perhaps you used some other method of running a counter.)

```
(d)  (00060) =   0 7 7 7 7 7 7 7 7 7 7 7 7 7 7 0

                  Set index                  Pass
     (00312) =  | ENI 2    0 0 4 5 3   | ENI 0    0 0 0 0 0 |

                  One word to A              Leftmost octal digit to Q

     (00313) =  | LDA 2    0 2 0 0 0   | LLS 0    0 0 0 0 3 |

                  Shift Q left 6 bit         Shift AQ left 42 bit
                  positions                  positions
     (00314) =  | QLS 0    0 0 0 0 6   | LLS 0    0 0 0 5 2 |

                  Mask to Q                  Set the original 14 center
                                             octal digits
     (00315) =  | LDQ 0    0 0 0 6 0   | SSU 2    0 2 0 0 0 |

                  Store word                 Jump if index register
                                             2 ≠ 0
     (00316) =  | STA 2    0 2 0 0 0   | IJP 2    0 0 3 1 3 |


     (00317) =  | Continue program     |                    |
```

<u>Explanation</u>: The number is shifted around to interchange the first and last octal digits without regard for what is happening to the other fourteen octal digits. After the first and last octal digits have been interchanged, the other fourteen octal digits are set between and the altered number is stored at its original position. Index register 2 is reduced by one in the lower instruction at step 00316, so that the next number will be interchanged when the jump is made.

(e) Given: each location is made up of the following format:

24 Bits First Quantity	24 Bits Second Quantity

Compare and order two quantities in each of 50 addresses.

	Enter Zero into Index register 6	Load Extract Pattern into Q Register
(01000) =	ENI 6 0 0 0 0 0	LDQ 0 0 0 0 5 0

(where 00050 contains 0 0 0 0 0 0 0 0 7 7 7 7 7 7 7 7
(this extracts the SECOND QUANTITY OF EACH ADDRESS)

	Load Q with Log. Prod. of Q and Each Address	Shift A Left 24 Bits
(01001) =	LDL 6 0 0 0 1 1	ALS 0 0 0 0 3 0
	Subtract Constants of each Address	If "A" is Negative Jump to (01005)
(01002) =	SUB 6 0 0 0 1 1	AJP 3 0 1 0 0 5
	Load "A" with Contents of each Address	Shift "A" left 24 bits
(01003) =	LDA 6 0 0 0 1 1	ALS 0 0 0 0 3 0
	Store in Original Address	Pass
(01004) =	STA 6 0 0 0 1 1	ENI 0 0 0 0 0 0
	Index Skip	
(01005) =	ISK 6 0 0 0 6 1	SLJ 0 0 1 0 0 1
(01006) =	Continue Program	

Explanation: The A JUMP test at step 01002 determines if the second quantity is larger than the first. If so, they are reversed in steps 01003 and 01004. The INDEX SKIP instruction at (01005) acts as a count of the 50_{10} times.

EXERCISE 13

a. (1) 31_8 = 0.1 1 0 0 1. (in binary)

Must shift 5 places to binary reference point. Exponent is + 5

floating-point = 010 000 000 101. 110 010 $0 \longleftrightarrow 0$ (in binary)

floating-point = 2 0 0 5 . 620 000 000 000 (in octal)

(2) - 156_{10} (Negative number, plus exponent) (Treat as a positive number, complement the result)

156_{10} = 234_8 = 010 011 100. (in binary)
Must shift 8 places; exponent is 8

floating-point = 010 000 001 000. 100 111 $0 \longleftrightarrow 0$ (in binary)

floating-point = 2 0 1 0 . 470 000 000 000 (in octal)

Complement to) 5767.307 777 777 777 (in octal)
obtain the final)
result)

(3) 0.4_8 = .100 (in binary)

(binary point is in the right position - no shifts are required; exponent is zero.)

floating-point = 010 000 000 000. 100 $0 \longleftrightarrow 0$ (in binary)

floating-point = 2 0 0 0 . 400 000 000 000 (in octal)

(4) - 0.002_8 (Negative number, negative exponent) (Treat as plus number, negative exponent, and complement the final result.)

+0.002_8 = .000 000 010 must shift 7 places, exponent is -7

floating-point = 001 111 111 000. $100 \longleftrightarrow 0$ (in binary)

floating-point = 1 7 7 0 . 400 000 000 000 (in octal)

Complement to obtain) = 6007. 377 777 777 777 (in octal)
the final result)

b. (1) 2 010. 612 000 000 000

The binary form is $\underset{\text{Sign of Number}}{0}\underbrace{10\ 000\ 001\ 000}_{\text{Biased Exponent}}.\ \underbrace{110\ 001\ 010\ 0 \longleftrightarrow 0}_{\text{Fractional Number}}$

Sign of number is +
Exponent (unbiased) is 8
To find number, shift point 8 places right

$.110\ 00101 \curvearrowright \quad = \quad 011\ 000\ 101 \quad = \quad 305$

Number is 305_8

(2) 5772.317 777 777 777

The binary form is $\underset{\text{Sign of Number}}{1}\underbrace{01\ 111\ 111\ 010}_{\text{Biased Exponent}}.\underbrace{011\ 001\ 111 \longleftrightarrow 1}_{\text{Fractional Number}}$

Sign of number is -

Since sign of number is minus, complement the whole form.

2005.460 000 000 000 (complemented)

$\underbrace{010\ 000\ 000\ 101}_{\text{Biased Exponent}}.\underbrace{100\ 110\ 0 \longleftrightarrow 0}_{\text{Number}}$

The unbiased exponent is +5. Therefore, shift 5 places right in the number.

$.100\ 11 \curvearrowright 0\ 0$

Number is -23

c.	(c) =	ENI 2 0	ENI 0 ———	Clear (B^2), pass
	(c+1) =	LDA 2 S1	SCA 1 2057	(S1) A Normalize
	(c+2) =	ALS 0 1	STA 0 T	Shift Left 1 Store Number
	(c+3) =	ENA 1 0	LDQ 0 T	Exponent to A, normalized number to Q
	(c+4) =	LLS 0 44		Shift into floating-point format in A

Using the above, follow the following example, assume (S1) = 35_8

(c+1) = 35 $\longrightarrow$ A, $\overset{42_{10} = 52_8 \text{ shifts}}{\boxed{\longleftarrow\ 011\ 101}}$ normalized in A

(c+2) = $\overset{\text{Shift left 1 in A}}{\boxed{111\ 010 \longleftrightarrow 0}} \longrightarrow$ Store in T

(c+3) = Shift count in B^1 = 2057 - 52 = 2005 in B^1. Enter this in A; enter normalized number in Q.

$\overset{A}{\boxed{0 \longleftrightarrow 0\ 010\ 000\ 000\ 101}}$ $\overset{Q}{\boxed{111\ 010 \longleftrightarrow 0}}$

(c+4) = Long left shift 36_{10} places puts number in floating-point format in A.

APPENDIX B

SOLUTIONS TO THE CHAPTER REVIEW TESTS

This appendix provides the key to the review tests which are found at the end of the first five chapters of the preliminary instruction manual. Alternative solutions are presented in some instances where program routines are asked for, indicating that oftentimes there is no hard and fast approach to programming situations.

CHAPTER I

1. Core

2. 48

3. $32,768_{10}$

4. False

5. 00000 - 77777

6.

Operation Code	Index Designator	Address
12	1	01000
6 bits	3 bits	15 bits

7.

Instruction	Instruction
Instruction	Pass
Pass	Instruction

8. True (unless there is an assembly routine which permits decimal notation)

9. False

10. 6

11. Accumulator and Q Register

12. Program Address Register (P)

13. Program Control Register (u)

14. Accumulator and Q Register

15. Yes

16. 62

17. 400 sq. ft.

18. (1) Magnetic tape units
(2) Paper tape reader
(3) Paper tape punch
(4) Typewriter

19. True

20. True

CHAPTER II

1. Some of the rightmost bits are lost

2. False

3. The sign bit is extended to the right in the register

4. Accumulator or Q Register

5. Yes, and a shift fault will be set

6. False

7. False

8. 0000000000000007

9. 0000000000000007

10.

(1)	LDA 0 00050	STA 0 01000
or		
(2)	LDQ 0 00050	STQ 0 01000

11. It is an advantage to be able to transfer through Q without disturbing (A), and it saves an instruction if it is desired to store a quantity already in Q.

12. 0000000000000001

13. At the address contained in index register 6

14. False

15.

00100	=	LDQ	0	00016	LDA	0	00017
00101	=	QJP	1	00102	STQ	0	00500
00102	=	AJP	1	00103	STA	0	00501
00103	=	STQ	0	00700	ADD	0	00700

16.

```
(00011)  =   0000000100000000 }  constants
(00005)  =   0000000000000144 }

00100  =  | STQ  0  00010 | LDA  0  00010 |   (Q) ——→ A
00101  =  | ADD  0  00012 | AJP  7  01000 |   Add T1, return
                                              jump if negative
00102  =  | STA  0  00400 | STA  0  00010 |   Store (A) in S1
                                              and at 00010
00103  =  | LDA  0  00102 | ADD  0  00011 |   Add 1 to S1
00104  =  | STA  0  00102 | LDA  0  00101 |   Store (S1) +1
                                              back in S1
00105  =  | ADD  0  00011 | STA  0  00101 |   Add 1 to 00101
00106  =  | LDA  0  00006 | ADD  0  00007 |   Add 1 to counter
00107  =  | STA  0  00006 | SUB  0  00005 |   Test counter for
                                              equality to 100₁₀
00110  =  | AJP  1  00111 | SLS  0  00000 |   Jump if (A)≠ 0,
                                              stop
00111  =  | LDA  0  00010 | AJP  0  00101 |   Load 00010 into
                                              A, jump to 00101
00112  =  | AJP  1  00101 | Continue Program|
```

CHAPTER III

1. True

2. False

3. True

4. 0000000000000001 (because the arithmetic is one's complement)

5. 7777777777777774

6. INA 4 00000

7. No

8. Multiplier - A register
   Multiplicand - storage address
   Product - QA register

9. A register - 7777777777777776
   Q register - 0000000000000002

10.

(00300)	=	LDA	0	00007	INA	2	00000
(00301)	=	AJP	3	00304	LDQ	0	00050
(00302)	=	DVI	0	00100	STQ	0	00200
(00303)	=	SLJ	0	01000	ENI	0	00000
(00304)	=	STA	0	00060	SLJ	0	01000

11.

(00200)	=	STQ 0 00100	STA 0 00101	Store (Q_i) and (A_i)
(00201)	=	SUB 0 00100	STA 0 00102	Subtract (Q) from (A), store the difference
(00202)	=	LDA 0 00101	ADD 0 00100	Load A_i into A, add Q_i
(00203)	=	AJP 3 00206	ENQ 0 00000	
(00204)	=	DVI 0 00102	Continue Program	
(00205)	=			
(00206)	=	ENQ 0 77777	SLJ 0 00204	

12.

(00100)	=	ENA 3 00000	STA 0 00200	Place X in Q and memory
(00101)	=	MUI 0 00200	ARS 0 00001	Square X and divide by two
(00102)	=	STA 0 00201	ENA 0 00001	Store (Q), enter 1 into A
(00103)	=	SUB 0 00201	STA 0 00077	Subtract (00201) from 1, store in 00077

CHAPTER IV

1. Upper

2. The next instruction is executed.

3. The contents of the memory location are not changed by the storage skip, they are shifted left one place by the storage shift.

4. Index register 3 is cleared and the next instruction is skipped

5. Twice

6. Memory, Accumulator or Q register

7. 0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

8. False

9. False (A yes, but Q no)

10. 0000000000000017

11. Yes

12. $\boxed{\text{ENI} \quad 0 \quad 00000}$

13. False

14. True

15.

(00100)	=	ENI 1 00000	ENI 0 00000	Clear B_1
(00101)	=	ENQ 0 00007	ENA 5 00000	Extractor $\rightarrow$ Q, $(B_5) \longrightarrow$ A
(00102)	=	STL 1 00100	ARS 0 00003	Store the last digit, Shift A right
(00103)	=	ISK 1 00004	SLJ 0 00102	Loop back to 00101 4 times

(00104)	=	LDA 1 00100	INI 1 00001	Place first digit in A, and a 1 in index register 1
(00105)	=	ENI 0 00000	ADD 1 00100	Add first two digits
(00106)	=	ISK 1 00004	SLJ 0 00105	Add 3 remaining digits
(00107)	=	STA 0 00050	LIL 3 00050	Store sum in 00050 and index register 3

(an alternative solution)

(00100)	=	ENI 1 00004	ENQ 5 00000	Enter 4 into index register 1 and (index register 5) into Q
(00101)	=	LDL 0 00201	ADD 0 00200	Load last digit into A, add (00200)
(00102)	=	STA 0 00200	QRS 0 00003	Store (A), Shift right 3
(00103)	=	IJP 1 00101	LIL 3 00200	Jump to 101, reduce index reg. 1, when ind. reg. 1 = 0, load sum into ind. reg.3

```
(00200)  =  0000000000000000 }
(00201)  =  0000000000000007 }  constants
```

CHAPTER V

1. False

2. A and Q

3. $0.100 \longleftrightarrow 011$

4. $0.111 \longleftrightarrow 111$ or $.377 \longleftrightarrow 77_8$

5. True

6. False

7. False (either 3 or 7 will stop the computer)

8. $\boxed{\text{SST} \quad 0 \quad 00100}$

9. $\boxed{\text{SCL} \quad 0 \quad 00100}$

10. One storage location will be searched - the one whose address is the execution address

11. An unconditional jump

12. The next instruction is skipped

13. The next instruction is executed

14. The execution address of the search instruction is added to the contents of the index register referenced by the search instruction

15. Equality Search
Threshold Search
Masked Equality Search
Masked Threshold Search

16. Instruction: $\boxed{\text{THS 5 00062}}$
Contents of index register 5: 00062

17. The Add instruction stores its sum in A, the Replace Add does this but also stores the sum in memory.

18. A register: 7777777777777776
(00100) : 7777777777777776

19. A register: 0000000000000007
(00100) : 0000000000000007

20.

(00100)	=	ENI 1 01747	ENI 0 00000	Enter 1747 (1000_{10}) into index reg. 1
(00101)	=	LDA 1 00201	STA 1 00204	Load contents of last address into A, store at address3 greater
(00102)	=	IJP 1 00101	Continue Program	Repeat 1000 times

(an alternative solution)

(00100)	=	ENI 1 77777	ENI 2 01747	Enter negative zero into B_1. Enter 1000_{10}-1 into B_2
(00101)	=	LDA 1 02150	STA 1 02153	Load contents of last address into A, store at address 3 greater
(00102)	=	INI 1 77776	IJP 2 00101	Subtract 1 from ind. reg. 1, jump to 00101 repeat 1000 times

APPENDIX C

1604 ASSEMBLY ROUTINE

(JUNE 1959)

GENERAL DESCRIPTION

The assembly routine will convert a program from a special symbolic shorthand notation to a fully-encoded program suitable for loading into the computer.

In this shorthand notation, each instruction or constant is represented by one entry. An entry is one line of information, as typed on a Flexowriter or punched onto one card.

Certain pseudo-instructions can also be represented by certain entries. Pseudo-instructions are used to specify a starting address, to insert constants, to reserve space, to include special remarks, to assign addresses to alpha-numeric location symbols, to call for the assembly of certain subroutines from a special magnetic library tape, or to end the assembly.

Each entry consists of one to four terms, followed by remarks. Each term consists of one or more consecutive characters, followed by one or more spaces or tabs (Flexowriter tabulator functions.). The remarks consist of anything following the last term.

The arrangement and nomenclature of terms and the remarks for each entry are as follows:

Tag, operation code, b-term, m-term, remarks.

For certain types of entries, some of these terms are not required. The remarks are always optional. The operation code term is always required.

The tag term is used to assign an alpha-numeric location symbol to an instruction or constant. This permits execution addresses in the program to be represented symbolically. A location symbol consists of a letter, followed by letters or digits, to a maximum of eight characters.

The operation code term can be either a three-letter mnemonic code, or two digits corresponding to the octal operation code portion of the desired instruction.

The b-term, used to represent the b-designator for instructions, can be either a single digit (0-7) or the letter "N" to represent indirect addressing. "N" is equivalent to b = 7.

The m-term is a value in either absolute or symbolic representation. For instruction entries, the m-term represents the m-portion, or base execution address, of the instruction.

For example, an instruction is represented by a tag (optional), a three-letter operation code, a single-character b-term, and an m-term representing a base execution address. A portion of a program encoded in this manner will appear as follows:

Tag	Operation Code	B-term	M-term	Remarks
LOOP	ADD	0	CONSTANT	Add Constant
	SUB	6	LIST+7	Subtract next value
INDEX	IJP	6	LOOP	Index B6
	SLS	0	EXIT	Stop, jump to exit

The symbol LOOP appearing in the m-term of the IJP instruction thus specifies an indexed jump to the ADD instruction. The terms CONSTANT, LIST, and EXIT refer to otherlocations specified in a similar manner, elsewhere in the program. In the SUB instruction, the value following the symbol LIST modifies it, so that the base execution address portion of the instruction is equal to the numerical location equivalent of LIST, plus seven.

A typical pseudo-instruction entry is that for the Origin Address function. To start a program at address 70000, the Origin pseudo-instruction would precede the first instruction entry, as follows:

Tag	Operation Code	B-term	M-term	Remarks
(none)	ORG	(none)	70000	Start at 70000

The use of the symbolic codes and locations not only simplifies the writing of a program, but also facilitiates debugging, allows rapid reencoding for different ranges of addresses, and provides the programmer with a clearer understanding of what a program is accomplishing than could be had by examination of an entirely numerical copy of the program. Also the pseudo-instructions used with the assembly permit many programming short cuts to be made.

A simplified flow diagram of the program is shown in Figure 17. Instructions are entered on consecutive lines of a worksheet. These entries along with remarks the programmer may have entered to the right of any entry are either typed on a Flexowriter to punch a paper tape, or punched onto 80-column cards (one line per card). The cards or paper tape entries are first converted to successive binary-coded-decimal records on magnetic tape, and the resulting tape is used as the input medium for the "assemble" portion of the assembly program.

The "assemble"routine will normally require two executions, or "passes". The first pass examines the input magnetic tape to determine the assignment of all location symbols. The assignments are stored in a directory capable of holding assignments for 2048 location symbols. The second pass, made with the same input tape, performs the translation from the symbolic-encoded form to the numeric form and records both forms along with the original remarks. Both passes check for certain errors and record these errors on a printed sheet via the monitor typewriter.

If a simple arrangement rule is followed, the routine can function as a one-pass assembler. The rule is that each location symbol must be defined by using it as a tag before it is used in an m-term. If the operator is attempting to write for one-pass but has ignored this rule, the assemble routine will sense the error and stop for a change to the two-pass mode.

The output is on magnetic tape, which can be converted to paper tape punched in Flexowriter code, converted to line-printer copy, or loaded into the computer memory.

Examples of a symbolic-encoded program and the resulting output are given in Tables I and II. The rules for using the instructions, pseudo-instructions, tags, and remarks are given below.

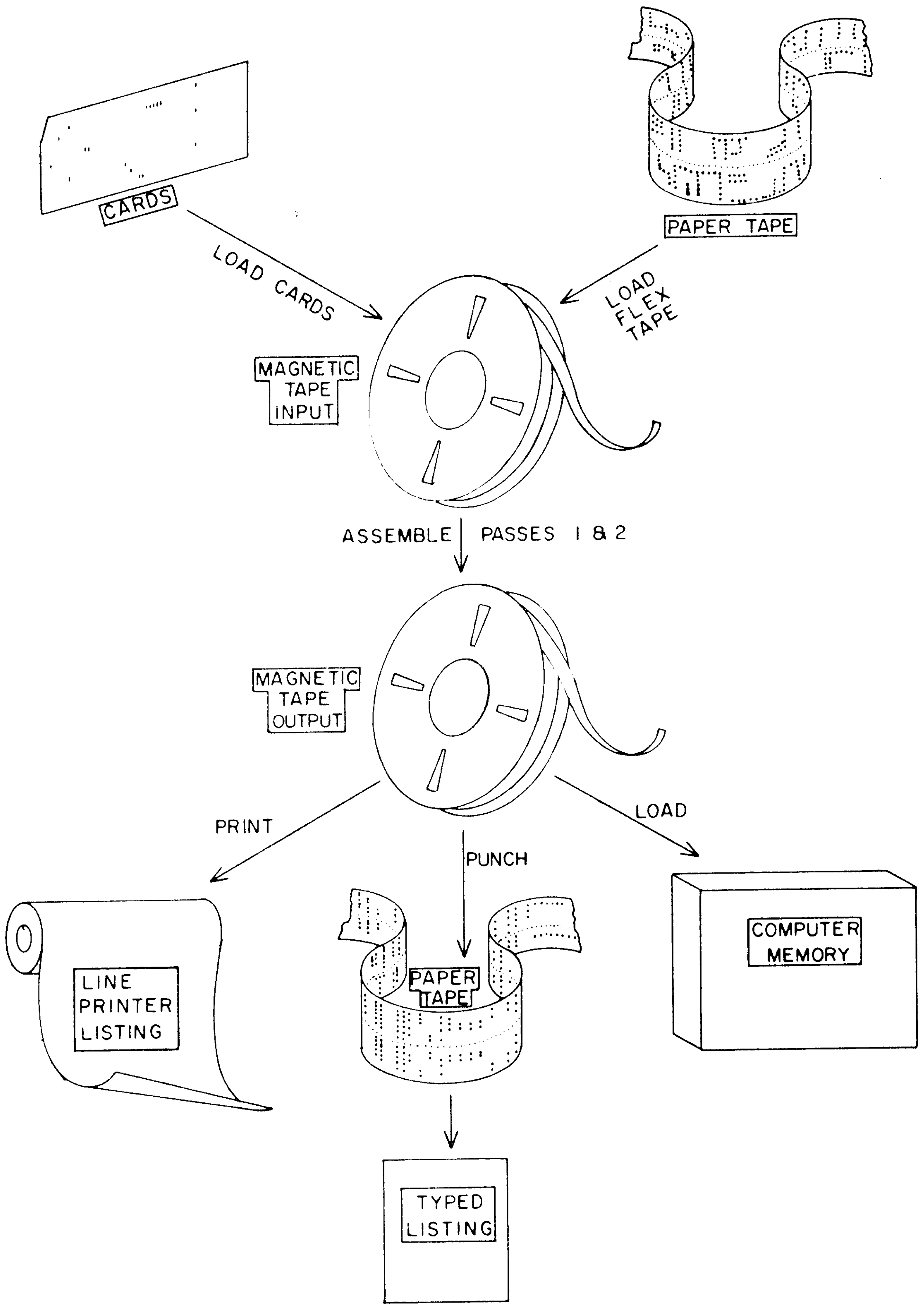


Figure 17
SIMPLIFIED FLOW DIAGRAM
Assembly Program

DESCRIPTION OF ENTRIES

a. REMARKS ONLY - If it is desired to enter a line of remarks only on the copy, an operation code REM is followed by a maximum of 80 characters. For all other types of entries, remarks are limited to 48 characters.

b. EQUIVALENCE - To assign a location symbol to the first location of a group of working storages, or to equate two location symbols, a tag consisting of a location symbol (letter followed by letters and digits, up to eight characters) is followed by the operation code EQU, and an m-term address. The address can be an all-numeric octal value denoted by a value consisting of a maximum of five digits; an all-numeric decimal value denoted by appending "D" to the value term; or regional address, consisting of a location symbol followed by a + or – and a value of one to five digits, with a "D" appended if the modifying value is to be interpreted as decimal.

c. ORIGIN ADDRESS - To start a sequence of instructions at a particular location, an operation code ORG is followed by a regional or numeric m-term address similar to that used to the right of the EQU operation code.

d. INSTRUCTION - For an instruction, the tag term is optional, but, when used, assigns a location symbol to theinstruction location. A location symbol tag can only appear to the left of an instruction that is to be assembled into the upper half of a word.

The b-designator may be a digit (0 through 7) or the letter "N", which is equivalent to 7 and denotes "indirect".

The m-term can consist of a numerical or regional address similar to that used after the EQU code. An additional feature is that the region symbol portion of this term can also consist of a slash (/), to denote the location of the current word. Thus, to jump to a location two greater than the current location, the operation code, b, and m-terms are as follows: SLJ 0 /+2.

e. DECIMAL VALUE - To insert a decimal constant into a location, the operation code DEC is used. A tag consisting of a location symbol is optional. The m-term consists of a sign (+, – , or blank), a value of up to 14 decimal digits, a decimal scaling consisting of a "D" followed by a sign (+ or–) and one or more decimal digits, and a binary scaling consisting of a "B" followed by a sign (+ or –) and one or more decimal digits. If the value contains a decimal point (period) in any position, the constant is packed into floating point form. If no decimal point is present, an integral constant will result. Both the decimal and binary scalings are optional.

f. OCTAL VALUE - To insert an octal constant, the operation code OCT is used. A tag consisting of a regional symbol is optional. The m-term consists of a sign (+, -, or blank) and up to 16 octal digits.

g. BINARY-CODED-DECIMAL INSERTION - To insert binary-coded-decimal characters into a word, the BCD operation code is used. A region symbol tag is optional. The m-term consists of a slash (/) and the next eight characters, including spaces.

h. FLEXOWRITER CODE INSERTION - To insert Flexowriter codes into a word, the FLX operation code is used. A region symbol tag is optinnal. The m-term consists of a slash (/) and eight characters, including spaces.

i. LIBRARY SUBROUTINE ASSEMBLY - To insert an entire library subroutine into the program, the title of the subroutine (stored on the library tape) is used as a tag. This is followed by the operation code term LIB, and an m-term representing a regional or absolute starting address for the subroutine. The starting address is similar to the m-term used for an instruction.

j. RESERVE BLOCK STARTING WITH SYMBOL - To reserve a block of consecutive addresses and assign a location symbol to the first location of the block, a location symbol tag term is followed by the operation code BSS and an m-term consisting of a maximum of five digits, appended by "D" if the value is to be interpreted as decimal, or no "D" if the value is octal. The m-term specifies the number of locations to be reserved.

k. RESERVE BLOCK, END WITH SYMBOL - If it is desired to reserve a block as described above, except that a location symbol is to be assigned to the last location of the block, a location symbol tag is followed by BES, and an m-term similar to that used for BSS.

l. END ASSEMBLY - To end the assembly program during either pass, the last entry must consist of the operation code END.

FORMAT RULES

Each entry is interpreted as the information on a punched card or one line of Flexowriter copy. If Flexowriter tape is to be used, a blank line (produced by a carriage return not preceded by any visible symbols) is ignored by the assembly program.

The assembly program detects terms by means of their separatinn of spaces, with the number of terms that are detected depending upon the type of entry. Anything appearing to the right of the last term is interpreted as remarks. The only input format requirement is that each term (except the m-term for FLX or BCD) consist of successive characters, and that the left-most character of the tag term, if present, must be in the extreme left position of the input: i. e., column 1 on a punched card, or the left margin for Flexowriter input. Any number of spaces may separate the terms. Non-significant digits of a value may be omitted. In an all-numeric term of positive value, the + sign may also be omitted.

All of these features are included to eliminate the tedious writing of redundant symbols on the input copy, and to make the reproduction onto the cards or tape as simple as possible.

During encoding, the programmer must realize that each instruction occupies one-half of a word, and each operand occupies an entire word. Because some instructions such as Load Index Lower refer to a specific half of a word, and because the addition of a constant n to an address increments the address by n locations (not by n instructions), it is necessary that the programmer pair his instructions; i. e., he must know, for each instruction, whether it is to be assembled into the upper or lower half of a computer word, and must include pass instructions (ENI with $b = 0$) or blanks.

The detailed format rules for each type of entry are given in Table III and its accompanying notes.

TABLE I

ASSEMBLER INPUT FORMAT

```
        REM  --  TRANSFER 6SIN3X TO Y, FOR 3 X-VALUES

YSTOR   EQU    6000               Assign Y storage
        ORG    500                Start at address 500
COMP    ENI 1 0                   Start of computation, clear B1
        ENA 0 0                   Clear A
LOOP    LDQ 1 XSTOR               X-Storage to Q
        MUI 0 CONST               Multiply by 3
        LLS 0 60                  Shift product to A
        SLJ 4 SINE                Return jump to sine
        LRS 0 60                  Sine to Q
        ENA 0 0                   Clear A
        MUI 0 CONST+1             Multiply by six
        STQ 1 YSTOR               Store Product
        ISK 1 2                   Index B1 toward 2
        SLJ 0 LOOP                Jump to obtain next X
        SLS 0 COMP                Stop
          0 0 0                   Blank
XSTOR   DEC   +1247D+1            Value 12470 dec.
        DEC   -79825              Value -78825 dec.
        DEC   +52891B+1           Value +105782 dec.
CONST   OCT   3                   Value 3
        OCT   6                   Value 6
TEMP    BSS   3                   Reserve 3 locations
SINE77  LIB   /                   Load Sine Routine No. 77 here
        END                       End Assembly
```

TABLE II

ASSEMBLER OUTPUT FORMAT

```
                                 REM   --  TRANSFER 6SIN3X TO Y, FOR 3 X-VALUES

                          YSTOR EQU    6000           ASSIGN Y STORAGE
                                ORG    500            START AT ADDRESS 500
00500   50 1 00000  COMP  ENI 1 0         START OF COMPUTATION, CLEAR B1
        10 0 00000        ENA 0 0         CLEAR A
00501   16 1 00600  LOOP  LDQ 1 XSTOR     X-STORAGE TO Q
        24 0 00100        MUI 0 CONST     MULTIPLY BY 3
00502   07 0 00060        LIS 0 60        SHIFT PRODUCT TO A
        75 4 00514        SLJ 4 SINE      RETURN JUMP TO SINE
00503   03 0 00060        LRS 0 60        SINE TO Q
        10 0 00000        ENA 0 0         CLEAR A
00504   24 0 00101        MUI 0 CONST+1   MULTIPLY BY SIX
        21 1 00604        STQ 1 YSTOR     STORE PRODUCT
00505   54 1 00002        ISK 1 2         INDEX B1 TOWARD 2
        75 0 00501        SLJ 0 LOOP      JUMP TO OBTAIN NEXT X
00506   76 0 00500        SLS 0 COMP      STOP
        00 0 00000          0 0 0         BLANK
00507   00 0 00000  XSTOR DEC   +1247D+1  VALUE 12470 DEC.
        00 0 30266
00510   77 7 77777        DEC   -79825    VALUE -79825 DEC.
        77 5 44056
00511   00 0 00000        DEC   +52891B+1 VALUE +105782 DEC.
        00 3 16526
00512   00 0 00000  CONST OCT   3         VALUE 3
        00 0 00003
00513   00 0 00000        OCT   6         VALUE 6
        00 0 00006
00514   00 0 00000  TEMP  BSS   3         RESERVE 3 LOCATIONS
        00 0 00000
00517   75 0 77777  SIN77 LIB   /         LOAD SINE ROUTINE NO. 77 HERE
        50 6 00007
           .
           .
        (etc., to end of sine routine)
           .
           .
01026   12 0 01000
        75 0 00514
                          END             END ASSEMBLY
```

TABLE III

Detailed Format Rules

(See Accompanying Notes)

Function	Tag	Op. Code	B-term	M-term	Remarks
Remarks only		REM			Note 13
Symbol Equivalence	Note 1	EQU		Note 7	Note 14
Origin Address		ORG		7	Note 14
Instruction	Note 2	Note 5	Note 6	Note 7	Note 14
Decimal Value	Note 3	DEC		Note 8	Note 14
Octal Value	Note 3	OCT		Note 9	Note 14
BCD Characters	Note 3	BCD		Note 10	Note 14
Flexowriter Characters	Note 3	FLX		Note 11	Note 14
Library Assembly	Note 4	LIB		Note 7	Note 14
Reserve Block, Start with Symbol	Note 3	BSS		Note 12	Note 14
Reserve Block, End with Symbol	Note 3	BES		Note 12	Note 14
End Assembly		END			Note 14

Note 1: The tag must be present, and must be a location symbol (a letter followed by letters or digits, to a maximum of eight characters).

Note 2: The tag is optional, but if present, must be a location symbol and must be located only to the left of an instruction that is to occupy the upper-half position of a word.

Note 3: The tag is optional, but, if present, must be a location symbol.

Note 4: The tag specifies the title of a library routine. The title, which must agree with that recorded in the first word of the library routine on the tape, must consist of one to eight characters.

Note 5: For instructions, the operation code may be either the three-letter mnemonic symbol, or the two-octal-digit operation code.

Note 6: The b-designator may be either a single octal digit (0 through 7), or the letter "N", to denote indirect addressing. The letter "N" is equivalent to 7.

Note 7: The m-term may be either an absolute octal address (one to five octal digits), an absolute decimal address (one to five digits followed by a D), or a symbolic relative address consisting of a location symbol, a sign (+ or -), and a modifying value. The modifying value may be octal, or decimal digits followed by a D. Also, for instructions on library assembly, the location symbol can be a slash mark (/) and a modifying value, with "/" denoting "this location".

Note 8: The m-term for a decimal constant must consist of a sign (+ , -, or blank) followed by a maximum of 14 digits. A decimal scaling consisting of a D followed by a sign (+ or -) and a decimal value, and a binary scaling consisting of a B followed by a sign (+ or - and a decimal value are both optional. If a decimal point appears in the value, or to the right of the value, the value is packed into floating point form.

Note 9: The value for an octal constant must consist of a sign (+ , -, or blank) followed by a maximum of 16 octal digits.

Note 10: To insert binary-coded-decimal characters, the value must consist of a slash (/) followed by eight characters, including spaces.

Note 11: To insert Flexowriter characters, the value must consist of a slash (/) followed by eight characters, including spaces, shift up, and shift down. If the program input is via punched cards or the output is via the line printer, certain characters are not permissible.

Note 12: The value for a block reserve must consist of a + sign or blank, followed by a maximum of five digits and a "D" if the value is decimal. If the "D" is not present, the value is interpreted as octal.

Note 13: Remarks for the REM code may consist of a maximum of 80 characters, including spaces.

Note 14: Remarks for all codes except REM may consist of a maximum of 48 characters, including spaces.

TABLE IV

SUMMARY OF OPERATION CODES FOR 1604 ASSEMBLER

A. INSTRUCTION OPERATION CODES

Note: Use Either Mnemonic or Two-Digit Code

Digits	Mnemonic	
00	ZRO	Zeros in Operation Code Position
01	ARS	A Right Shift
02	QRS	Q Right Shift
03	LRS	Long (AQ) Right Shift
04	ENQ	Enter Q
05	ALS	A Left Shift
06	QLS	Q Left Shift
07	LLS	Long (AQ) Left Shift
10	ENA	Enter A
11	INA	Increase A
12	LDA	Load A
13	LAC	Load A, Complement
14	ADD	Add
15	SUB	Subtract
16	LDQ	Load Q
17	LQC	Load Q, Complement
20	STA	Store A
21	STQ	Store Q
22	AJP	A Jump
23	QJP	Q Jump
24	MUI	Multiply Integer
25	DVI	Divide Integer
26	MUF	Multiply Fractional
27	DVF	Divide Fractional
30	FAD	Floating Add
31	FSB	Floating Subtract
32	FMU	Floating Multiply
33	FDV	Floating Divide
34	SCA	Scale A
35	SCQ	Scale AQ
36	SSK	Storage Skip
37	SSH	Storage Shift
40	SST	Selective Set
41	SCL	Selective Clear
42	SCM	Selective Complement
43	SSU	Selective Substitute
44	LDL	Load Logical

INSTRUCTION OPERATION CODES (Continued)

45	ADL	Add Logical
46	SBL	Subtract Logical
47	STL	Store Logical
50	ENI	Enter Index
51	INI	Increase Index
52	LIU	Load Index, Upper
53	LIL	Load Index, Lower
54	ISK	Index Skip
55	IJP	Index Jump
56	SIU	Store Index, Upper
57	SIL	Store Index, Lower
60	SAU	Substitute Address, Upper
61	SAL	Substitute Address, Lower
62	INT	Input Transfer
63	OUT	Output Transfer
64	EQS	Equality Search
65	THS	Threshold Search
66	MEQ	Masked Equality
67	MTH	Masked Threshold
70	RAD	Replace Add
71	RSB	Replace Subtract
72	RAO	Replace Add One
73	RSO	Replace Subtract One
74	EXF	External Function
75	SLJ	Selective Jump
76	SLS	Selective Stop
77	SEV	Sevens in Operation Code Position

B. PSEUDO-OPERATION CODES

REM	Enter Remarks Only
EQU	Equate or Define Location Symbol
ORG	Set Origin Address
DEC	Insert Decimal Constant
OCT	Insert Octal Constant
BCD	Insert Eight BCD Characters
FLX	Insert Eight Flexowriter Characters
LIB	Insert Library Routine
BSS	Reserve Block, Start with Location Symbol
BES	Reserve Block, End with Location Symbol
END	End Assembly

TABLE V

BCD CODES USED FOR ASSEMBLER

Character	Code	Character	Code
0	12	O	46
1	01	P	47
2	02	Q	50
3	03	R	51
4	04	S	22
5	05	T	23
6	06	U	24
7	07	V	25
8	10	W	26
9	11	X	27
A	61	Y	30
B	62	Z	31
C	63	=	13
D	64		
E	65	-	40
F	66	$	53
G	67	*	54
H	70	+	60
I	71	.	73
J	41	)	74
K	42	blank (space)	20
L	43	/	21
M	44	,	33
N	45	(	34

Note: In the event that these do not agree with the codes used at a particular installation, the assembly routine can be modified easily, as these codes are all contained in a compact "code list" in the routine.

APPENDIX D

LOAD AND DUMP ROUTINES

FLEX LOAD

The Flex Load routine will read a paper tape prepared in flexcode, translate it into binary, and load it into the core memory of the computer. For each memory location to be loaded, the paper tape must contain the address of the memory location and the information which is to be loaded into that location. The address must be preceded by a carriage return. The routine will accept paper tapes which contain a check sum (output of Flex Dump routine). To load paper tapes prepared on the Flexowriter, set Selective Jump Key 1. (It is assumed that tapes prepared on the Flexowriter will not contain a check sum.)

The Flex Load routine will accept tapes prepared in two different formats:

1. Carriage return, address, tab, upper instruction, remarks (optional), carriage return, tab, lower instruction, remarks (optional).

2. Carriage return, address, upper instruction, lower instruction, remarks (optional).

If a tape prepared in format 2 is to be loaded, set Selective Jump Key 2. In either format the spacing within the instruction may be variable.

The last information to be entered into the computer should be followed by a carriage return and an end code. When the routine recognizes this end code, the check sum punched on the paper tape is compared with the check sum computed while the paper tape is being loaded. If these are the same, a jump to the beginning of the Flex Load routine is made and the computer stops. If they are not the same, the typewriter types 'check sum error', a jump is made to the beginning of the Flex Load routine, and the computer stops.

FLEX DUMP

The Flex Dump routine punches, in flexcode, a paper tape of the contents of a specified area of core memory. The area is specified by placing the first address of the area in index register 6 and the last address of the area in index register 5. These addresses are inclusive. If it is desired to suppress punching the contents of those addresses containing zero, Selective Jump Key 1 should be set.

A check sum of the contents of the locations being dumped is computed and punched on the tape. This check sum is preceded and followed by carriage returns. Each address of the specified area is punched, followed by the contents of that address and a carriage return. When an address ending with seven is punched, the contents of that address are followed by two carriage returns. After the carriage return following the contents of the last address to be dumped, an end code is punched. This end code will be recognized by the Flex Load routine if this tape is used as an input tape.

Format sample:

```
76 5 43210  01 2 34567       (check sum)
01000  75 1 01101  20 0 01076
01001  10 0 01200  14 3 02000
```

BIOCTAL LOAD

The bioctal load routine is used to load a bioctally-punched paper tape into the computer via the paper tape reader. The format required on the paper tape is the same as the output format of the Bioctal Dump routine. The addresses to be loaded appear first on the tape, followed by a check sum of the information to be loaded. The information to be loaded into those addresses then follows.

A check sum is computed as the information from the paper tape is loaded into the computer. After the tape has been loaded, the computed check sum is compared with the punched check sum to see if they are the same. If they are the same, this is a reliable indication that the information has been loaded correctly. A difference in check sums indicates that the information has not been correctly loaded into the computer. In the latter case, the typewriter will type'check sum error'. In either case, the computer stops at the beginning of the bioctal load routine.

The reader assembly mode of operation is used in this routine, necessitating the presence of a seventh level hole every eighth frame.

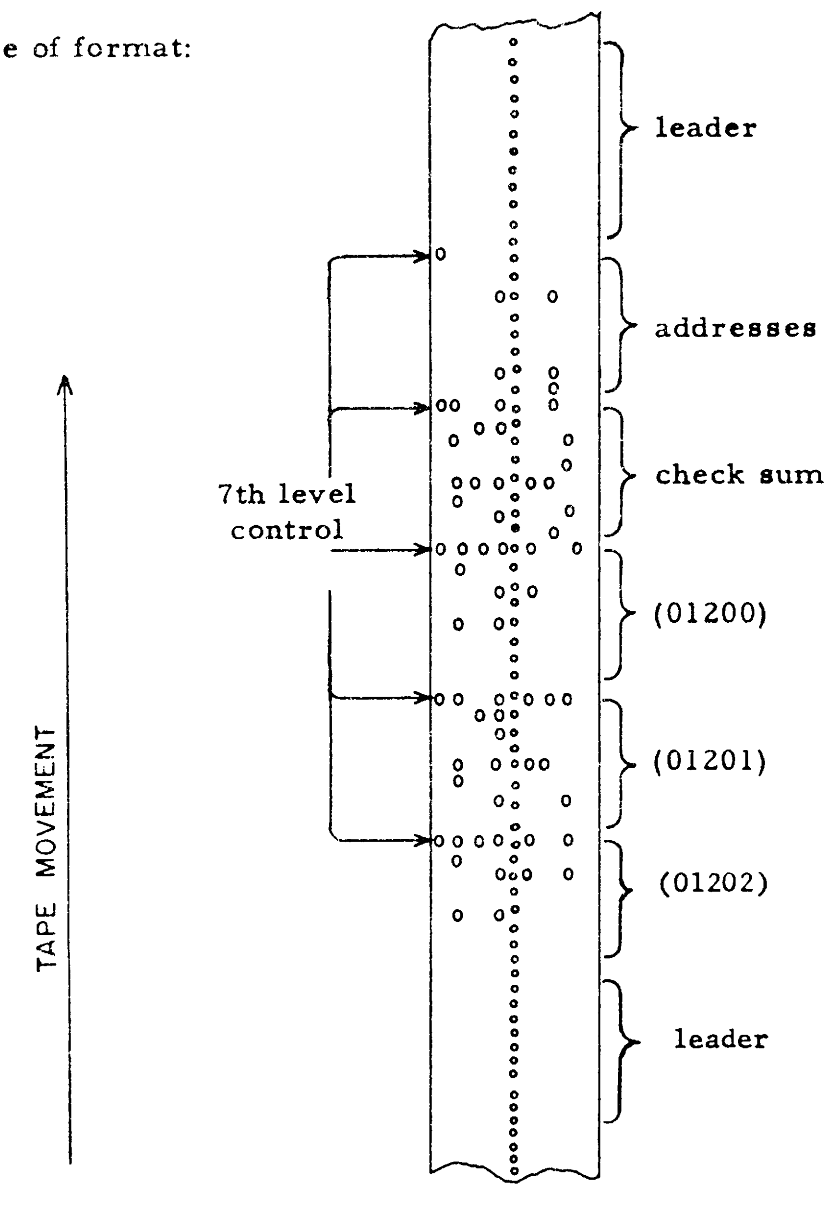
BIOCTAL DUMP

The Bioctal Dump routine punches a bioctally-coded paper tape of the contents of a specified area of core memory. This area is specified by entering the first address into index register 6 and the last address into index register 5. Each location of memory dumped occupies eight frames of paper tape (six bits per frame) and a seventh level hole is punched in the first frame of each group of eight frames. This seventh level hole is used for control in loading tapes.

The beginning and ending addresses of the area of core memory being dumped are first punched on the tape. This makes the tape suitable for reloading into the computer by use of the Bioctal Load routine. Following the area being dumped, a check sum of the contents of that area is punched. The actual contents of each storage location being dumped are then punched on the tape in consecutive order.

This routine also punches leader at the beginning and end of the punched area. There are no seventh level holes in the leader.

Sample of format:



Bioctal dump of addresses 01200 through 01202, containing the following:

```
(01200)  75  4  01400   50  0  00000
(01201)  57  3  01000   56  4  01100
(01202)  75  4  01500   50  0  00000
```

TAPE CONTROLLED LOAD

The Tape Controlled Load routine will load a paper tape into any area of core memory, providing the tape has been prepared in the correct format. It is impossible to load a tape in the same area in which the Tape Controlled Load routine is located, but the routine itself checks for this condition. The desired starting address is entered into index register 6.

Tapes to be loaded using this routine should be coded starting at address 00000. However, instead of typing the address on the tape, as is done for the Flex Load routine, a control digit precedes the information to be loaded into each address. This control digit is interpreted by the routine as an address modifier, so that when a tape is loaded starting at a base address (index register 6) the base address is added to each execution address which requires modification. Addresses are loaded consecutively starting with the base address, so there must be information on the tape for each address and it must be in consecutive order. The paper tape must be coded in flexcode. It may be prepared on the Flexowriter or by using the Tape Controlled Dump routine.

The control digits used are 0, 1, 2, 3, 4, and 7. The meanings of these digits are as follows:

- 0. do not modify
- 1. modify the upper address of this word
- 2. modify the lower address of this word
- 3. modify both addresses of this word
- 4. check sum (not to be loaded into computer)
- 7. end of tape, stop

Example of tape format:

```
1.  75 4 00066  50 0 00045
3.  55 1 00011  16 0 00122
0.  10 0 00045  50 1 00000
3.  20 1 00114  75 4 00075
2.  54 1 00004  75 0 00014
4.  56 7 00267  30 5 00302
7.
```

Using the above tape and setting index register 6 to 76400, the contents of addresses 76400 - 76404, after the tape has been loaded, will be:

```
(76400)  75  4  76466  50 0 00045
(76401)  55  1  76411  16 0 76522
(76402)  10  0  00045  50 1 00000
(76403)  20  1  76514  75 4 76475
(76404)  54  1  00004  75 0 76414
```

TAPE CONTROLLED DUMP

The Tape Controlled Dump routine will punch on paper tape the contents of a specified area of core memory in a format which may be used as input for the Tape Controlled Load routine.

Each execution address of each memory location in the specified area is examined to see whether it falls between the first and last address being dumped (inclusive). If neither address is in this area, the contents are punched on the paper tape preceded by a control digit of zero. This control digit indicates to the Tape Controlled Load routine that this word should be loaded as it is, without modification. If either address falls within the area being dumped, the word is further examined to see if either of the function codes is one which uses the execution address as an operand*, in which case the half word in which that function code occurs is not modified. If it is found that only the upper address of a word should be modified, the control digit for that word is a one; if only the lower address should be modified, the control digit is a two; and if both addresses should be modified, the control digit is a three. The first address of the area being dumped is subtracted from those addresses which should be modified before they are punched on the paper tape.

A check sum of the area being dumped is computed. This sum is punched at the end of the tape, preceded by a control digit of four. The four indicates to the Tape Controlled Load routine that the word following should not be loaded into the next location but should be used to compare with the check sum computed during the loading process.

The last entry on the tape, following the check sum, is a control digit of seven, which is merely a signal to stop. Each control digit is preceded by a carriage return. An extra carriage return is punched after every eight words.

*01, 02, 03, 04, 05, 06, 07, 10, 11, 34, 35, 50, 51, 54, 74, 0, 74.7.

TYPE TEXT

Type Text is a subroutine which may be used to type out error indications or other information for the benefit of the operator. The subroutine is entered by a return jump.

Flexcode for the information to be typed should be stored, eight codes per storage location, in consecutive addresses immediately following the address in which the return jump is located. The order in which the flexcode is sent to the typewriter is from left to right. A 77 code should be stored following the flexcode for the last character to be typed.

Any number of locations may be used for storage of flexcode. The Type Text subroutine modifies its own exit each time it uses one storage location packed with flexcode. In this way, when the 77 code is reached, control is returned to the main routine at the address following the one in which the 77 code is stored.

APPENDIX E

TRACE PROGRAM

GENERAL

The TRACE program monitors the execution of any other program by indicating the instruction location and the contents of A and Q, after execution of each instruction. Indication is primarily by means of a paper tape punched in Flexowriter code, but a printed copy can also be made at any time at the monitor typewriter, so that the progress of the monitored program can be observed.

A number of options are available, so that the program can be made to (1) either punch, or print and punch; (2) indicate either all instructions, or only those lying within certain pre-selected ranges; (3) indicate either the location and contents of A and Q, or just the location; and (4) either perform the functions described above, or indicate only locations of all executed jumps and their destinations.

To accomplish this, the TRACE program is first set up by loading the starting address of the program-to-be-monitored, along with the desired ranges of locations to be traced, into a "directory."

When the TRACE program is started, it extracts each instruction from the monitored program, enters it into a special "test block," executes it, and punches or prints certain indicators, in accordance with parameters in the directory. Also, if any of the input/output channels are activated by an instruction of the monitored program, printing or punching ceases until all previously activated channels are deactivated; this feature prevents interference with the use of the input/output equipment by the trace program, and thus permits the use of TRACE on any type of program, providing the monitored program does not destroy any of the addresses within TRACE.

DIRECTORY

The directory is punched on paper tape in Flexowriter code, in a format which is the same as that used for Flex-load. An example is shown in Table V 1.

The first word of the directory, to be loaded into address 77600, contains a "mode designator" and a starting address equal to the initial location of the program to be monitored. Succeeding words contain ranges, with the lower-value address in the left half, and the higher, in the right half. The directory is terminated by choosing the right-most address of the next word equal to zero. A maximum of 200 (octal) words can be used in the directory.

OPERATING INSTRUCTIONS

After TRACE and the program-to-be-monitored are loaded, load the directory via the Flex-load routine. Set the program address counter to 72000, and select the jump keys in accordance with the program being monitored.

TABLE VI
TYPICAL DIRECTORY

77600	77	7	00000	00	0	01000	Designator Starting Address
77601	00	0	01000	00	0	01377	Range
77602	00	0	01600	00	0	01607	Range
77603	00	0	01614	00	0	01614	Range
77604	00	0	02010	00	0	02014	Range
77605	00	0	00000	00	0	00000	End of Directory

Note: In word 77600, the left-most three octal digits (8^{15}, 8^{14}, 8^{13}) control the mode, and are selected as follows:

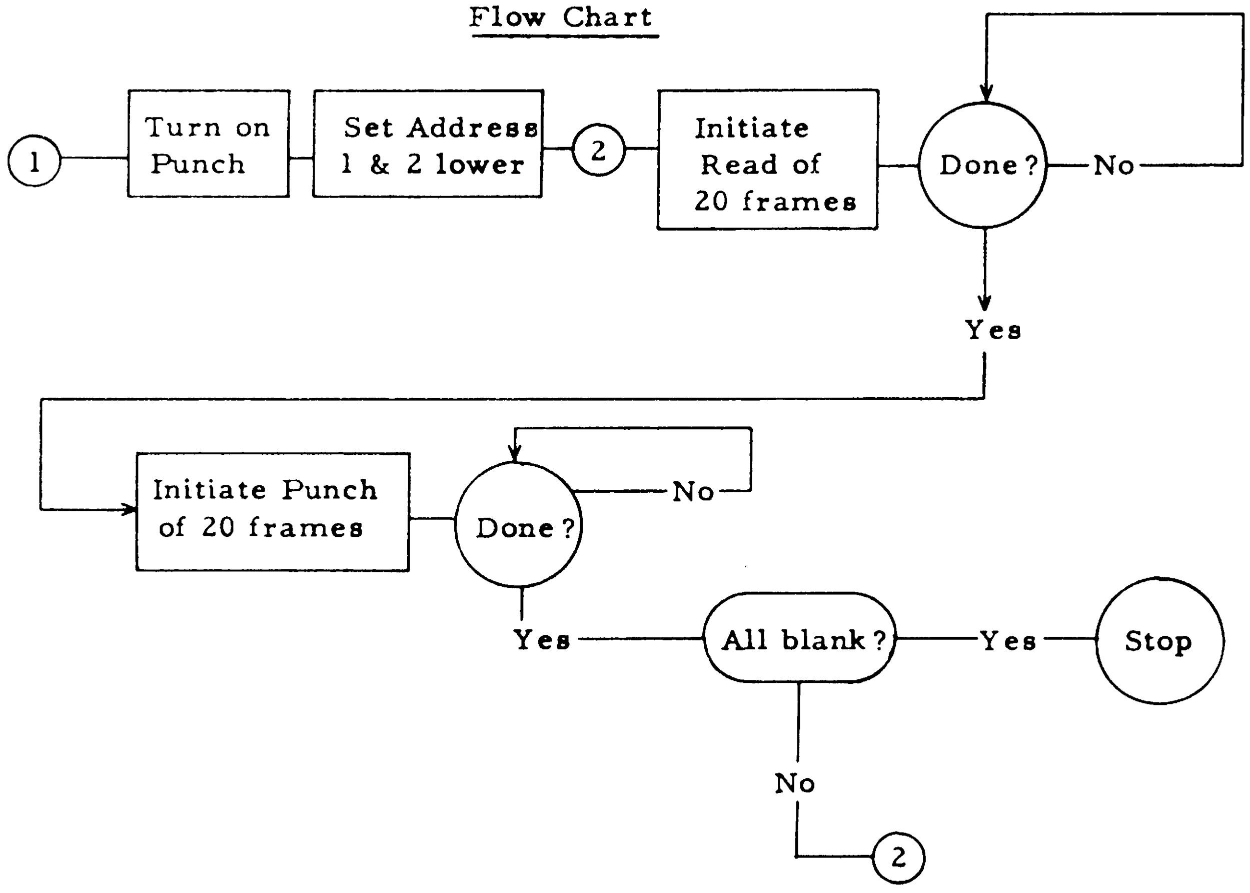
Digit 8^{15}	$= 0$	: Punches instruction location and contents of A and Q.
	$\neq 0$	: Punches only jump sources and destinations.
Digit 8^{14}	$= 0$	: Punches for each instruction.
	$\neq 0$	: Punches only for instructions within ranges specified by directory.
Digit 8^{13}	$= 0$	: Punches
	$\neq 0$	: Prints and punches.

APPENDIX F

Program for Duplicating Punched Tape

The operator places a tape in the reader with first punched data frame less than 20 (octal) frames ahead of the reading station. The program reads each group of 20 consecutive frames, and reproduces these on the high speed punch. As soon as 20 consecutive blank frames (trailer) have been punched, program stops.

Flow Chart



Program

```
(60000)  =  EXF   0   11211      Select and turn on punch motor
            ENI   6   60120      Set terminating address in $B^6$

(60001)  =  SIL   6   00001      Store Term. Address in lower (00001,
            SIL   6   00002      Store Term. Address in lower (00002)

(60002)  =  EXF   7   11211      Sense presence of char. mode; if not
            SLS   0   00000      present, stop.

(60003)  =  EXF   0   11200      Select Reader
            EXF   1   60100      Activate read, starting at 60100

(60004)  =  LIU   6   00001      Put machine count of words
            Pass                 read from 00001 (upper) into $B^6$

(60005)  =  ISK   6   60120      If ($B^6$) =Terminating Address, skip
                                 If ($B^6$) ≠Terminating Address, take N.I.
            SLJ   0   60004      Jump to test step again.

(60006)  =  EXF   0   11200      Select punch in character mode
            EXF   2   60100      Activate punch starting from (00100)

(60007)  =  LIU   6   00002      Put machine count of words punched
            Pass                 from upper (00002) in $B^6$

(60010   =  ISK   6   60120      If ($B^6$) =Terminal Address, skip
                                 If ($B^6$) ≠Terminal Address, take N.I
            SLJ   0   60007      Jump to test step again

(60011)  =  ENI   6   00000      Clear $B^6$
            Pass

(60012)  =  LDA   6   60100      Send word to A
            AJP   1   60000      If first word = 0, go back for more data

(60013)  =  ISK   6   00017      If ($B^6$)  = 17, skip, (all last words were
            SLJ   0   60012      If ($B^6$)  ≠ 17, take N.I.              zeros)

(60014)  =  SLS   0   60000      STOP
             0    0   00000
```

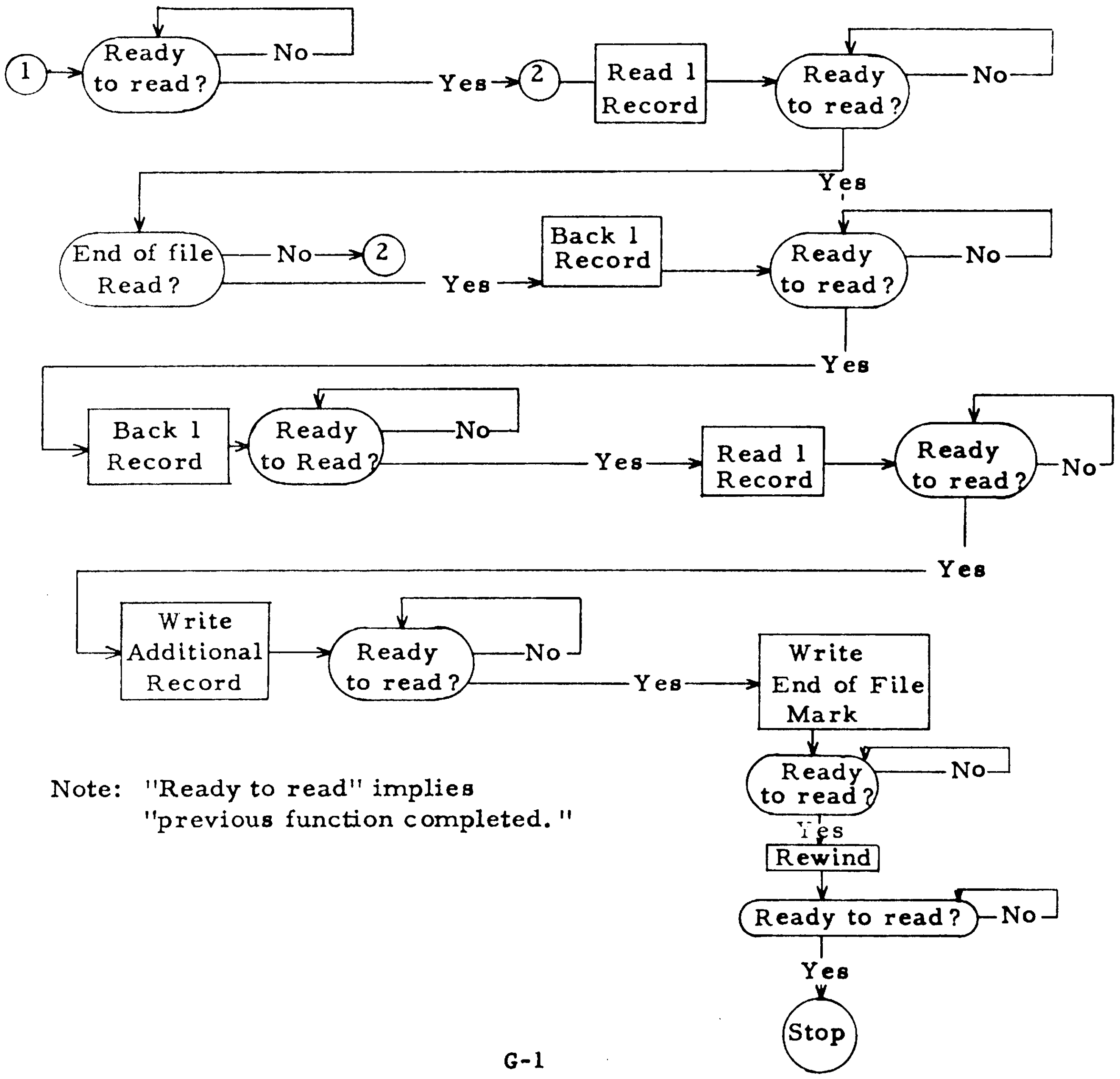
APPENDIX G

Program for Adding One Record to File

This program adds one record to the end of magnetic tape 1. It is assumed the end of data on tape is marked by an end-of-file mark. The program advances the tape past the end of file mark, backs up two records, advances one record and then writes the additional record from memory 60000-60177, and then writes a new end-of-file mark.

The reason for backing up two after reading end-of-file, then advancing by one, is so that writing shall follow a forward movement. If writing follows a back-up, especially over an end-of-file mark, the spacing between records will not be constant.

Flow Chart



Note: "Ready to read" implies "previous function completed."

program

```
(50000) = EXF    7    32010      Is mag. tape 1 ready to read?
          SLJ    0    50000      If not ready, wait.

(50001) = ENI    6    Temp + 1   Set B^6 to temp. address  1
          SIL    6    00003      Store term address in lower (0003)

(50002) = EXF    0    32010      Select Mag. Tape 1 to read
          EXF    3    Temp       Read one word to memory - tape
                                 moves one block.

(50003) = EXF    7    32010      Sense if ready to read again
          SLJ    0    50003      If not ready, wait.

(50004) = EXF    7    32012      Sense if end-of-file read
          SLJ    0    50001      If not end-of-file - read again.

(50005) = EXF    0    32011      Move back one record
          Pass

(50006) = EXF    7    32010      Sense, if ready to read
          SLJ    0    50006      If not ready, wait.

(50007) = EXF    0    32011      Move back one record.
          Pass

(50010) = EXF    7    32011      Sense, if ready to read
          SLJ    0    50010      If not ready, wait.

(50011) = ENI    6    Temp+1     Set B^6 to Term Address
          SIL    6    00003      Store term. add. in 00003.

(50012) = EXF    0    32010      Select mag. tape 1
          EXF    3    Temp       Activate read

(50013) = EXF    7    42012      Sense end-of-file on tape 1
          SLJ    0    50013      If not ready, wait.

(50014) = ENI    6    Rec+200    Set B^6 to term address
          SIL    6    00004      Enter term. add. in 00004.

(50015) = EXF    0    42010      Select Write
          EXF    4    Rec        Write activate

(50016) = EXF    7    42012      Sense end of write indicator
          SLJ    0    50016      If not ready, wait.

(50017) = EXF    0    42016      Write end-of-file mark on tape
          Pass
```

```
(50020) =  EXF      7      42012      Skip if ready
           SLJ      0      50020      Wait

(50021) =  EXF      0      32015      Rewind
           Pass

(50022) =  EXF      7      32010      Sense end of rewind
           SLJ      0      50022      Wait

(50023) =  SLS      0      50000      Stop
           0                   0
```

COMMENT SHEET

CONTROL DATA 1604 COMPUTER

Programming Training Manual

Pub. No. 60001500

FROM NAME : ________________________________

BUSINESS ADDRESS : ________________________________

COMMENTS: (DESCRIBE ERRORS, SUGGESTED ADDITION OR DELETION AND INCLUDE PAGE NUMBER, ETC.)

CUT ALONG LINE

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
8100 34TH AVENUE SOUTH
MINNEAPOLIS 20, MINNESOTA

ATTN: TECHNICAL PUBLICATIONS DEPT.
COMPUTER DIVISION
PLANT TWO

CUT ALONG LINE

FOLD FOLD

STAPLE STAPLE