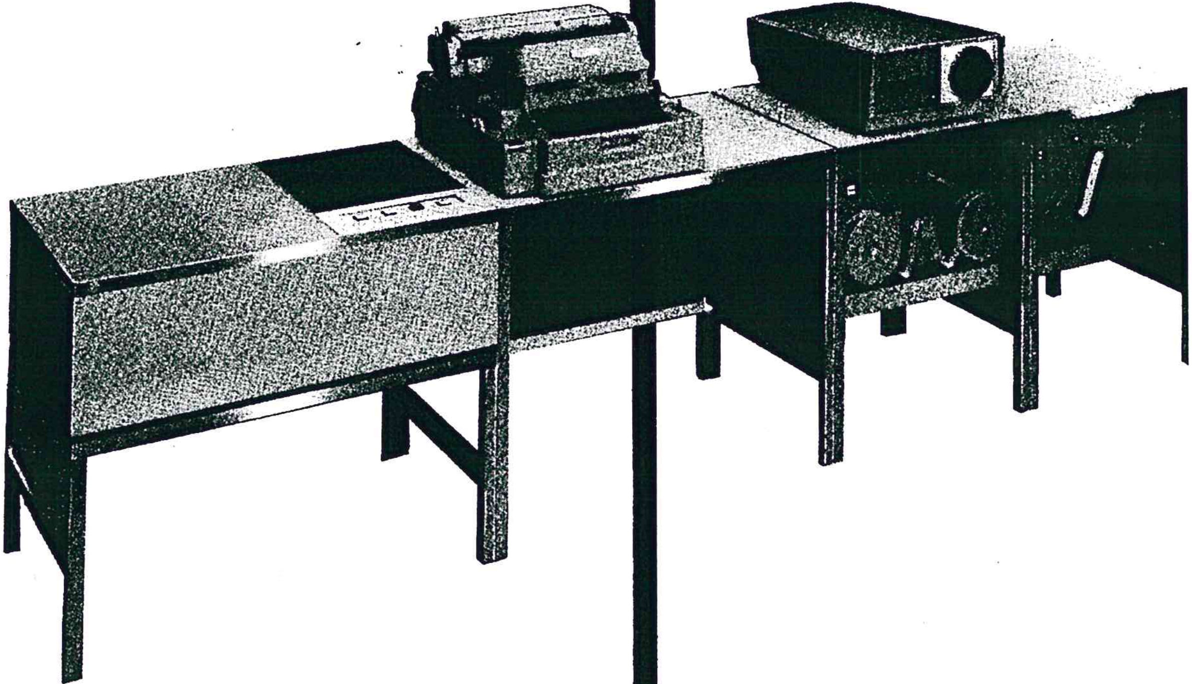
21

BRADFIELD



PROGRAMMING MANUAL

GENERAL PRECISION, INC.
Commercial Computer Division

L
G
P
2
1

PROGRAMMING MANUAL

for the **LGP 21** General Precision Electronic Computer

CONTENTS

		Page
1	INTRODUCTION TO THE LGP-21	1-1
	General	1-1
	Computer Elements	1-1
	Memory	1-2
	Arithmetic Unit	1-3
	Control Unit	1-4
2	THE LOP-21 COMMAND STRUCTURE	2-1
	Introduction	2-1
	LGP-21 Instructions	2-1
	Arithmetic Instructions	2-2
	A ADD	2-2
	D DIVIDE	2-3
	M MULTIPLY	2-3
	N MULTIPLY	2-4
	S SUBTRACT	2-4
	Logical Instructions	2-5
	E EXTRACT	2-5
	T CONDITIONAL TRANSFER	2-5
	-T TRANSFER CONTROL	2-6
	U UNCONDITIONAL TRANSFER	2-6
	Z STOP	2-7
	Z SENSE BS AND TRANSFER	2-8
	-Z SENSE OVERFLOW AND TRANSFER	2-8
	Manipulative Instructions	2-9
	B BRING	2-9
	C CLEAR	2-10
	H HOLD	2-10
	I 6-BIT SHIFT	2-10
	-1 4-BIT SHIFT	2-10
	R SET RETURN ADDRESS	2-11
	Y STORE ADDRESS	2-11
	Input/Output Instructions	2-11
	I 6-BIT INPUT	2-11
	-I 4-BIT INPUT	2-12
	P 6-BIT PRINT	2-12
	-P 4-BIT PRINT	2-12
3	CONSTRUCTION OF AN LGP-21 PROGRAM	3-1
	LGP-21 Coding Sheet	3-1
	The 4-Phase Instruction Cycle	3-3
	Sector Reference Timing Track	3-3
	Transfer Instructions	3-6
	Instruction Modification and Looping	3-8
	The Y Instruction	3-11
	Initialization	3-14
	Subroutine Concept	3-15

Page

4

BINARY NUMBER SYSTEM 4-1
Addition in Binary 4-3
Subtraction in Binary 4-3
Multiplication and Division in Binary 4-4
Negative Numbers 4-4
Instruction Words 4-5
Data Words 4-7
Binary Data 4-7
Binary-Coded Decimal Data 4-a
Alphanumeric Data 4-9
Hexadecimal Notation 4-10
Decimal to Hexadecimal Conversion 4-11
Hexadecimal Instruction Words 4-12

5

NUMBER SCALING FOR THE LGP-21 5-1
Scaling 5-1
Decimal Constants in a Program 5-3
The M and N Instruction 5-4
Variations of Z Instruction 5-4
Transfer Control Switch and the T Instruction 5-6
The E Instruction 5-6

6

INPUT/OUTPUT 6-1
Input Information 6-1
Character Representation on Tape 6-1
The Input Instruction 6-3
Non-Entering Characters 6-5
The Print Instruction 6-5
The Shift Instruction 6-6
Examples of Output Operations 6-6
Input to the Computer 6-8
Manual Input 6-8
Program-Controlled Input 6-9
Storing Information in Memory 6-9
Input to Memory from Typewriter 6-9
Input to Memory from Tape 6-10
LGP-21 Bootstrap 6-10

7

PROGRAM TAPE PREPARATION 7-1
Tape Preparation 7-1
Correction of Errors 7-2

8

TIMING AND OPTIMIZATION 8-1
Timing 8-1
Input Timing a-2
Output Timing 8-2
Optimization 8-2

Page

A

COMPUTERCONTROLPANEL	A-1
Mode	A-1
NORMAL	A-1
ONE OPERATION	A-1
MANUAL INPUT	A-1
FILL CLEAR	A-2
EXECUTE	A-2
TRANSFER CONTROL	A-2
BRANCH SWITCHES	A-2
POWER	A-2
I/O	A-2
STOP	A-2
START	A-2
INTERLOCK	A-2
RECORDENABLE	A-3

B

INPUT/OUTPUT UNIT CONTROLS	B-1
POWER ON-OFF	B-1
START COMP	B-1
MANUAL INPUT	B-1
CODE DELETE	B-2
TAPE FEED	B-2
MANUAL INPFJT LIGHT	B-2
PUNCH ON	B-2
STOP READ	B-2
START READ	B-2
COND STOP	B-2
PAPER GUIDE	B-2
TAB STOP	B-2
LEFT MARGIN STOP	B-2
AUTOMATIC CARRIAGE RETURN	B-3
PAPER SCALE	B-3
TYPE GUIDE	B-3
WRITING LINE	B-3
PAPER RELEASE	B-3
LINE SPACER	B-3
PLATEN	B-3
CARRIAGE RELEASE (Right and Left)	B-3
PLATEN KNOBS (Right and Left)	B-3
PLATENVARIABLE	B-4
MARGIN RELEASE	B-4
RIBBON POSITIONER	B-4
SPACE	B-4
COND STOP (')	B-4
TAB	B-4
COLORSHIFT	B-4
CAR RET	B-4
BACKSPACE	B-4
LOWER CASE, UPPER CASE	B-4
TAPE INTERLOCK	B-4
FEED KNOBS (Reader and Punch)	B-4

C

TABLE I COMMAND AND ADDRESS EQUIVALENCES	C-1
Table Ia Command Equivalences	C-1
Table Ib Address Equivalences	c-2
TABLE II POWERS OF 2	c-3
TABLE III INPUT/OUTPUT CODES	c-4
Table IIIa Input/Output Codes for the 121 Typewriter	c-4
Table IIIb Additional Input/Output Codes	c-5

ILLUSTRATIONS

		Page
FIGURE 1.1	LGP-21 Computer System	1-1
FIGURE 1.2	Track/Sector Numbering System	1-2
FIGURE 1.3	Memory Disc	1-3
FIGURE 1.4	Control Registers	1-4
FIGURE 2.1	Word Structure	2-1
FIGURE 3.1	LGP-21 Coding Sheet	3-2
FIGURE 3.2	Sector Reference Timing Track	3-3
FIGURE 3.3	Instruction Cycle Phase 1	3-4
FIGURE 3.4	Instruction Cycle Phase 2	3-5
FIGURE 3.5	Instruction Cycle Phase 3	3-5
FIGURE 3.6	Instruction Cycle Phase 4	3-6
FIGURE 3.7	Typical Flow Chart	3-15
FIGURE 4.1	Instruction Word Format	4-5
FIGURE 4.2	List of LGP-21 Commands	4-5
FIGURE 4.3	Data Word Format	4-7
FIGURE 4.4	Hexadecimal Equivalences	4-10
FIGURE 4.5	Hexadecimal Designation of Commands	4-13
FIGURE 5.1	Extended Accumulator	5-4
FIGURE 6.1	Character Representation on Tape	6-2
FIGURE 6.2	List of Tape Codes	6-2
FIGURE 6.3	LGP-21 Special Character Codes	6-3
FIGURE 6.4	Basic Bootstrap	6-11
FIGURE 6.5	Decimal Coding for Hexadecimal Fill Sequence	6-12
FIGURE 6.6	Hexadecimal Fill Sequence	6-13
FIGURE 8.1	Optimum Timing Requirements	8-1
FIGURE 8.2	Range of Optimum Sectors	8-2
FIGURE 8.3	Optimum Address Locator	8-3
FIGURE A. 1	LGP-21 Control Panel	A-1
FIGURE B. 1	Model 121 Tape Typewriter	B-1

GENERAL

General Precision's LGP-21 was designed as a compact: mobile computerwith particular application potential to the problems of the small businessman, engineering firm, or scientific research group. The LGP-21 is not merely economical to own, but simple to program and operate. These skills can be acquired by qualified personnel within a two-week training course which is provided at no charge by General Precision, Inc. This programming manual is provided as an adjunct to the regular programming class, and should serve as a useful reference text thereafter.

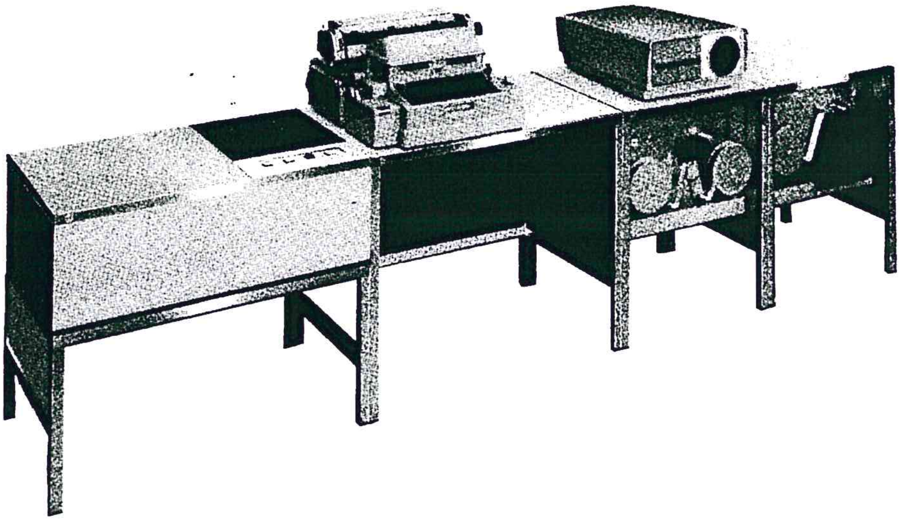


FIGURE 1 .1 LGP-21 Computer System

When the new LGP-21 programmer has finished his course, he will also be aware of the fact that a large library of programs and subroutines is available to assist him in his programming tasks. However, a discussion of the program library falls beyond the scope of this manual and should be conducted by each customer with his General Precision salesman/analyst.

COMPUTER ELEMENTS

A number of computer elements are of particular interest to the programmer as they provide for the storage and manipulation of information. They are the memory, arithmetic and control units and will be discussed below with particular emphasis upon their function in the program-execution process.

Memory

The LGP-21 memory unit is a disc with a total information storage capacity of 4096 computer words. For programming purposes, these words are considered as stored on 64 tracks in main memory, each track being divided into 64 sectors or storage locations. Both tracks and sectors are numbered from 00 through 63. This constitutes a simple means of locating information in memory: the combination of a word's track and sector number provides the address of the computer word. For example, the address of a word stored in Track 17, Sector 05, is "1705"; while the address 0261 refers to Track 02, Sector 61. There is no break in continuity of addresses from one track to the next, or from one sector to the next. Thus, consecutive addresses in computer memory can be said to range from 0000, 0001, 0002.. . .0063, into the next track, 0100, 0101.. . . etc. through 6363. . . .after which the next address would be 0000 again.

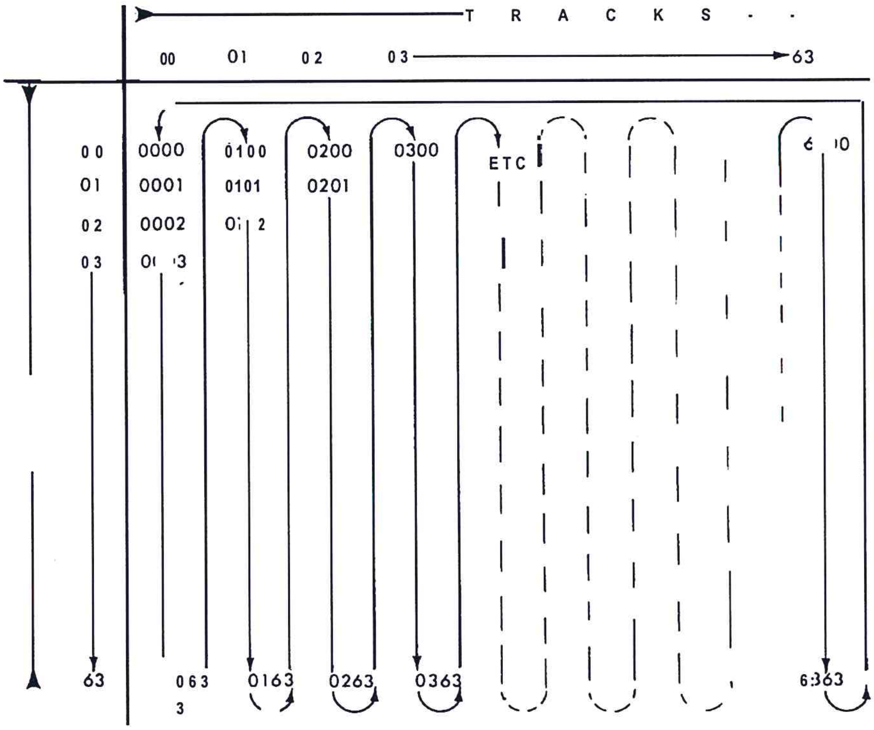


FIGURE 1.2 Track/Sector Numbering System

Mounted above the surface of the memory disc are 32 read/write heads which "read" or "write" information into the various memory locations as the disc revolves. Each head serves two tracks which are assigned alternate sectors in a circle. Thus, read/write head 0 reads the first sector of Track 00, then the first sector of Track 01; then the second sector of Track 00, and the second sector of 01, etc. Read/write head 1 serves Tracks 02 and 03; head 5 serves tracks 10 and 11, and so on.

It should be mentioned at this point, that the engineering characteristics of the memory disc are disregarded for most programming purposes. One exception, optimizing, represents a programming refinement which may be of limited interest to most LGP-21 programmers. It is therefore ignored for the time being, and will not be discussed until the end of the manual, in Chapter 8. Other exceptions will be pointed out as they occur. Suffice it to say that while the memory disc actually consists of 32 tracks with 128 sectors each, it will be treated as a 64 track/64 sector unit for most programming purposes. This concept makes the LGP-21 fully compatible with other General Precision computers with which the programmer may already be familiar.

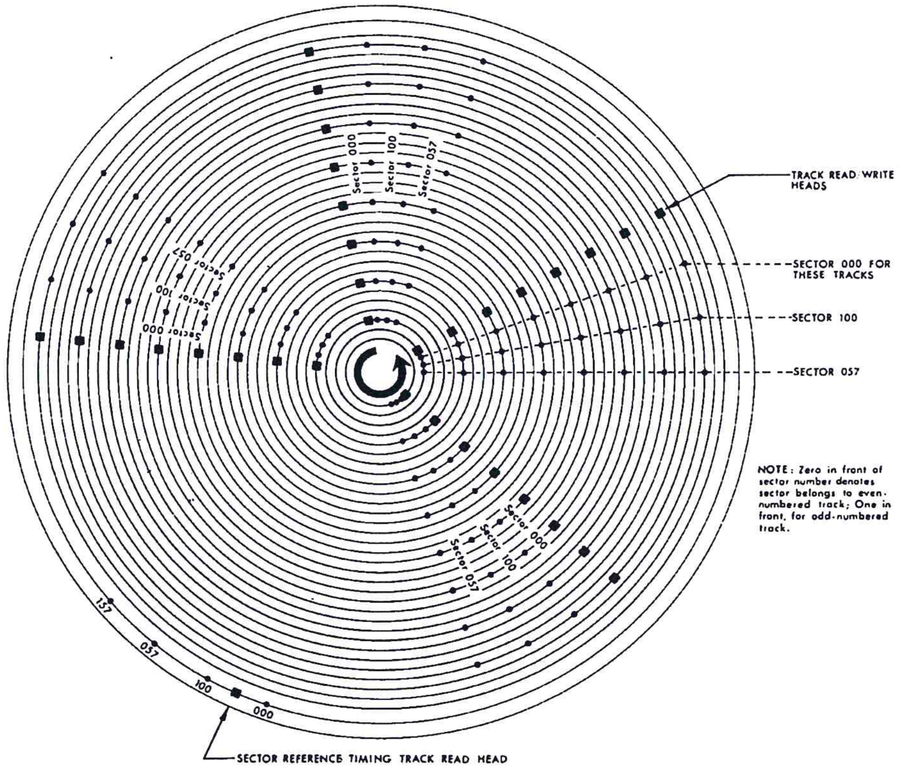


FIGURE 1.3 Memory Disc

Arithmetic Unit

All internal computations are performed in the arithmetic unit of the LGP-21. It consists of the Accumulator (A), and the Extended Accumulator (A*), which are recirculating lines on the memory disc.

The Accumulator (A) is a working register which is used for all manipulation of data. Through it passes all information which is transferred from one part of memory to another. Prior to the execution of an arithmetic operation, the Accumulator holds one of the operands (the other is stored in memory); following the execution of the instruction, the Accumulator holds the result of the arithmetic operation. In addition, all communications between the computer and its input/output devices pass through this register.

The Accumulator has one read- and one write-head, located one word-time apart. They continually copy information from one sector into the next, making the same word constantly available. On the same track is a two-word recirculating line, the Extended Accumulator (A*). It is not addressable by programming, but contains the intermediate results during multiplication and division operations. The track on which the recirculating lines are recorded is not one of the 64 tracks of main memory.

Control Unit

The control unit directs the operations of the computer. It consists of two registers: the Instruction Register (I) and the Counter Register (C). Each is a one-word recirculating line located on the same track as the Accumulator and the Extended Accumulator.

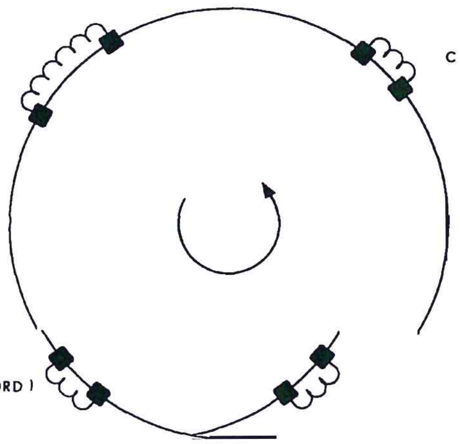


FIGURE 1.4 Control Registers

The Instruction Register(I) holds whatever instruction is to be executed. The two exceptions to this rule are the Multiply and divide instructions which depend upon continuous availability of the operand. To provide such continuous access, the multiplier or divisor -not the instruction — is copied from memory into the Instruction Register.

The Counter Register (C) contains the address of the next instruction to be executed. In other words, if the Counter Register reads 2438, it means that the next instruction to be executed is in Track 24, Sector 38. This register also holds the overflow indicator. (Overflow will be discussed in Chapter 6).

With this basic understanding of the various computer elements which are involved in the manipulation of information for the LGP-21, it is now possible to approach the actual programming procedures for the computer.

THE *LGP-21* COMMAND *STRUCTURE* 2

INTRODUCTION

Generally stated, programming is the process by which problems are put into a form which a computer can handle. Since the computer can only calculate numerical answers to numerical problems, the programmer has to formulate all problems in this form and replace non-numerical problems with equivalent numerical ones.

Calculations on numerically-stated problems involve the use of basic arithmetic operations; i. e . , addition, subtraction, etc. These operations are initiated by a set of commands which are easily remembered as they bear a mnemonic relationship to familiar operations, such as "A" signifying Add, "D" Divide, etc. In all, the LGP-21 responds to 23 basic commands or orders concerning arithmetic, logical, manipulative, and input/output operations.

It was mentioned before that the LGP-21 memory disc has 4096 sectors in which information may be stored. The information unit which is stored in a sector is called a computer word and may consist of data or an instruction (Figure 2.1).

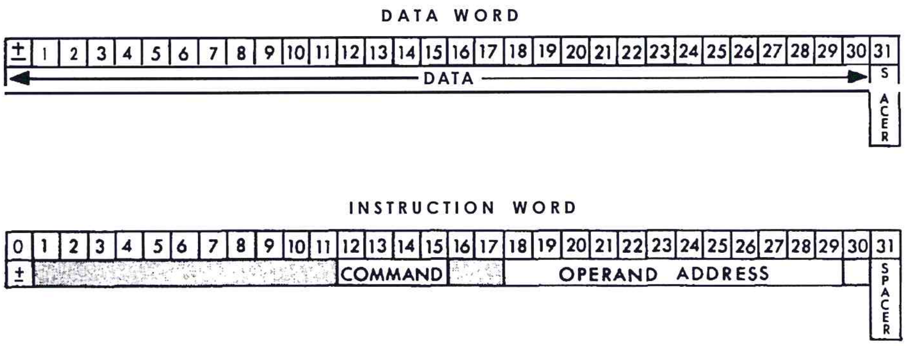


FIGURE 2.1 Word Structure

An LGP-21 word consists of 32 binary digits or "bits" which are used to represent decimal numbers or alphabetic symbols as combinations of l's and 0's. The presence of a bit in a computer word is represented as a 1, the absence of a value as 0. Since the computer performs all internal information manipulation in binary form, the programmer must acquire some familiarity with binary arithmetic. However, this chapter will be concerned only with the decimal representation of instruction words, and with a basic understanding of their functions. The binary number system will be discussed in Chapter 4.

LGP-21 INSTRUCTIONS

The programmer uses an instruction to tell the computer what operation it must perform. Each instruction is composed of two significant parts which identify the instruction as such to the computer: the command part which specifies the type of operation (add, multiply, etc.), and the address of the operand in track-and-sector notation.

Each command is assigned one of sixteen alphabetic characters. A few of these characters are used to represent two different functions. As a means of distinguishing between these alternate functions, the character is preceded by a minus sign for one operation, and by no sign for the other. When such an instruction is entered into the computer, however, it is recorded as follows: a minus instruction is recorded with a 1 in bit position zero; a non-minus instruction is recorded with a 0 in bit position zero of the word.

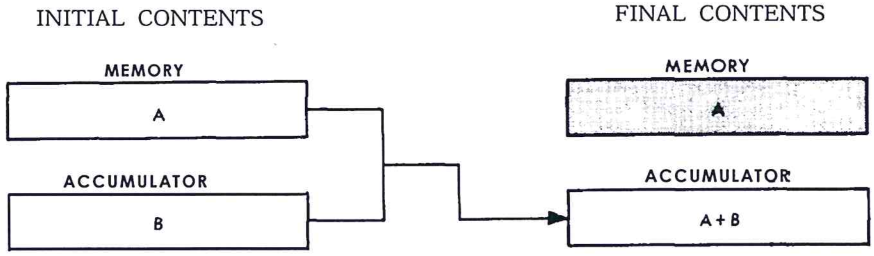
The two significant parts of an instruction word are recorded in the following positions: the command in bit positions 12 through 15, the operand address in bit positions 18 through 29. The operand address, furthermore, consists of a track address (bits 18 through 23), and the sector address (bits 24 through 29). For instructions whose command portion calls for a transfer of data, the operand address specifies the memory location from which the data is brought to the Accumulator, or in which the data is stored from the Accumulator. For arithmetic operations this address specifies the memory location of the second operand (the first operand must be in the Accumulator).

A typical instruction would be A 1532; that is, the instruction to the computer to ADD the contents of Location 1532 to whatever is in the Accumulator at the time the instruction is issued. This would be stored as follows: a 0 bit in the zero position of the computer word, to indicate that this is not a negative instruction; the binary equivalent of "A" (add) in bit positions 12 through 15; and the binary equivalent of the address 1532 in positions 18 through 29 of the instruction word. The unfilled bit positions 1 through 11, 16 and 17, and 30 and 31 are ignored by the computer when it executes an instruction. Actually, bit position 31 is always recorded in memory as a "0" as it serves to separate computer words. It is called the "spacer bit".

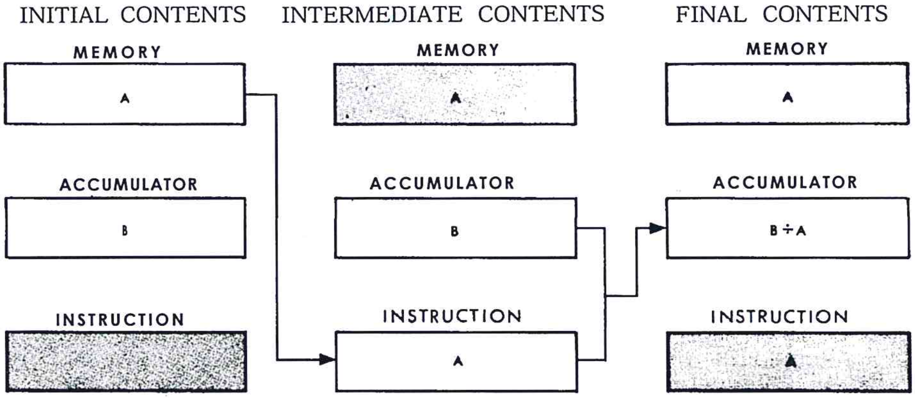
The four groups of instructions — arithmetic, logical, manipulative, and input/output — are summarized below in the following manner: the first column, headed "Order, " gives the alphabetic designation of the command; the "Address" column contains an "m" or "n", where "m" represents any one of the 4096 memory locations and "n" represents a value rather than an address. The "Interpretation" column explains the function of each instruction.

Arithmetic Instructions

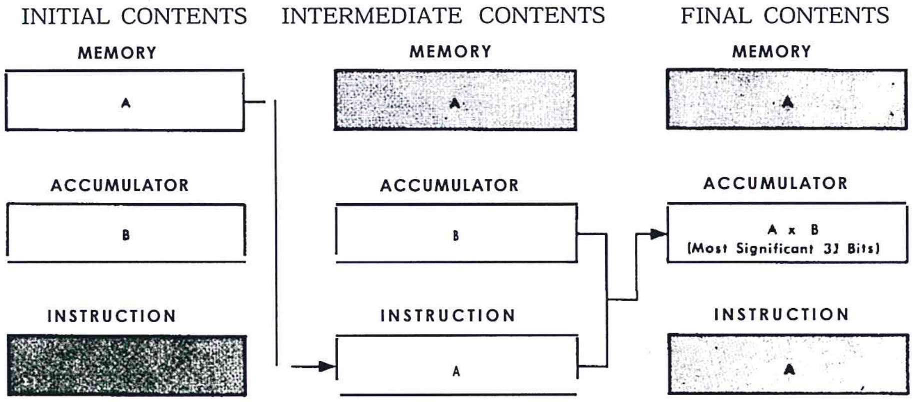
		Order	Address	Interpretation
A	ADD	A	m	ADD--Add the contents of location m to the contents of the Accumulator. The sum replaces the contents of the Accumulator. If an addition results in a number beyond the limits of the Accumulator, overflow will occur. The contents of m remains unaltered.



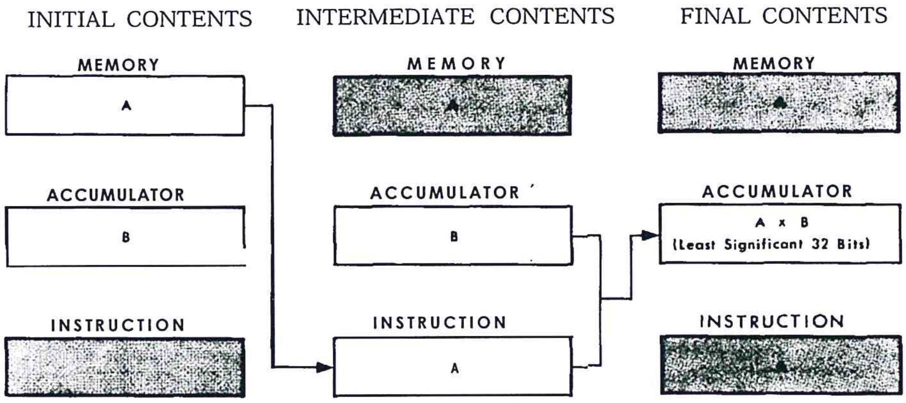
	Order	Address	Interpretation
D DIVIDE	D	m	DIVIDE--Divide the number in the Accumulator by the number in location m, retaining the quotient, rounded to 30 bits, in the Accumulator. The absolute value of the contents of m must be greater than the absolute value of the contents of the Accumulator, or overflow will occur. During the divide operation the Instruction Register holds the divisor. m remains unaltered.



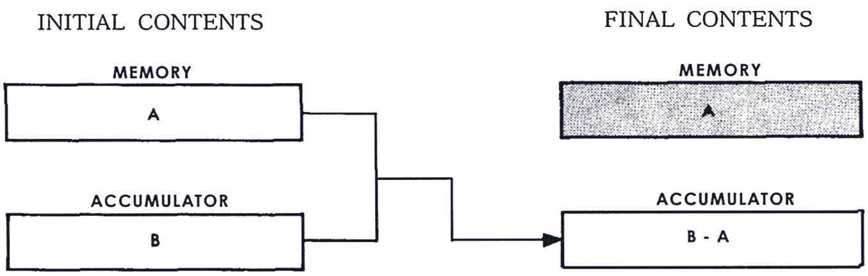
	Order	Address	Interpretation
M MULTIPLY	M	m	MULTIPLY--Multiply the contents of the Accumulator by the contents of location m, forming a 62-bit product of which 31 bits are retained: the sign and the most significant 30 bits of the product replace the contents of the Accumulator. The Instruction Register holds the multiplicand during the multiply operation. Memory remains unaltered.



	Order	Address	Interpretation
N **MULTIPLY**	N	m	MULTIPLY--Multiply the contents of the Accumulator by the contents of location m, forming a 62-bit product of which 31 bits are retained: the least significant 31 bits replace the contents of the Accumulator, occupying bit positions 0 through 30. Loss of any of the most significant bits does not cause overflow. During the multiply operation, the Instruction Register holds the multiplicand. Memory remains unaltered.

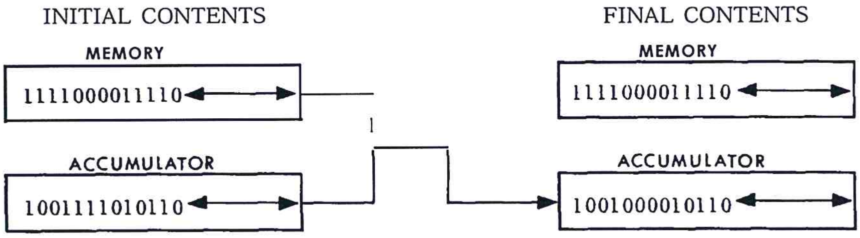


	Order	Address	Interpretation
S **SUBTRACT**	S	m	SUBTRACT--Subtract the contents of location m from the contents of the Accumulator and retain the difference in the Accumulator. If a subtraction results in a number beyond the limits of the Accumulator, overflow will occur. Memory remains unaltered.

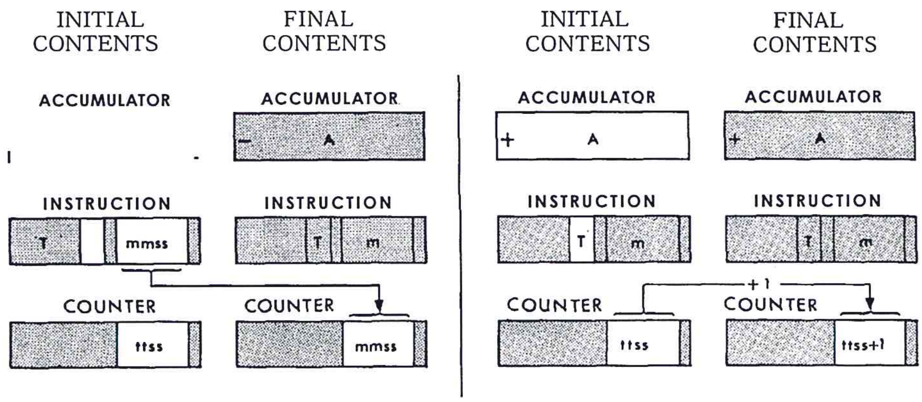


Logical Instructions

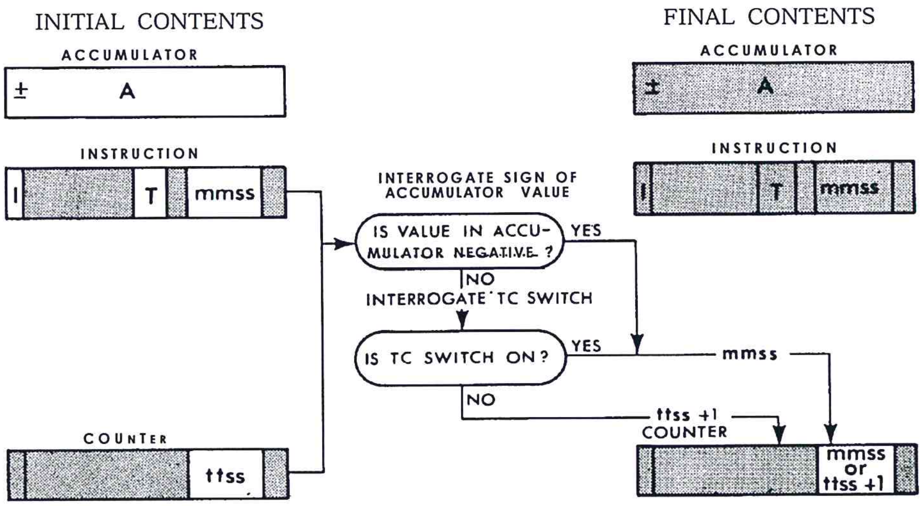
		Order	Address	Interpretation
E	**EXTRACT**	E	m	EXTRACT--Where "1" bits are in location m, retain the value of the corresponding bit positions in the Accumulator; where "0" bits are in m, place 0 bits in the corresponding positions in the Accumulator. The word in location m is called the "mask" and remains unaltered.



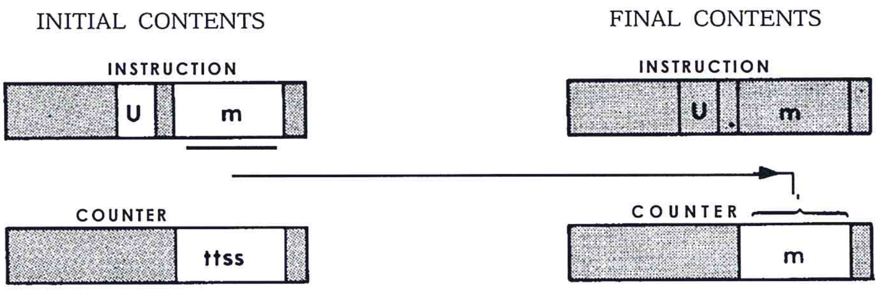
		Order	Address	Interpretation
T	**CONDITIONAL TRANSFER**	T	m	CONDITIONAL TRANSFER--If the contents of the Accumulator is negative (1 in the sign position), replace the contents of the address portion of the Counter Register with m and get the next instruction from location m. If the contents of the Accumulator is positive, continue to the next instruction in sequence without altering the Counter.



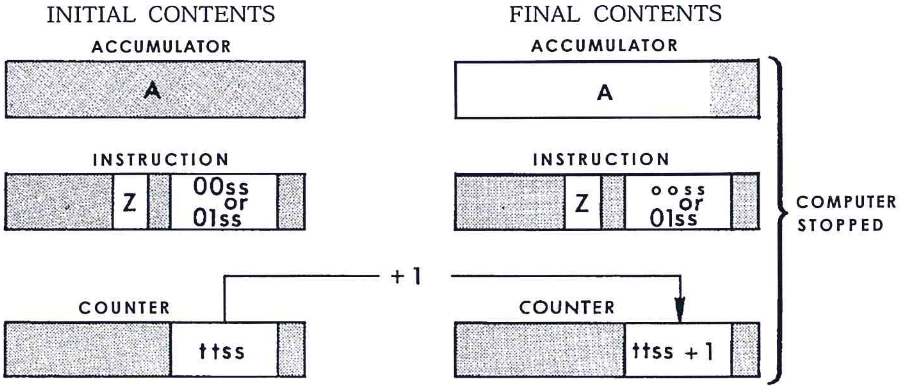
		Order	Address	Interpretation
-T	**TRANSFER CONTROL**	-T	m	TRANSFER CONTROL-If the contents of the Accumulator is negative, or if the TC switch on the console is ON, replace the contents of the address portion of the Counter Register with m and get the next instruction from location m.



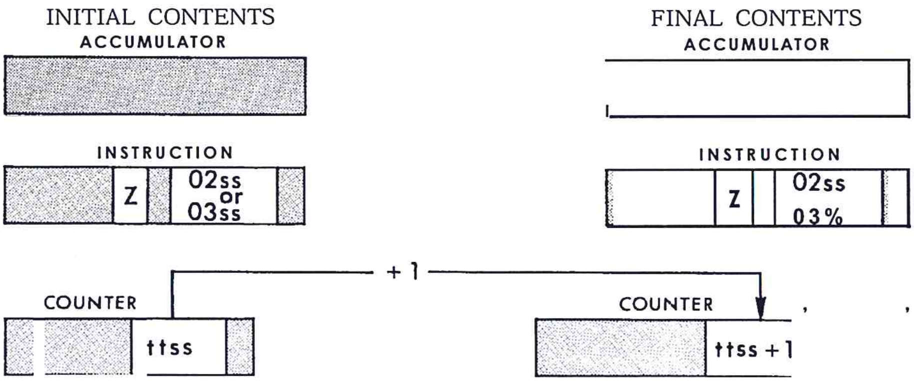
		Order	Address	Interpretation
U	**UNCONDITIONAL TRANSFER**	U	m	UNCONDITIONAL TRANSFER--Replace the contents of the address portion of the Counter Register with m and get the next instruction from location m.



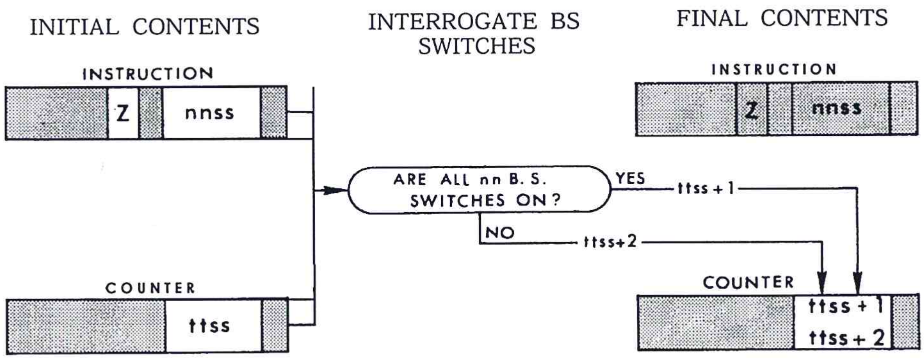
	Order	Address	Interpretation
STOP **Z**	Z	n	STOP--When n = 0000 or 0100, halt computation.



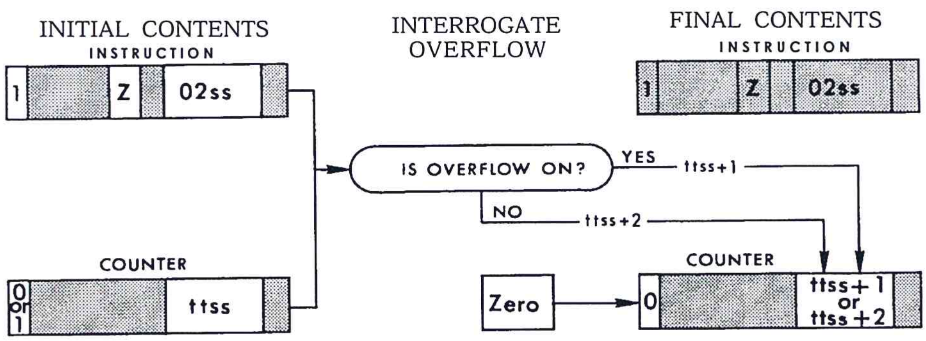
When n = 0200 or 0300, no operation occurs; i.e., the computer does not halt, the contents of the Accumulator remains unchanged, and nothing in memory is altered.



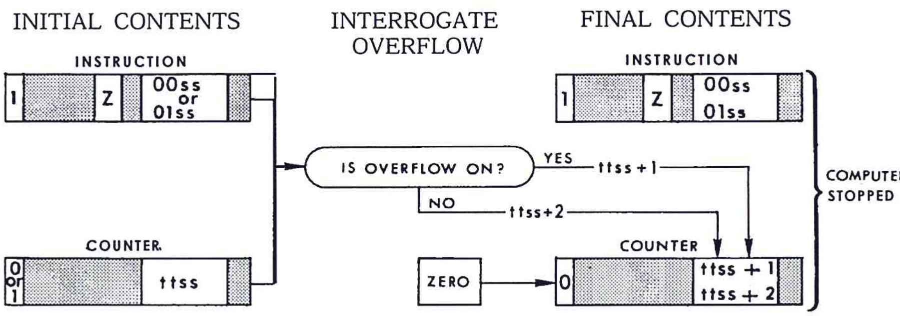
		Order	Address	Interpretation
Z	**SENSE BS AND TRANSFER**	Z	n	SENSE BS AND TRANSFER--Interrogate **the Branch Switches** specified by the track portion of n **(3<n≤63).** If all of the specified Branch Switches are ON, the next sequential instruction will be executed. If any of them is OFF, the next instruction will be skipped. The Branch Switches are numbered 4, 8, 16 and 32.



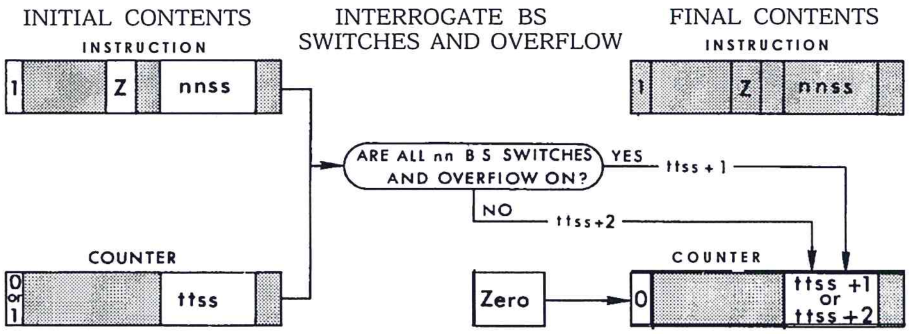
		Order	Address	Interpretation
- z	**SENSE OVERFLOW AND TRANSFER**	**-Z**	n	SENSE OVERFLOW AND TRANSFER--If overflow is OFF (0 in the sign position of the Counter Register), skip the next instruction in sequence. If overflow is ON (1 in the sign position of the Counter), reset the overflow bit to zero; then execute the next instruction. The track portion of n designates which, if any, Branch Switches are also to be interrogated.



If Sense Overflow is combined with Stop (-ZOOOO), the skip or no skip is deferred until after the stop. If no Branch Switches are to be tested and no stop is desired, the track address can be 02 or 03.

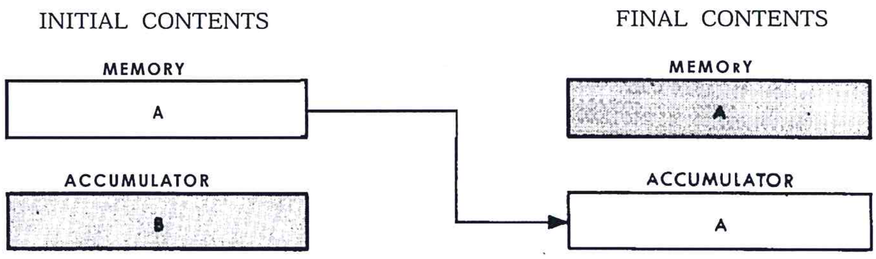


Overflow and/or any combination of Branch Switches can be interrogated with one Sense and Transfer instruction.

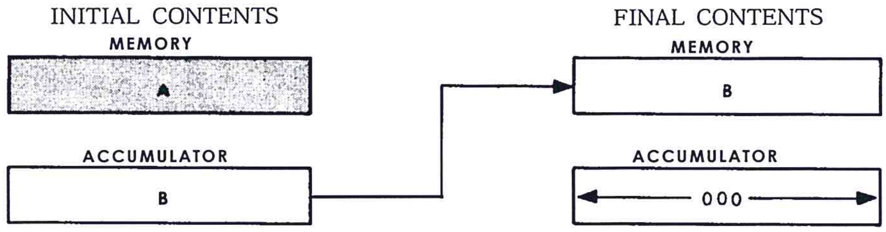


Manipulative Instructions

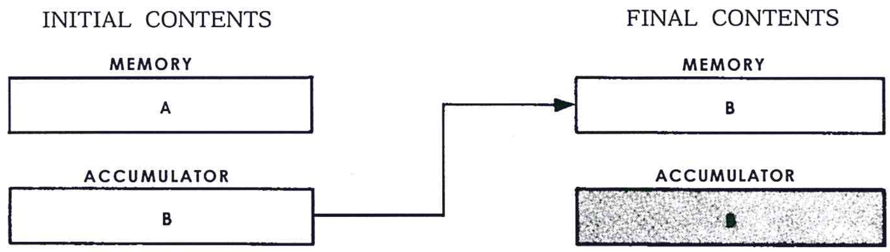
		Order	Address	Interpretation
B	BRING	B	m	BRING--Bring the contents of location m into the Accumulator, replacing its contents. Memory remains unchanged.



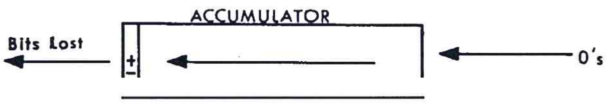
		Order	Address	Interpretation
C	**CLEAR**	C	m	CLEAR--Store the contents of the Accumulator into memory location m; then clear the Accumulator to zero.



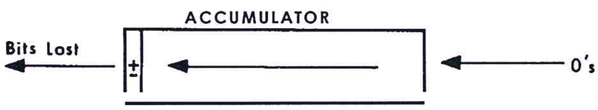
		Order	Address	Interpretation
H	**HOLD**	H	m	HOLD--Store the contents of the Accumulator into location m, without altering the contents of the Accumulator.



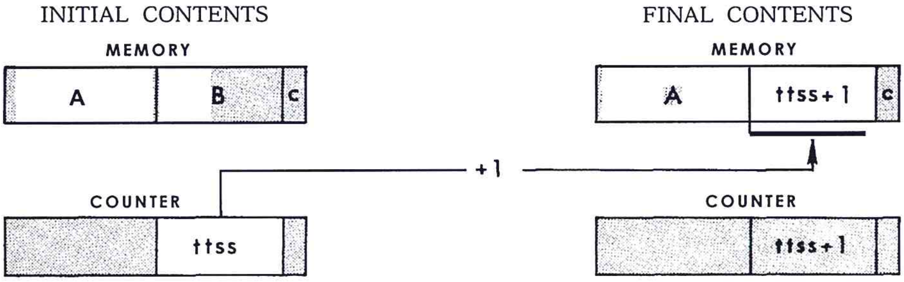
		Order	Address	Interpretation
I	**6-Bit SHIFT**	I	n	6-BIT SHIFT--When n-6200, shift the contents of the Accumulator left 6 places, inserting zeros at the right.



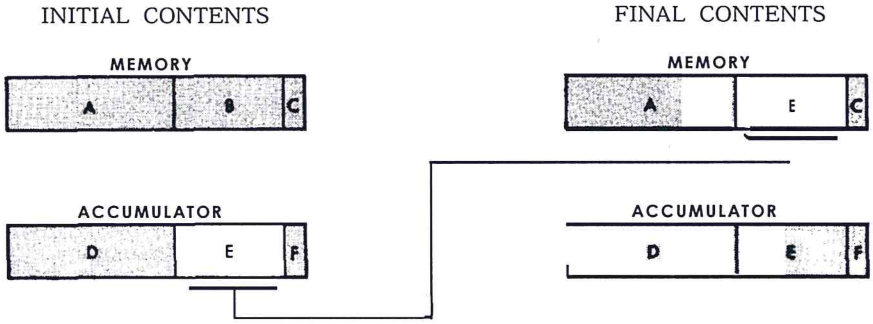
		Order	Address	Interpretation
-I	**4-Bit SHIFT**	-1	n	4-BIT SHIFT--When n=6200, shift the contents of the Accumulator left 4 places, inserting zeros at the right.



		Order	Address	Interpretation
R	**SET RETURN ADDRESS**	R	m	SET RETURN ADDRESS--In the address portion of location m, record the address which is 2 greater than the location of the I instruction being executed (i. e. , the contents of the Counter Register plus 1).

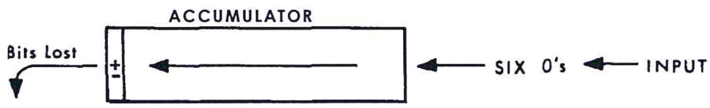


		Order	Address	Interpretation
Y	**STORE ADDRESS**	Y	m	STORE ADDRESS--Replace the address portion of the word in location m with the address portion of the word in the Accumulator, leaving the rest of m and all of the Accumulator undisturbed.

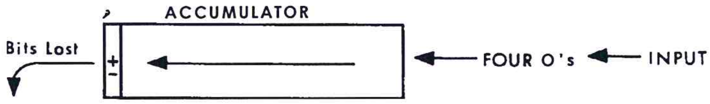


Input/Output Instructions

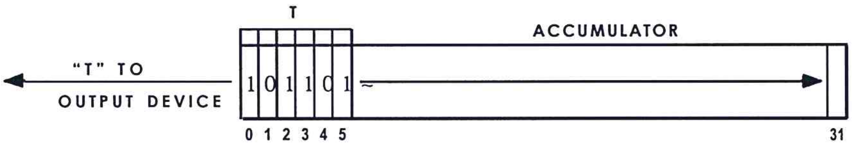
		Order	Address	Interpretation
I	**6-BIT INPUT**	I	n	6-BIT INPUT--Shift the contents of the Accumulator left 6 places, inserting zeros at the right. Then give a start read signal, allowing 6 bits of each character read by the input device specified by n to enter the Accumulator. A character enters the low-order (right) end of the Accumulator, shifting the previous contents of the register toward the high-order end. Once input is initiated, characters will be shifted into the Accumulator (and out the left end if too many are entered) until input is terminated.



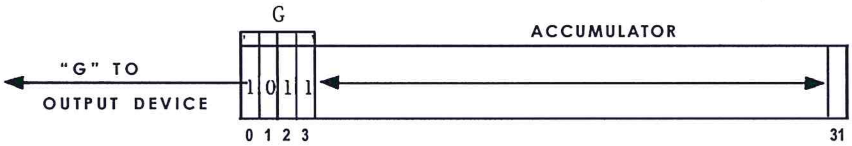
			Order	Address	Interpretation
-I	**4-Bit**	**INPUT**	-I	n	4-BIT INPUT--Shift the contents of the Accumulator left 4 places, inserting zeros at the right. Then give a start read signal, allowing the 4 bits of each character read by the input device specified by n to enter the Accumulator. A character enters the low-order (right) end of the Accumulator, shifting the previous contents of the register toward the high-order end. Once input is initiated, characters will be shifted into the Accumulator (and out the left end if too many are entered) until input is terminated.



			Order	Address	Interpretation
P	**6-Bit**	**PRINT**	P	n	6-BIT PRINT--Transmit the character represented by bits 0 through 5 of the Accumulator to the output device specified by n. The contents of the Accumulator remains unaltered.



			Order	Address	Interpretation
-P	**4-Bit**	**PRINT**	-P	n	4-BIT PRINT--Combine "1" for channel 5 and "0" for channel 6 with bits 0 through 3 from the Accumulator, then transfer this character to the output device specified by n. The contents of the Accumulator remains unaltered.



CONSTRUCTION OF AN LGP-2 7 PROGRAM 3

A computer program consists of a series of step-by-step instructions from the programmer to the computer. To illustrate the basic concept, the following steps would have to be specified to solve the problem below:

$$\left[\frac{(7+8)}{3}\right]9 \; -6 = x.$$

As explained in Chapter 2, "A" is the alphabetic symbol for addition, "M" for multiplication, "D" for division, and "S" for subtraction. According to the definition, each instruction must consist of a command portion which identifies the operation to be performed and an address. Therefore, assuming that the numbers 7, 8, 3, 9, and 6 are stored in memory in locations 0300, 0301, 0302, 0303, and 0304 respectively, the program would look like this:

Step	Order	Address	Notes	
1	B	0300	Bring the number 7 to the Accumulator.	
2	A	0301	Add 8	$7 + 8 = 15$
3	D	0302	Divide by 3	$\frac{7+8}{3} = 5$
4	M	0303	Multiply by 9	$(\frac{7+8}{3})$
5	S	0304	Subtract 6	$((\frac{7+8}{3})\ 9)\ -6$
6	H	0305	Hold the answer in 0305	
7	Z	0000	stop	

It is important to clearly understand the distinction between the address of a memory location and the contents of that location. An address, such as 0300, refers to a place on the disc, while contents refers to the word recorded at that place.

LGP-21 CODING SHEET

Programs are usually written on LGP-21 Coding Sheets. The sample below (Figure 3.1) shows the general format and explains in detail the purpose of the seven columns provided.

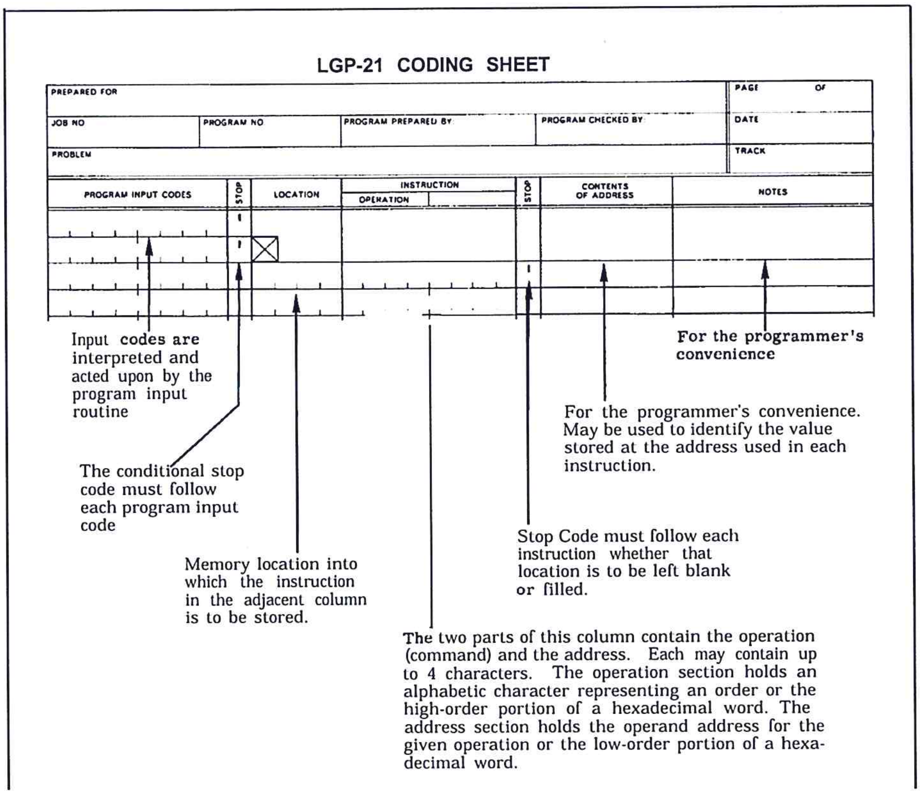


FIGURE 3.1 LGP-21 Coding Sheet

The last column, "Notes", should be used to provide all the necessary explanatory information which will be helpful for subsequent reading of a program. The programmer will find it very useful to develop the habit of providing such information.

Anything written in parenthesis on the coding sheet should be read as "the contents of"; an arrow as "replace"; and the abbreviation "Acc. " will be used for "Accumulator. " For example, (m) is to be read "the contents of memory location m, " and (m)⟶(Acc.) is to be read "the contents of memory location m replaces the contents of the Accumulator. " This notation will be used throughout the manual.

If the example problem were written on a coding sheet, with the instructions to be stored in locations 1000 through 1006, it would appear as follows:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		1000	B	0300	'	7	(0300) → (Acc.) = 7
		1001	A	0301	'	8	7 + (0301) → (Acc.) = 15
		1002	D	0302	'	3	15 ÷ (0302) → (Acc.) = 5
		1003	M	0303	' ☒	9	5 x (0303) → (Acc.) = 45
		1004	S	0304	'	6	45 - (0304) → (Acc.) = 39
		1005	H	0305	'		(Acc.) → (0305)
		1006	Z	0000	'	STOP	

THE 4-PHASE INSTRUCTION CYCLE

At the start of an operation, the computer memory must contain the data to be processed, and the instructions which tell the computer what operations to perform on these data. Ignoring, for the moment, how this information is initially entered into the computer, it need merely be remembered here that any memory location may be used to store one instruction word or one data word. To start execution of the instructions, the programmer specifies the storage location of the first instruction to be executed. After it is found and operated on, the computer automatically takes all successive instructions from sequential memory locations (e.g. , if execution starts at Location 1400, the next instruction will be taken from 1401, then 1402, etc.). The time required for completing a specified operation depends, in part, on the location in memory of the instruction and of its operand, if one is necessary. The process by which the computer obtains and executes an instruction is called an instruction cycle. An instruction cycle begins with a memory search for the instruction word and ends with the commencement of the search for the next instruction word.

The complete cycle consists of four phases:

Phase 1 - Search for the instruction.

Phase 2 - Transfer the instruction from main memory to the Instruction Register and increment the Counter Register by 1.

Phase 3 - Search for the operand.

Phase 4 - Execute the instruction.

SECTOR REFERENCE TIMING TRACK

In order for the computer to find a specific location in memory, a Sector Reference Timing Track is used. This track contains the sector numbers 00 through 127 permanently pre-recorded at the time of manufacture. As explained in Chapter 1, there are actually 32 concentric circles on the disc which are divided into 128 sectors each. However, for programming purposes, sector addresses are numbered 00 through 63. Therefore, on the Sector Reference Timing Track numbers greater than 63 are interpreted modulo 64. For example, sector 97 on the Sector Reference Timing Track represents sector 33 for odd-numbered tracks (i.e. 97 - 64 = 33).

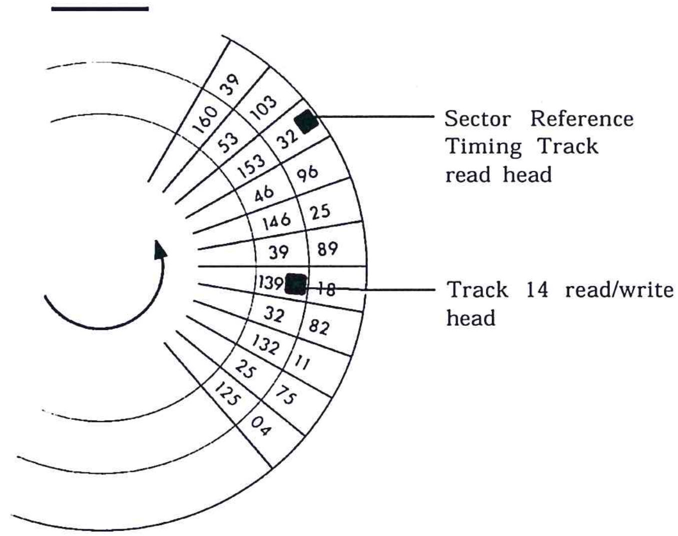


FIGURE 3.2 Sector Reference Timing Track

The Sector Reference Timing Track (Figure 3.2) has only a read-head and cannot be modified by the programmer. The numbers on this track pass under its read-head one sector before the corresponding sector in main memory does. Thus, when a specified sector address is read on the Sector Reference Timing Track, the read/write head on the appropriate track is activated, and the word can be read from or recorded in memory. For example, assume the contents of Location 1432 is to be brought to the Accumulator. Because Track 14 is even-numbered, the Sector Reference Timing Track searches for sector 32. When it is read, read-head 7, which serves Tracks 14 and 15, is activated; and as sector 32 moves under that read-head, its contents is copied into the Accumulator.

This sequence of actions may be more easily understood if two instructions are considered in terms of the instruction cycle. For example:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
		☒					
		1115	B	4458	'	8	8 (Acc.)
		1116	A	4452	'	7	8 + 7 (Acc.)

During Phase 1 the Counter Register contains the address 1115. Since 11 is an odd-numbered track, the computer searches the Sector Reference Timing Track for sector 79 (79 - 64 = 15). When it is read, the read-head 5, serving Tracks 10 and 11, is activated, and Phase 1 ends (Figure 3.3).

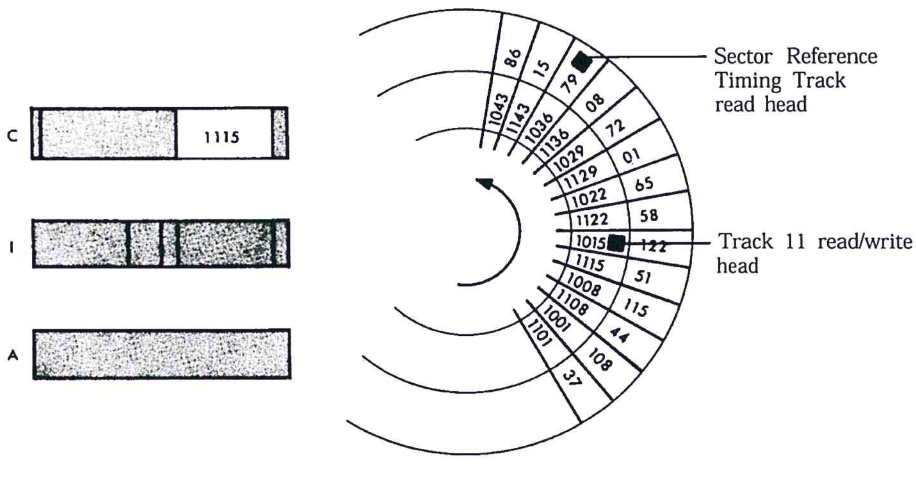


FIGURE 3.3 Instruction Cycle Phase 1

In Phase 2 the contents of Location 1115 is copied into the Instruction Register, and the Counter Register is incremented by 1, so that it now contains 1116 (Figure 3.4).

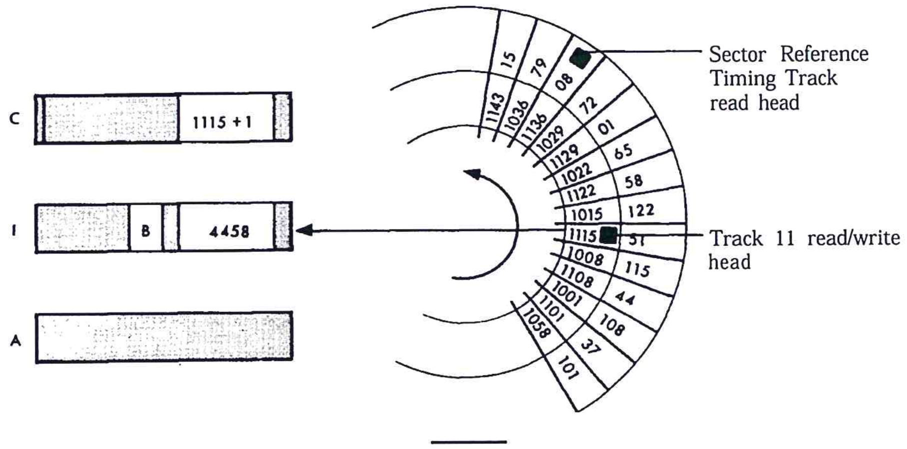


FIGURE 3.4 Instruction Cycle Phase 2

During Phase 3 the computer searches the Sector Reference Timing Track for the operand sector specified in the Instruction Register- that is, sector 58 (Figure 3.5).

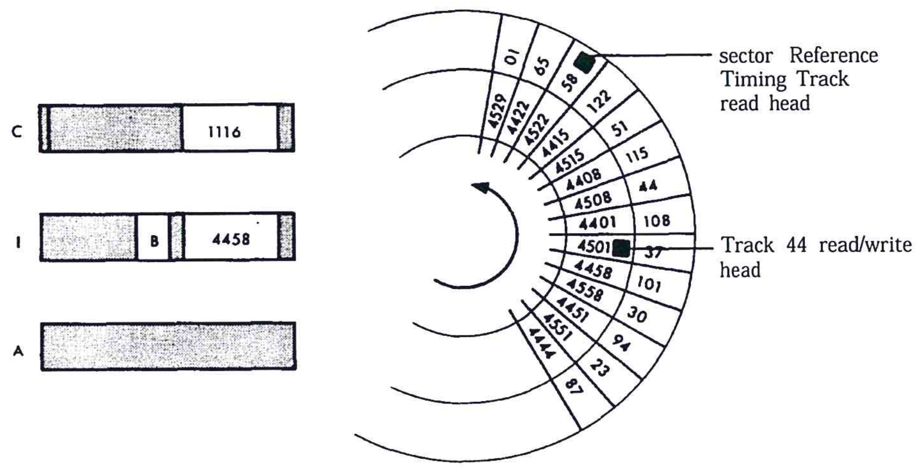


FIGURE 3.5 Instruction Cycle Phase 3

When sector 58 is read, Phase 3 ends, and the computer goes to Phase 4 (Figure 3.6) to execute the instruction B4458. Therefore, the contents of Location 4458 (the number 8) is copied into the Accumulator.

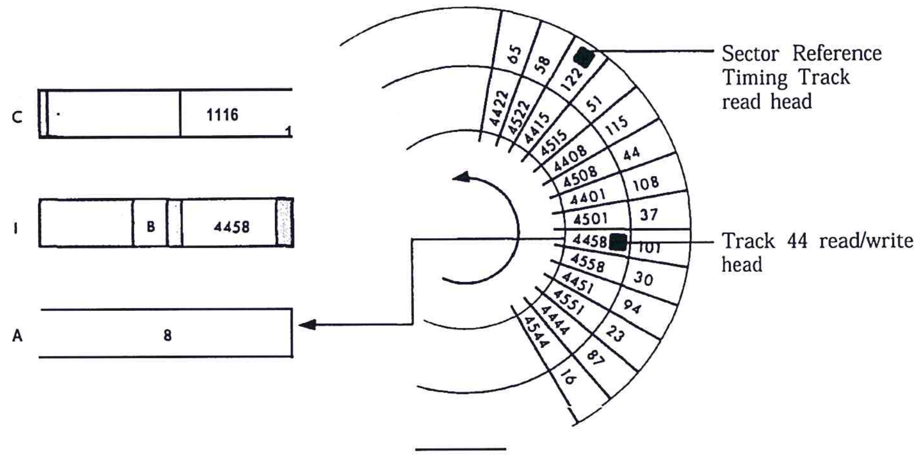


FIGURE 3.6 Instruction Cycle Phase 4

Then the cycle begins again:

Phase	Activity
1	Counter Register contains 1116, therefore search for sector 80 on the Sector Reference Timing Track. When sector 80 is found, activate read-head 5 for Track 11.
2	Copy contents of Location 1116 (A4452) into the Instruction Register. Increment the Counter Register by 1 to 1117.
3	Search for sector 52. When it is read, activate read-head 22 for Track 44.
4	Execute the instruction; that is, add the contents of 4452 (the number 7) to the contents of the Accumulator (8) and leave the result (15) in the Accumulator.

The minimum time required for a complete 4-phase cycle is 18 word-times. A "word-time" is the time required for one word to pass under the read/write head. Since the disc revolves at approximately 1180 rpm, a word-time takes approximately .40 millisecond for the LGP-21. The maximum time for the 4-phase cycle is 146 word-times (one disc revolution plus 18 word-times). Program execution time can be minimized by selecting operand addresses according to a special method which is called "optimizing." This process is explained in Chapter 8. However, optimization is not considered in most of the examples given in this manual.

TRANSFER INSTRUCTIONS

When the computer has to take the next instruction from some location other than the next one in sequence-that is, execute a branch-two types of instructions may be used: an unconditional or a conditional transfer instruction.

The Unconditional Transfer instruction, Urn, tells the computer to branch unconditionally to location m to obtain the next instruction, instead of going to the next location in sequence. After this transfer, the sequential mode is resumed, starting at location m, until another transfer instruction is encountered. If the U instruction is regarded in terms of the 4-phase cycle, it can be explained as follows:

Instruction	Explanation
U	m ⟶ (Counter)

Instructions are executed in Phase 4. If m replaces the contents of the Counter in Phase 4, the computer will go to m during Phase 1 of the next 4-phase cycle to obtain the next instruction. The contents of the Counter is replaced by the address portion of the U instruction which is in the Instruction Register during Phase 4. (Note: It is not the contents of m which replaces the contents of the Counter.)

The Conditional Transfer instruction, Tm, tells the computer to branch to location m to obtain the next instruction only if the Accumulator contains a negative word; otherwise, to go to the next location in sequence for the next instruction. If the transfer takes place, the sequential mode is resumed, starting at m, until another transfer instruction is encountered. If the T instruction is thought of in terms of the 4-phase cycle, the instruction can be explained as follows:

Instruction	Explanation
T	If the Accumulator contains a negative word, m ⟶(Counter); if the Accumulator contains a positive word (Counter) remains unchanged. In either case (Acc.) remains unchanged.

Phase 3 for U and T instructions is a dummy phase, as a memory search for an operand is unnecessary in conjunction with these two instructions.

Consider a problem using these transfer instructions. The problem requires one of two calculations to be made-the choice depending upon the sign of a certain number. B, C, D, and E are given, and the problem is stated as follows:

If B is positive, calculate $\left[\frac{B}{C}\right]D$ = answer

If B is negative, calculate $\frac{B+E}{C}\ D$ = answer

Data Storage

Location	Data
0300	B
0301	C
0302	D
0304	E
0400	Answer

The coding for this problem follows:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		1040	B	0300	'	B → (Acc.)	
		1041	T	1046	'	Test B for	positive or negative
		1042	D	0301	'	(Acc.) ÷ C →	(Acc.)
		1043	M	0302	' ☒	(Acc.) x D →	(Acc.)
		1044	H	0400	'	(Acc.) → 0	400
		1045	Z	0000	'	HALT	
		1046	A	0304	'	(Acc.) + E →	(Acc.)
		1047	U	1042	' ☒	Branch back	to complete calculations
					'		

The T1046 instruction in Location 1041directs the computer to 1042 for the next instruction if B is a positive number. If B is a negative number, the computer branches to Location 1046 to obtain the next instruction. Starting at 1047 it is necessary to execute the same instructions which are in Locations 1042 through 1045; to avoid repeating these instructions, a U1042 instruction in Location 1047 is used to transfer back to them.

INSTRUCTION MODIFICATION AND LOOPING

As already explained, the instruction to be executed is transferred to the Instruction Register during Phase 2 and executed during Phase 4. It should be noted that the computer can only interpret a word as an instruction word when it is in the Instruction Register. An instruction word in any other place is interpreted as a data word. This makes it possible to manipulate instruction words as if they were data words. For example, using the appropriate sequence of instructions, one can bring an instruction word into the Accumulator, modify it in some way (possibly by adding some constant to it), and hold the modified instruction back in its original location. The computer is unaware that it is actually processing an instruction word. The modified instruction word will not be interpreted as an instruction until it is transferred to the Instruction Register during Phase 2 of some subsequent 4-phase cycle; and this will not occur until the address of this instruction is in the Counter Register during Phase 1 of the subsequent 4-phase cycle. This LGP-21 feature—internally stored program operation which permits modification of instructions—can be a very useful programming aid.

Consider this problem:

128 numbers are stored in Locations 0300 through 0463 (Tracks 03 and 04). Compute their sum and store the result in Location 0500.

This problem could be solved by bringing the first of above numbers into the Accumulator with a Bring instruction, then adding the other 127 numbers by using 127 Add instructions, and finally storing the result as specified. This would be a tedious way to code the problem, though it would be a possible approach. However, the program can be reduced to a few instructions by using the instruction modification feature. The coding would be as follows:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0008	C	0500	'		0→(Acc.)
		0009	C	0500	'	Sum	0→(Acc.)
		0010	B	0500	'	Bring sum to Acc.	
		0011	A	0300	' ☒	Add the next number	
		0012	H	0500	'	Hold the sum in 0500.	

The first instruction will be stored in Location 0008. This is possible since the computer will start execution of the program at any location specified by the programmer.

The first CO500 instruction places whatever is in the Accumulator into 0500 and creates a zero in the Accumulator. The second CO500 instruction (which could just as well be H0500) sets the sum in 0500 to zero. The next 3 instructions bring the sum (zero at this time), add to it the first number (which is in 0300), then hold this answer back in 0500 as the new sum. The next sequence of instructions must effect (1) the address modification of the A0300 instruction in Location 0011, (2) a branch back to Location 0010 to repeat the sequence, and (3) a means of terminating the repetition. This process is called "looping". Thus, the instruction in Location 0011 can be changed to A0301 for the next time it is executed; then changed to A0302, etc. There must be control over the number of times that the instruction is modified and the loop repeated; then an exit from the loop can be made after the 128 numbers have been summed.

Continuing with the coding, the instructions in Locations 0013 through 0015 accomplish the modification of the A0300 instruction in Location 0011:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0008	C	0500	'	Zero→(Acc)	
		0009	C	0500	'	Store zero in 0500 [the sum]	
		0010	B	0500	'	Bring the sum to the Acc.	
		0011	A	0300	' ☒	Add the next number	
		0012	H	0500	'	Hold the sum in 0500	
		0013	B	0011	'	Bring the instruction to be modified to	
					'	the Acc.	
		0014	A	0019	' ☒	Add Z 0001 to the instruction	
		0015	H	0011	'	Hold modified instruction 0011	
		:			'		
		:			'		
		0019	Z	0001	' ☒	Constant used in address modification	
					'		

The BOO11 instruction in Location 0013 brings into the Accumulator, from Location 0011, the instruction to be modified. Now arithmetic operations can be performed on this instruction word as if it were a data word. In the Accumulator is the instruction word A0300, to which another word must be added, so that the instruction word A0301 will be obtained as the result. The A0019 instruction in Location 0014 accomplishes this by adding the contentsof Location 0019 to the contents of the Accumulator and leaving the sum in the Accumulator. This addition takes place:

A 0300 - Initial contents of Accumulator
+ Z 0001 - Plus contents of Location 0019

A 0301 - Final contents of Accumulator

(A "Z" in the command portion of an instruction is treated as a zero by the computer.)

Thus, when the computer is ready to execute the instruction in Location 0015, the Accumulator contains the instruction word A0301. The HO011 instruction in 0015 places the contents of the Accumulator in Location 0011. Therefore, the A0300 instruction in Location 0011 has been replaced by the instruction A0301.

When the sum of the 128 numbers has been accumulated in Location 0500, the program must exit from the loop. The instructions in Locations 0016 and 0017 enable the program to determine whether the loop is to be repeated or terminated:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0008	C	0500	'	Zero the Accumulator	
		0009	C	0500	'	Store zero in 0500 (the sum)	
		0010	B	0500	'	Bring the sum to the Acc.	
		0011	A	[0300]*	'	☒ Add the next number	
		0012	H	0500	'	Hold the sum in 0500	
		0013	B	0011	'	Bring the instruction to be modified to	
					'	the Acc.	
		0014	A	0019	'	☒ Add Z 0001 to the instruction	
		0015	H	0011	'	Hold modified instruction in 0003	
		0016	S	0020	'	Subtract A 0500 from the instruction	
		0017	T	0010	'	Return to beginning of loop if (Acc.)	
					'	☒ negative	
		0018	Z	0000	'	HALT	
		0019	Z	0001	,	Constant use in address modification	
		0020	A	0500	'	Constant use to test for end of loop	
					,	☒	
					,		
					,		

Before the SO020 instruction in 0016 is executed, the Accumulator contains the A instruction which has just been held in 0011 by the instruction in 0015. What is the address portion of the A instruction now in the Accumulator? It depends on how many times the loop, extending from 0010 through 0017, has been executed. If it has been executed once, the A instruction reads A 0301; if twice, A 0302 and so on. If the loop has been executed 128 times, the instruction reads A0500. The following example shows that the subtraction will yield a negative result whenever the A instruction has an address portion less than **0500:**

A XXXX
-A 0500

Result: Some negative number for any XXXX $<$0500

* It is good practice to enclose an address with brackets to indicate that it will be modified during the execution of the program. The brackets have no other significance, but make it easier for a programmer to follow the program.

The A instruction is in the Accumulator and in Location 0011 before the SO020 instruction is executed. Whenever this A instruction has an address portion less than 0500, the result of the SO020 instruction in 0016 will be a negative word in the Accumulator. The TOO10 instruction in 0017 will then branch to the beginning of the loop at Location 0010. At the start of the 128th execution of the loop, the A instruction in 0010 will be A0463. Therefore, the instructions from 0010 through 0012 will add in the last number and hold the sum in 0500. The instructions from 0013 through 0015 will modify the instruction in 0011 to read A0500 and leave this instruction in the Accumulator. The 50020 instruction in 0016 will subtract A0500 from this instruction word. For the first time the result will be positive (zero). Therefore, rather than branching back to the beginning of the loop, the TOO10 instruction will allow the computer to exit to the instruction in location 0018, a halt. At this time 0500 will contain the sum of the 128 numbers stored in 0300 through 0463.

A question on elementary arithmetic might have occurred to the reader. If the above program is to work correctly, the following answer must result from the modification of the instruction in 0011:

A 0363	- Initial contents of Accumulator
+Z 0001	- Plus contents of Location 0013
A 0400	- Final contents of Accumulator

If the addition were done according to the rules of decimal arithmetic, the answer would be A 0364. However, there is no address 0364, and the computer gives A 0400 as the answer. This is due to the following rule: when the sector portion exceeds 63, as in 0364, subtract 64 from the sector and add 01 to the track to arrive at the "right" answer.

THE Y INSTRUCTION

The Y instruction stores the address portion of the contents of the Accumulator in location m, replacing the address portion of the word in location m. The remaining bit positions of location m are unchanged.

Instruction	Explanation
Y	Address portion of (Acc.)-address portion of (m); (Acc.) and all but the address portion of (m) remain unchanged.

This allows storage of the address portion of the word in the Accumulator in memory without changing the command portion of the word already there. The most common use of the Y instruction is in address modification. Consider the following problem: Add the contents of 0300 to the contents of 0400 and store the sum in 0500; add the contents of 0301 to the contents of 0401 and store the sum in 0501; and so forth, until all the values in Track 03 have been added to the values in the corresponding sectors in Track 04 and stored in the corresponding sectors in Track 05. The coding for this follows:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
		X					
		1100	B	0300			Add contents of corresponding sectors
		1101	A	0400			in Tracks 03 and 04 and hold sum in
		1102	H	0500			corresponding sector in Track 05.
		1103	B	1100	X		Bring instruction from 0000 into Acc.
		1104	A	1112			Add Z 0001 to the instruction in Acc.
		1105	Y	1100			Store modified address into instruction
							in 0000.
		1106	A	1113	X		Add Z 0100 to the instruction in Acc.
		1107	Y	1101			Store modified address into instruction in
							0001.
		1108	A	1113			Add Z 0100 to the instruction in Acc.
		1109	Y	1102	X		Store modified address into instruction in
							0002.
		1110	S	1114			Subtract B 0600 from instruction in Acc.
		1111	T	1100			Return to beginning of loop if (Acc.) negative.
		1112	Z	0001	X		HALT — also used as constant in address
							modification.
		1113	Z	0100			Constant used in address modification.
		1114	B	0600			Constant used to test for end of loop.
					X		

The instructions in 1100 through 1102 perform the addition described in the "Notes" column. The B1100 instruction in Location 1103 places the contents of Location 1100, B0300, in the Accumulator. Then, the A1112 instruction in Location 1104 adds the contents of Location 1112, 20001, to the contents of the Accumulator, B0300, resulting in B0301, as follows:

B 0300 - Initial contents of Accumulator
+Z 0001 - Plus contents of 1112

B 0301 - Final contents of Accumulator

The YllOO instruction in Location 1105 now replaces the address portion of the instruction word in Location 1100, B0300, by the address portion of the instruction word in the Accumulator, B0301. The result is to change the instruction in 1100 from B0300 to B0301. The YllOO instruction does not alter the word in the Accumulator. Therefore, B0301 remains in the Accumulator.

The A1113 instruction in Location 1106 adds the contents of 1113 to the contents of the Accumulator, as follows:

B 0301 - Initial contents of Accumulator
+Z 0100 - Plus contents of 1113

B 0401 - Final contents of Accumulator

The Y1101 instruction in Location 1107 now replaces the address portion of the word in 1101, A0400, by the address portion of the word in the Accumulator, B0401. The result of this is to change the instruction in 1101 from A0400 to A0401. Notice the command portion, A, of the word in Location 1101 did not change even though it is different from the command portion, B, of the word in the Accumulator.

The A1113 instruction in Location 1108 adds the contents of 1113 to the contents of the Accumulator, as follows:

B 0401 - Initial contents of Accumulator
+Z 0100 - Plus contents of 1113

B 0501 - Final contents of Accumulator

The Y1102 instruction in Location 1109 then changes the instruction in 1102 from HO500 to H0501.

The S1114 instruction in 1110 subtracts the contents of 1114 from the contents of the Accumulator. This results in a negative word, as shown below, since B0600 is mathematically larger than B0501.

B 0501 - Initial Contents of Accumulator
- B 0600 - Subtract the contents of 1114

Negative Word - Final contents of Accumulator

The T1100 instruction in Location 1111 will, therefore, transfer to the beginning of the loop to add the next pair of numbers from Tracks 03 and 04 and store the result in Track 05.

At the beginning of the final pass through the loop, the instructions in 1100 through 1102 read as follows:

Location	Instruction
1100	B 0363
1101	A 0463
1102	H 0563

The final sum, therefore, is stored in 0563. The instructions in 1103 through 1109 then modify the above instructions to read as follows:

Location	Instruction
1100	B 0400
1101	A 0500
1102	H 0600

The S1114 instruction in Location 1110, for the first time, results in a positive word (zero) as follows:

B 0363 - Initial contents of Acc. as a result of the B 1100 instruction in Location 1103.
+ z 0001 -
B 0400 - Contents of Acc. as a result of the A ill2 instruction in Location 1104.
+ z 0100
B 0500 - Contents of Acc. as a result of the A 1113 instruction in Location 1106.
+ z 0100
B 0600 - Contents of Acc. as a result of the A 1113 instruction in Location 1108.
− B 0600
ZERO - Contents of Acc. as a result of the S 1114 instruction in Location 1110.

The TlIOO instruction in Location 1111, therefore, rather than branching back to 1100 and through the loop again, allows the computer to continue to Location 1112, where it halts.

Notice the ZOO01 instruction in 1112 is used both as a halt, when the loop terminates, and as a constant by the A1112 instruction in Location 1104. This is convenient if the program must be in a limited memory area in the computer. Generally, however, this dual function is not used.

INITIALIZATION

Taking another simple program, compute the product of consecutive pairs of numbers on Track 03 and store these 32 products in Locations 0400 through 0431 as follows: (0300) x (0301) ⟶ (0400); (0302) x (0303) ⟶ (0401); etc. , through (0362) x (0363) ⟶ (0431).

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0000	B	[0300]	'		Compute the product of a pair of
		0001	M	[0301]	'		numbers from Track 03 and store
		0002	H	[0400]	'		on Track 04.
		0003	B	0000	'	☒	Bring instruction from 0000 into Acc.
		0004	A	0014	'		Add Z 0002
		0005	Y	0000	'		Store modified address into instr. in 0000
		0006	A	0015	'		Add Z 0001
		0007	Y	0001	'	☒	Store modified address into instr. in 0001
		0008	B	0002	'		Bring instruction from 0002 into Acc.
		0009	A	0015	'		Add Z 0001
		0010	Y	0002	'		Store modified address into instr. in 0002
		0011	S	0016	'	☒	Subtract H 0432
		0012	T	0000	'		Test for end of loop
		0013	U	1400	'		Transfer to output program
		0014	Z	0002	'		Constant used in address modification
		0015	Z	0001	'	☒	Constant used in address modification
		0016	H	0432	'		Constant used to test for end of loop.
					'		
					'		

Assume the program has been executed and the products in 0400 through 0431 have been printed out, or otherwise disposed of, so they are no longer needed. With the program still in memory, a new set of data could be stored in Track 03, and the program restarted at Location 0000. Would it make the same calculations on the data and store the answers in Locations 0400 through 0431? In other words, after replacing the old data with new, could the program be restarted and do exactly the same thing the second time? The answer is no, because the instructions in 0000 through 0002 were modified during execution of the program so that, when the program halts, these instructions read:

Location	Instruction
0000	B0400
0001	MO401
0002	HO432

The program is set to process data on Track 04, not Track 03, and to store the products starting at 0432 instead of 0400. These modified instructions must be reset or **"initialized"** before the program is executed a second time. One method for initializing these instructions is to enter a new program in memory with the same instructions as in the original program. The more efficient and preferred method is to write the program to be "self-initializing" as follows:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0000	B	[0300]	'		Compute the product of a pair of
		0001	M	[0301]	'		numbers from Track 03 and
		0002	H	[0400]	'		store on Track 04.
		0003	B	0000	'	☒	Bring instruction from 0000 into Acc.
		0004	A	0014	'		Add Z 0002.
		0005	Y	0000	'		Store modified address into instr. in 0000.
		0006	A	0015	'		Add Z 0001.
		0007	Y	0001	'	☒	Store modified address into instr. in 0001.
		0008	B	0002	'		Bring instr. from 0002 into Acc.
		0009	A	0015	'		Add Z 0001.
		0010	Y	0002	'		Store modified address into instr. in 0002.
		0011	S	0016	'	☒	Subtract H 0432.
		0012	T	0000	'		Test for end of loop.
		0013	U	1400	'		Transfer to output program.
		0014	Z	0002	'		Constant used in address modification.
		0015	Z	0001	'	☒	Constant used in address modification.
		0016	H	0432	'		Constant used to test for end of loop.
		0017	B	0024	'		Bring Z 0300 into Accumulator.
		0018	Y	0000	'		Initialize instr. in 0000 to read B 0300.
		0019	A	0015	'	☒	Add Z 0001 resulting in Z 0301.
		0020	Y	0001	'		Initialize instr. in 0001 to read M 0301
		0021	B	0025	'		Bring Z 0400 into Acc
		0022	Y	0002	'		Initialize instr. in 0002 to read H 0400
		0023	U	0000	'	☒	Branch to beginning of loop.
		0024	Z	0300	'		Constant used in initializing
		0025	Z	0400	'		Constant used in initializing

The instructions in Locations 0017 through 0025 are initializing instructions which make the program self-initializing when additional data are to be processed by it. After the program is in memory, it can be executed as many times as desired by starting execution at Location 0017, not 0000. All programs should be self-initializing. The execution of the program then starts at the beginning of the initializing instructions.

SUBROUTINE CONCEPT

The solution to a problem often requires that the same operation be performed more than once. This can be graphically shown in the form of a "Flow Chart" (Figure 3.7):

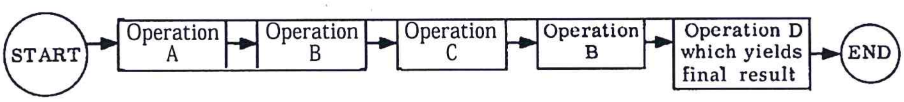


FIGURE 3.7 Typical Flow Chart

Note that the example above constitutes an extreme simplification of a programming flow-chart. In actuality, each operation would be plotted out in every detail, so that Operation A alone might represent a series of steps which could cover an entire page or more.

Assuming now that Operation B is long and involved and the program coded as flow-charted above, it would further be necessary to show the same long sequence of instructions for Operation B twice. Obviously, it would be preferable if the instructions for this operation could be written just once and used again wherever required in the program.

If this is done, the program could transfer (with a U instruction) at the end of Operation A or C to the beginning of the sequence of instructions which performs Operation B. The question now arises, how does one exit from (or branch out of) Operation B to the appropriate place in the program- the beginning of either Operation C or Operation D? The exit instruction from Operation B is a U instruction with a variable address portion and will be set prior to transfer to Operation B. At the end of Operation A and before the transfer to Operation B, the address portion of the U instruction must be set to exit from Operation B to Operation C; at the end of Operation C and before entering Operation B, the address portion of the U instruction must be set to exit from Operation B to Operation D.

This introduces the R instruction:

<u>Instruction</u>	<u>Explanation</u>
R	(Counter) + 1 -address portion of (m); that part of (m) other than the address portion is unchanged.

In other words, the contents of the Counter Register plus 1 replace the address portion of memory location m. At the time an instruction is executed, the Counter contains the location of the next instruction to be executed. Adding 1 to the contents of the Counter when the R instruction is executed gives the location of the R instruction plus 2. Therefore, the R instruction causes its own location plus 2 to replace the address portion of (m). The rest of the word in location m is unchanged.

The skeleton coding for the flow-charted problem could look like this:

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	,						
	,	☒					
		0000		ANY	,		
		:		:	,	Operation A	
		0010		ANY	,		
		0011		0040	,	☒ Set exit from Operation B to return to 0013.	
		0012	U	0032	,	Enter Operation B	
		0013		ANY	,		
		:		:	,	Operation C	
		0020		ANY	,	☒	
		0021	R	0040	,	Set exit from Operation B to return to 0023	
		0022	U	0032	,	Enter Operation B	
		0023		ANY	,		
		:		:	,	☒ Operation D	
		0030		ANY	,		
		0031	Z	0000	,	End – Halt	
		0032		ANY	,	Entrance Point	Operation B
		:		:	,	☒	
		0039		ANY	,		
		0040	U	[ ]	,	Exit Point	

Operation A extends from Locations 0000 through 0010. The R0040 instruction in 0011 sets the address portion of the U instruction in 0040 to 0013 (location of R0040 instruction plus 2). The UO032 instruction in 0012 branches to Operation B. A branch to Operation C will occur at the end of Operation B because the U instruction in 0040 now reads UO013.

Operation C extends from Locations 0013 through 0020. The R0040 instruction in 0021 sets the address portion of the U instruction in 0040 to 0023 (location of R0040 instruction plus 2). The UO032 instruction in 0022 transfers to Operation B again. This time, at the end of Operation B there will be a transfer to Operation D, because the U instruction in 0040 now reads UO023.

Operation D extends from Locations 0023 through 0030, and a Halt is at 0031.

Operation B, in Locations 0032 through 0040, is termed a "subroutine," and the instructions in Locations 0000 through 0031 constitute a "source program." The source program may "call" (use) the subroutine any number of times. In the example, the subroutine is only called twice. The entry point to the sample subroutine is Location 0032 and the exit point is 0040. Actually, the entry point does not have to be the first instruction in the written subroutine as in the example, nor does the exit point have to be the last instruction. Subroutines are programs which are used many times. Thus, like all programs, they may start and end anywhere in the written program.

The R-U sequence which is used to call the subroutine is termed a "calling sequence." In the example, the calling sequence consists merely of these two instructions. Some subroutines may require more elaborate calling sequences.

For example, some subroutines may require, before being entered, that certain information to placed in the Accumulator. Also, it is possible to "nest" subroutines to any desired depth; i.e., one subroutine could call another subroutine, which in turn could call still another, and so on. When standard subroutines from the Commercial Computer Division library are acquired, they are accompanied by a program description which details the function of the program, how to load it, what the exact calling sequence must be, and any other information necessary for its operation.

BINARY NUMBER SYSTEM 4

An understanding of the binary number system is necessary before proceeding with a further examination of LGP-21 programming concepts. Each digit of a decimal number has a multiplier associated with it. Take, for example, the number 237.

Multipliers:	etc. ← 1000	100	10	1
Digits:		2	3	7

Starting with the least significant digit (first digit to the left of the decimal point) the associated multiplier is 1 (or 10^0); moving one place to the left, the multiplier is 10 (or 10^1), then 100 (or 10^2), 1000 (or 10^3), etc. The multipliers, starting with the least significant digit and moving to the left, are consecutively higher powers of 10. The number 237, then means:

7 ones plus	7x 1 = 7
3 tens plus	3 x 10 = 30
2 one hundreds	2 x 100 = 200
Total	237

The binary number system is similar to the decimal system, with two important differences. First, the multipliers starting with the least significant digit and moving to the left are consecutively higher powers of 2: 1 (or 2^0), 2 (or 2^1), 4 (or 2^2), 8 (or 2^3), etc. The second difference is that any digit position may contain only a 0 or 1, whereas, in the decimal system, any digit position may contain 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. An example of a binary number, then, is

Multipliers:	etc. ←	128	64	32	16	8	4	2	1
Digits:		1	1	1	0	1	1	0	1

This binary number, 11101101, is constructed just like the decimal number 237, above.

By adding the respective multiplier values for each binary digit, starting with the least significant digit, we find that

1x1	=	1
0x2	=	0
1x4	=	4
1x8	=	8
0x16	=	0
1x 32	=	32
1x 64	=	64
1x 128	=	128
		237

Thus, the decimal number 237 is equivalent to 11101101 in binary. The decimal system is based on 10 digits, and the binary system on 2. The standard notation used to specify the base of a number is a subscript. Therefore, the equivalence could be written:

$$237_{10} = 11101101_2$$

To convert a binary number to its decimal equivalent, write the multipliersabove each of the binary digits, then total all the multipliers that have the digit "1" below them. For example, find the decimal equivalent of 11000011010_2:

1024 512 256 128 64 32 16 8 0 2 0

$$
\begin{array}{r}
1024 \\
512 \\
16 \\
8 \\
2
\end{array}
$$

1562_{10} = 110000110102

One way to convert a decimal integer (whole number) to its binary equivalent is to divide the number by 2. The remainder becomes the least significant binary digit; the quotient (e.g. , 237 ÷ 2 gives a quotient of 118 and a remainder of 1) is again divided by 2 and the remainder becomes the next binary digit. This process continues until the quotient is zero. The remainders become the binary number, where the first remainder is the least significant binary digit and the last remainder is the most significant (far left) binary digit.

Example: Convert 23710 to its binary equivalent.

Quotient	Remainder
2 | 237	
2 | 118	1 least significant
2 | 59	0
2 | 29	1
2 | 14	1
2 | 7	0
2 | 3	1
2 | 1	1
0	1 most significant

Therefore, 237_{10} ◦ 11101101_2.

In the decimal system the digits to the right of the decimal point (fractions)also have multipliers. Take, for example, the number .6875:

Multipliers:	1/10	1/100	1/1000	1/10,000 ——► etc.
Digits:	6	8	7	5

The most significant fractional digit (first digit to the right of the decimal point) has a multiplier of 1/10 (or 10^{-1}); moving one place to the right, the multiplier is 1/100 (or 10^{-2}), then 1/1000 (or 10^{-3}), 1/10,000 (or 10^{-4}), etc. The multipliers, starting with the most significant digit and moving to the right, are consecutively lower powers of 10. The number .6875, therefore, constitutes a series of additions, as follows:

$$
\begin{array}{rcl}
6 \times 1/10 & = & .6 \\
8 \times 1/100 & = & .08 \\
7 \times 1/1000 & = & .007 \\
5 \times 1/10000 & = & \underline{.0005} \\
 & & .6875
\end{array}
$$

Again, the binary system works similarly. The multipliers, starting with the most significant fractional digit and moving to the right, are consecutively lower powers of 2, namely 1/2 (or 2^{-1}), 1/4 (or 2^{-2}), 1/8 (or 2^{-3}), 1/16 (or 2^{-4}), etc. Again, a digit position can only contain a 0 or a 1. An example of a binary fraction is

Multipliers:	1/2	1/4	1/8	1/16 ——► etc.
Digits:	.1	0	1	1

To convert a binary fraction to its decimal equivalent, the multiplier values of the binary fraction are added again, just as in the decimal example:

```
1 x1/2  - .50
0 x 1/4 - . o o
1 x 1/8  = .125
1 x 1/16 - .0625
           .6875
```

Therefore, $.6875_{10} = .1011_2$

One way to convert a decimal fraction to its binary equivalent is to multiply successively by 2, ignoring any digit to the left of the decimal point in the multiplicand, when performing the successive multiplications.

Example: Convert $.6875_{10}$ to its binary equivalent.

```
 .6875
     2
 ------
1x37 50
 x    2   (ignoring the "1" to the left of the point in the
 ------    multiplicand)
0.7500
     2
 ------
1x5000
 x    2   (ignoring the "1" to the left of the point in the
 ------    multiplicand)
1.0000
```

Continue until there are all zeros to the right of the decimal point, as on the last multiplication above, or until the number of multiplicands equals the number of bits to the right of the binary point in the number. Going back to the first result, write down the digits to the left of the point in each product; place a point in front of these to get the binary equivalent of the decimal number.

Therefore, $.6875_{10} = {}_{\wedge}1011_2$

In the decimal system this is called a decimal point; in the binary system, a binary point. The binary point is usually represented as a caret (A). Also in binary terminology, the word "bit" is often used synonymously with "binary digit"-thus, "a 32 bit number" and "a 32 digit binary number" are the same thing.

ADDITION IN BINARY

Addition is the same as in the decimal system, except, 1 + 1 = 0 with a 1 carried.

Examples:

```
  1     10     11    111
+ 1    + 1    + 1   + 11
 ---
 10     11    100   1010
```

SUBTRACTION IN BINARY

Subtraction is also the same as in decimal, except, 0 -1 = 1 with a 1 borrowed: i.e. borrow 1 from the left and add 2 to the digit on the right, just as you would add 10 if working in decimal.

Examples

```
 -1     0    -10    1010    100
  0    -0      1   1.-111   -11
```

MULTIPLICATION AND DIVISION IN BINARY

. The rules are the same as in decimal.

Examples:

0 x 0 = 0	0 ÷ 0 = 0
1 x 0 = 0	0 ÷ 1 = 0
1 x 1 = 1	1 ÷ 1 = 1

Once the binary configuration of a number has been established, it is not difficult to imagine what it looks like in a memory location. For example,

$$.375_{10} = {}_\wedge 011_2$$

appears in the LGP-21 as

0 0110000000000000000000000000000

- position of binary point* t spacer is always 0
- sign bit, always 0 for positive numbers.

NEGATIVE NUMBERS

Bit position zero of a computer word will indicate whether the number is positive or negative. However, the sign of a positive number is not changed by simply inserting a 1 in position zero. Instead, negative numbers are held in the computer as a 2's complement.

Consider, for example, the number -. 375 which is represented in.binary as:

0	1	2	3	4	5	6	7	8	9		29	30	31	bit positions of computer word
1$_\wedge$	1	0	1	0	0	0	0	0	0	. . .	0	0	0	

The quickest way to see why this is the computer's way of representing $-.375_{10}$ is to add it as a binary number to the representation of $+.375_{10}$:

		0	1	2	3	4	5	6	7	8	9		29	30	31	bit positions of computer word
$+.375_{10}$ =		0$_\wedge$	0	1	1	0	0	0	0	0	0.	. .	0	0	0	
$-.375_{10}$ =		1$_\wedge$	1	0	1	0	0	0	0	0	0	. . .	0	0	0	
	1	0$_\wedge$	0	0	0	0	0	0	0	0	0	. . .	0	0	0	

If the 1 to the left of bit position zero is dropped, the result is 0, just as $.375_{10}+(-.375_{10})=0$.

One way to obtain the representation of a negative number is the following:

1. Change its sign to + and write its binary representation.
2. Starting at the left, change all the I's to O's and all the O's to I's, until the last 1 is reached. This 1 and all the following zeros remain unchanged.

The largest positive number the LGP-21 Accumulator can holds is

01111111111111111111111111111110

---

* The binary point for a number is never actually stored in memory. The location of the imaginary binary point inside the computer is between bit positions 0 and 1. However, for convenience in expressing integer values, the binary point is often assumed to be moved to other positions. This relative position is referred to as "q" and is discussed later in this chapter.

If the negative value of this number is used, it appears as

10000000000000000000000000000010

The number -1, which cannot be converted according to the above rule, appears as

10000000000000000000000000000000

Notice that the first bit position of all positive numbers contains a zero, and that of all negative numbers a 1. It is the sign bit of the Accumulator which is examined by the circuits associated with the Conditional Transfer instruction.

INSTRUCTION WORDS

The format of a word which is interpreted by the computer as an instruction is as follows (Figure 4.1):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
±												Command						Track Address						Sector Address							

└ Interpreted only in conjunction with T, I, P, and Z instructions.

FIGURE 4.1 Instruction Word Format

The only bits the computer considers when interpreting an instruction word are 0, 12 through 15, and 18 through 29. Other positions in the word do not affect the meaning of the instruction.

Each of the command symbols or letters has a 4-bit code. This code is held in positions 12 through 15 of the instruction word. The 4 positions 12 through 15 allow for 16 different 4-bit patterns. The 16 eommand symbols and their 4-bit codes are listed in Figure 4.2 (Note: some of these have not yet been discussed.

Symbol	Command	Code
Z	Halt; Sense and Transfer	0000
B	Bring	0001
Y	Store Address	0010
R	Set Return Address	0011
I	Input or Shift	0100
D	Divide	0101
N	N Multiply (save right)	0110
M	M Multiply (save left)	0111
P	Print or Punch	1000
E	Extract	1001
U	Unconditional Transfer	1010
T	Conditional Transfer	1011
H	Hold	1100
C	Clear	1101
A	Add	1110
S	Subtract	1111

FIGURE 4.2 List of LGP-21 Commands

Examples of some instruction words:

DECIMAL	BINARY												COMMAND						TRACK						SECTOR							
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
B 0 5 2 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	1	1	1	0	0
S 6317	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	0	1	0	0	0	1	0	0

In the discussion of the Y instruction, it was explained that this instruction causes the address portion of the contents of the Accumulator to replace the address portion of the contents of location m. This means that the contents of bit positions 18 through 29 of the Accumulator replaces the contents of bit positions 18 through 29 of memory location m.

Also discussed earlier was half of the rule for track-and-sector arithmetic when adding two instruction words. The rule was that, when the sector comes to 64 or more, subtract 64 from it and add 1 to the track. Now, consider track modification. When a track address exceeds 64, a 1 is carried into bit position 17 (one of the bits which are ignored in an instruction). This allows "end-around" programming; i. e. , one could consider the tracks as being in sequence, numbered 00, 01, 02,. . .60, 61, 62, 63, 00, 01, etc. For example, if the address 1500 were to be added to the address 5329, the resulting address would be 0429 and a 1 bit would be carried into bit position 1'7. This carry is important if an address is used to terminate a loop which results in an "end-around" operation.

It was also noted earlier that adding Z is the same as adding zero. These rules are based on binary arithmetic. Some examples of arithmetic operations using two instruction words follow:

DECIMAL	BINARY												COMMAND						TRACK						SECTOR							
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
B4218	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	3	0	1	0	0	1	0	0	0
+Z0056	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	1	1	0	0	0	0	0
B4310	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	1	0	0	1	0	1	0	0	0
H3638	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	1	0	3	1	0	0	1	1	0	0	0
+ 23300	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
H0538	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	1	0	1	1	0	0	1	1	0	0	0
S4215	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	0	1	0	1	3	0	0	1	1	1	1	0	0
+ 23551	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0	1	1	0	0
S1402	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0	1	1	1	3	0	0	0	0	1	0	0	0
H0301	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0
-H0500	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
-S6201	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	0	0

Notice that the bits in the Command portion of the instruction word can be manipulated, too. For example, if a Bring command is added to a Hold command, the result would be a Clear command.

DECIMAL	BINARY																															
													COMMAND						TRACK						SECTOR							
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	3	31
B1408	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0
+H1026	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	1	0	0	1	1	0	1	0	0	0
C2434	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0

Therefore, care must be exercised when adding or subtracting instructions (e.g., to test for the end of a loop) so that the desired result will be obtained.

DATA WORDS

The format of a word interpreted as data by the computer is shown in Figure 4.3. It consists of a sign (in bit position zero) and 30 bits of magnitude. The 31st, or spacer bit, is always zero in memory. A computer word can represent data in a number of different forms, including:

1. Binary
2. Binary-Coded Decimal (4-bit format)
3. Alphanumeric (6-bit format)

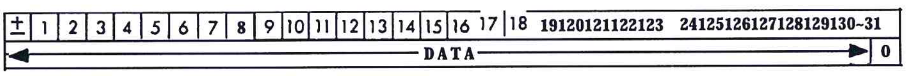


FIGURE 4.3 Data Word Format

Binary Data

When the number 125.25_{10} is handled in the computer as binary data, it appears in this form: $1111101_{\wedge}01_2$. Since there are 32 places in a computer word, the question arises: Where in the 32 places is the $1111101_{\wedge}01$ positioned. The answer is that it can be anywhere in the word. The convention for denoting the position of the number is to specify the value of q; q being the position of the least significant integer bit, and the caret symbol indicating the position of the binary point in the computer word. For example:

Decimal No.	Computer Word																															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
125.25 @ q = 12	0	0	0	0	0	0	1	1	1	1	1	0	1‸	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
125.25 @ q = 10	0	0	0	0	1	1	1	1	1	0	1‸	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The letter q is sometimes dropped and the decimal number written as 125.25 @ 12 or 125.25 @ 10. This convention also applies to instruction words. Therefore, for the example above, we could write that the command is @ 15, the track address @ 23, and the sector address @ 29.

Since the position of the binary point in a computer word is merely an assumption for the programmer's convenience, the computer does not know where it is, but assumes it to be between bits 0 and 1, or at a q of 0 for all numbers, including results of arithmetic operations. The programmer therefore considers a number (as interpreted by the computer) to be multiplied by 2^q, where q is the assumed binary point.

Example:

Computer Word	Number as Interpreted by Computer	Number as Interpreted by Programmer
0100 --------0	.5	$.5 \times 2^q$

If the programmer's q is 2, the number is $.5 \times 2^2$ or 2; if his q is 3, the number is $.5 \times 2^3$ or 4. This is analogous to multiplying by 10^x in the decimal system by moving the decimal point "x" places to the right.

When decimal data is to be entered into the computer, it can be read in and converted to binary by one of the data input subroutines available from General Precision. The q of the binarized data is specified by the programmer. Care must be taken to specify a q at which the data can actually be held. The q can be determined only when the largest value is known which the subroutine is being asked to read at a given time. This means that the programmer must specify a q at least large enough that the largest data value can be binarized to that q. Further, if the programmer wants to retain as much significance to the right of the binary point as possible, he should not make the q any larger than necessary.

By consulting the Powers of 2 Table, (Appendix C), it is easy to determine the largest number that can be held at any given q and the decimal places of accuracy possible to the right of the binary point. For example, $2^9 = 512$ means that at a q of 9, the computer can hold binary numbers ranging from -512 to almost $512, as shown below:

DECIMAL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-512 @ 9	1	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	0
511.9...9@9	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

(BINARY)

To determine the precision to the right of the binary point at a q of 9, one must consider how many binary places there are between positions 10 and 30 inclusive (position 31 is the spacer bit). This would allow 21 binary places. The Powers of 2 Table shows that $2^{-21} = .000000476...$ Therefore, the programmer can safely expect, at a q of 9, to hold in binary the equivalent of decimal numbers accurate to 6 decimal places to the right of the decimal point.

Binary-Coded Decimal Data

Each decimal digit has a 4-bit code as follows:

Decimal Digit	Code	Decimal Digit	Code
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

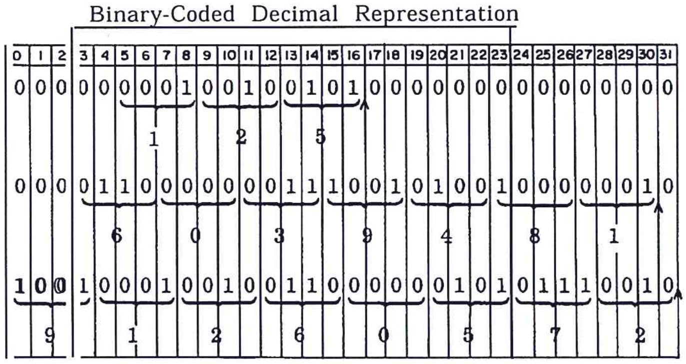
Binary-coded decimal data is held in groups of 4 bits, each group representing a decimal digit. Up to 8 such digits can be held in a 32-bit computer word. Examples:

Decimal No. — Binary-Coded Decimal Representation

125 @ 16

6039481 @ 30

91260572 @ 31



It should be observed that the same number in binary-coded decimal and in simple binary presents two entirely different bit patterns:

125 in BCD @ 30 o - - - - - 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 ~ 0
125 in Binary @ 30 o - - - - - 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 ~ 0

Decimal data enters the computer in binary-coded decimal; usually it is converted to binary and stored. However, there are some instances when this conversion is not necessary. If it is an identification number (such as a stock or employee number), has only numeric (no alphabetic) characters, and the problem requires merely that the program be able to determine its relationship to other identification numbers-equal, not equal, less than, or greater than-binarization may be unnecessary. For even though the computer performs pure binary, not binary-coded decimal arithmetic, it can subtract one binary-coded decimal number from another and use the sign of the difference to indicate relative magnitudes. This is possible because the two numbers, as interpreted by the computer, retain the same relative magnitudes as they have when they are interpreted by people as binary-coded decimal numbers. For example, assume X and Y are two binary-coded decimal numbers at the same q and that Y is greater than X. When they are interpreted by the computer as binary numbers at a q of 0, Y will still be larger than X. The result of subtracting Y from X will, of course, be meaningless except for the sign. Note however, that this type of arithmetic is not possible when either of the binary-coded decimal numbers has a binary 1 in bit position zero, as the computer would then consider it a negative number.

Data binarization is also unnecessary for a one digit number and for data which, after being entered, becomes part of the output but is used in no other way.

Alphanumeric Data

When data consists of a combination of alphabetic and numeric characters—such as names, identification numbers which also contain alphabetic characters, or typewriter control codes-it is called alphanumeric. This kind of data must be stored in 6-bit form. That is, 6 instead of 4 bits must be stored for each character, since four bits can only represent 16 different characters which is obviously insufficient for all the numeric and alphabetic characters available.

Appendix C contains a list of all available characters and their 6-bit codes. The first four bits are called the numeric bits and the last two, the zone bits. Notice that, in some cases, two characters have the same four numeric bits and can be distinguished only by their zone bits. The programmer must specify for every character which enters the computer whether he wants it recorded in memory in 4-bit or 6-bit mode. When entering strictly numeric data, the 4-bit mode should be used, as no two digits have the same numeric bits. However, for alphanumeric data input the 6-bit mode should be used, so that distinction between characters with identifical four numeric bits is possible; e.g., between F and U. A 32-bit word can hold five alphanumeric characters. Example: "LGP21" in 6-bit format at a q of 29 appears as follows:

00011010111010000100101000011000

L G P 2 1

There are other forms of internal data representation (such as floating-point), but their discussion is not necessary in this manual.

HEXADECIMAL NOTATION

Since it is awkward to write thirty-two O's and I's, a shorthand or hexadecimal notation for writing computer words has been devised. To find the hexadecimal representation of a computer word, divide its 32 bits into eight groups of four bits each. There are 16 possible combinations for any group of four bits. Therefore, each combination of four bits can be represented by one of a group of 16 characters, zero through W, used for this purpose, as well as the decimal equivalent of the 4-bit numbers. This is shown in Figure 4.4.

Binary	Hexadecimal	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	F	10
1011	G	11
1100	K	12
1101	Q	13
1110		14
1111	W	15

FIGURE 4.4 Hexadecimal Equivalences

Some examples follow:

Decimal Number: 23.75 @ 14

23 .75

	0 1 2 3	4 5 6 7	6 9 10 11	12 13 14 15	16 17 18 19	20 21 22 21	24 25 26 27	26 29 30 31
Computer Word	0 0 0 0	0 0 0 0	0 0 1 0	1 1 1 ~	1 1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
Hexadecimal Word	0	0	2	W	8	0	0	0

Decimal Instruction: B 2917

				COMMAND= 1		TRACK=29		SECTOR = 17
	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31
Computer Word:	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1 1	1 0 1 0	1 0 0 0	1 ~ 0 0
Hexadecimal Word	0	0	0	1	1	K	4	4

Alphanumeric Data: LGP21 @ 29

	L	G		P		2		1
Computer Word:	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31
	0 0 0 1	1 0 1 0	1 1 1 0	1 0 0 0 0	1 0 0 1	0 1 0 0	0 0 1 1	0 ~ 0 0
Hexadecimal Word	1	F	Q	B	4	F	1	8

Since the last character involves the spacer bit, it will normally be one of the eight even characters, which have zero as their fourth bit: 0, 2, 4, 6, 8, F, J, Q.

Decimal to Hexadecimal Conversion

A method for determining the appearance of numbers in the LGP-21 as sequences of thirty-two O's and l's was given earlier in this manual. Now a simpler method shall be explained which provides the eight hexadecimal characters which can be used to represent a given number at a given q.

Suppose 94.87654, at a q of 7, is to be expressed in hexadecimal. Two steps are required to find the first character:

1. Subtract the q of the given number from 3

   $3 - q = x$ therefore $3 - 7 = 4$

2. Evaluate 2^x and multiply this value by the given number:

   2^x(number) therefore $2^{-4}(94.87654) =$
   $(.0625)(94.87654) =$
   2.92978375

The first hexadecimal character is 5

Each of the remaining characters requires a single process:

3. Multiply the fractional part of the previous product by 16 (always 16, regardless of q). The integer part of the new product is the next hexadecimal character.

Thus, in the example given:

.92978375
x 16
14.87654000

.87654
x 16
14.02464

.02464
x 16
0.39424

.39424
x 16
6.30784

.30784
x 16
4.92544

.92544
x 16
14.80704

.80704
x 16
12.91264

Since the hexadecimal characters equivalent to 14 and 12 are Q and J, respectively, the hexadecimal representation is

$$94.87654_{10} \text{ @ } 7 = 5QQ064QJ$$

The fractional portion after the last multiplication, .91264, is greater than .5, so it may appear that the correct hexadecimal representation for 94.87654 at a q of 7 is closer to 5QQ064QK than to 5QQ064QJ. However, it may be recalled that the last character involves the spacer bit, and so must be even: 0, 2, 4, 6, 8, F, J, or Q. Therefore, 5QQ064QJ is the best possible approximation in the LGP-21.

This example illustrated a positive number. For negative numbers, one preliminary step is needed: subtract the negative number from the power of 2 which is 1 greater than the given q. For example, suppose the first two hexadecimal characters are to be found for the number -3.1415927 at a q of 3. First, the number must be subtracted from the power of 2 which is 1 greater than 3 (i.e., 2^4):

$$\begin{aligned} 2^4 &= 16.0000000 \\ \text{number} &= \underline{-3.1415927} \\ & 12.8584073 \end{aligned}$$

Then, proceed as with positive numbers: multiply by $2^{3-3} = 2^0 = 1$. No multiplication is necessary for this step.

$$\underline{12}.8584073$$

Thus, the first character is J (decimal value 12).

$$\begin{array}{r} .8584073 \\ \underline{\text{x} \quad\quad 16} \\ \underline{13}.7345168 \end{array}$$

The next character is K (decimal value 13), and so on.

Hexadecimal Instruction Words

Instruction words as well as data words may be represented in hexadecimal. The Command portion of an instruction occupies bits 12 through 15 and can be represented by a hexadecimal character. A complete list of the LGP-21 commands and their hexadecimal designations is given in Figure 4.5.

COMMAND	BINARY	HEXADECIMAL	DECIMAL
Z	0000	0	0
B	0001	1	1
Y	0010	2	2
R	0011	3	3
I	0100	4	4
D	0101	5	5
N	0110	6	6
M	0111	7	7
P	1000	8	8
E	1001	9	9
U	1010	F	10
T	1011	G	11
H	1100	J	12
C	1101	K	13
A	1110	Q	14
S	1111	W	15

FIGURE 4.5 Hexadecimal Designation of Commands

Thus, the hexadecimal word 8W517F36 would appear in memory as

±	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	0	0	0	1	1	1	1	0	1	0	1	0	0	0	1	0	1	1	1	1	0	1	0	0	0	1	1	0	1	1	0
8				W				5				1				7				F				3				6			

Since bits 12 through 15 are 0001 ($0001_2 = 1_{10}$), which is the binary equivalent of the Bring command, this word would be interpreted as a B instruction if it were to reach the Instruction Register.

The six bits, 18 through 23, contain the track portion of the operand address. In the above example, the bits are 111010. Their decimal equivalent is

etc. ← 1^{32} 1^{16} 1^{8} 0^{4} 1^{2} 0^{1} =

1 x 32 = 32
1 x 16 = 16
1 x 8 = 8
0 x 4 = 0
1 x 2 = 2
0 x 1 = 0
58

Therefore the track number is 58.

The next six bits, 24 through 29, contain the sector number. These bits are 001101, so the sector number is 13, according to the same conversion process:

etc. ← 0^{32} 0^{16} 1^{8} i 0^{2} i =

0 x 32 = 0
0 x 16 = 0
1 x 8 =8
1 x 4 =4
0 x 2 =0
1 x 1 =1
13

In summary, the hexadecimal word **8W517F36** is treated as a B5813 instruction, if it is in the instruction register.

```
                         B       track 58     sector 13
1000 1111 0101 0001 01 111010 001101 10
  8    W    5    1    7    F    3    6
```

NUMBER SCALING FOR THE LGP-27 5

SCALING

The LGP-21 considers all numbers to be within the range $-1 \leq n < +1$. How then, can one use a number like 50.5625_{10}?

A natural approach would be to move the decimal point two places left, until a number is obtained which can be handled by the computer; namely $.505625_{10}$.

However, this process, known as decimal scaling, is not accurate enough for the LGP-21, and since the computer performs all internal computations in binary form, a preferable approach is to use binary scaling. This means moving the binary instead of the decimal point.

For example, the binary equivalent of 50.5625_{10} is

$$110010_{\wedge}1001_2$$

This configuration suggests moving the binary point 6 places to the left to obtain a satisfactory fraction, namely $_{\wedge}1100101001_2$. This process is called q-scaling; in this case, scaling the number at a q of 6 (arithmetically: multiplying by 2^{-6}).

It would be rather tedious, however, if the programmer had to convert his data — which is normally stated in decimal form — to its binary equivalent. Fortunately, this is not necessary. Appendix C contains a Powers of 2 Table which will tell at a glance that 50.5625 would yield a fraction after a 6-place shift of its binary point.

The table is used in this manner: the left column, labelled "2^N", shows that the first power of 2 greater than 50.5625 is 64. Next to the 64, in the center column, is the number 6. This means that 50.5625_{10} becomes a fraction if its binary point is shifted left 6 places (or more). Thus, when the number is scaled at a q of 6, it will be within the range of the LGP-21.

It should be emphasized that the appropriate scaling value in the Powers of 2 Table must always be chosen as the next number greater than the one to be converted. For example, if the number above had been 64.000 instead of 50.5625, it could not have been held at a q of 6. The largest number for which this scaling value is valid is 63.999. The rule is not as strict for negative numbers. Both -63.999 and -64 can be held at a q of 6; but -64.001 can not. (It may be recalled that -1 can appear in the LGP-21 unscaled, while +1 can not.)

Sometimes it is desirable to scale numbers which are already fractions, in order to obtain greater arithmetic precision. To do this, a negative scale factor (-q) is specified. For example, the scale factor for the number $.01234_{10}$ is determined by finding the next number larger than .01234 which appears in the right-hand column (labeled "2^{-N}") of the Powers of 2 Table-namely .015625. Next to it, in the center column, is a 6. This means that .01234 can be stored at a q of -6 (or any larger q: -5, -4,. . . 0, 1, 2, etc.).

In summary, q-scaling operates as follows:

1. If a number can be expressed exactly in no more than 30 bits and is q-scaled for the LGP-21, it can be stored as an exact number.

2. If a number has to be divided by 2 (that is, multiplied by 1 at a q of 1) to align binary points or avoid overflow, only one of the 30 bits of magnitude of the number is lost.

3. If the binary representation of a number is known, its appearance at any q in a memory location or in the oscilloscope display can be written down. For example, the decimal number 19 has the binary configuration $10011_{\wedge}$, since 19 = 16+2+1. In the LGP-21, 19 at a q of 5 would appear as

```
  19
0 1 0 0 1 1^0 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
```

or at a q of 21 as:

```
                                  19
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1^0 0 0 0 0 0 0 0 0 0
```

When a number is shifted to the right, the vacated positions on the left side are filled with the original contents of bit position zero, because this is an arithmetic multiply. An example of a positive number shifted right is shown above; for a negative number:

```
             +1.25
+ 1.25 @ 1   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
             -1.25
- 1.25 @ 1   1 0^1 1 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
                              -1.25
-1.25 @ 9    1 1 1 1 1 1 1 1 1 0^1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

The programmer must know where the binary point is in the result of each arithmetic operation. The rules involved are simple:

1. Addition and Subtraction

   The q's of the operands must be the same, and the q of the result is the same as that of the operands.

   Examples:

```
   6 @ q = 9          15 @ q = 20
  +4 @ q = 9          -9 @ q = 20
  ----------          -----------
  10 @ q = 9           6 @ q = 20
```

   Overflow may occur in which case the computer continues after setting an overflow flag which may be tested by the -Z command.

2. Multiplication

   The q of the product equals the sum of the q's of the two operands.

   Examples:

```
   5 @ q = 5          22 @ q = 26
  x3 @ q = 2          x1 @ q =  3
  ----------          -----------
  15 @ q = 7          22 @ q = 29
```

   The M instruction can be used to shift right. To accomplish this, multiply the number to be shifted by 1 at a q = S, where S is the number of places to shift. In the second example above, 22 is shifted right 3 places because it is multiplied by 1 at a q = 3. The product extends to bit position 30 and is not rounded.

   In M multiplication, no overflow can occur. Bits shifted out of the Accumulator are lost.

   The N instruction can be used to shift left. To N-multiply by 1 @ q = n shifts the number in the Accumulator left 31 - n places without the possibility of overflow, although bits may be shifted out of the Accumulator on the left.

3. Division

The q of the quotient is equal to the q of the dividend minus the q of the divisor. In division it is necessary to determine what q is required for the dividend to insure that the developed q of the quotient will be sufficient to hold the largest expected result. Overflow will occur if the scale of the quotient is not sufficient to cover the answer that is developed.

Examples:

(24 @ 10) ÷ (2 @ 4) = 12 @ 6
(19 @ 17) ÷ (1 @ 2) = 19 @ 15

The D instruction can be used to shift left. To accomplish this, divide the number to be shifted left by 1 at a q = S, where S is the number of binary places to shift. In the second example above, 19 is shifted left 2 places because it is divided by 1 at a q of 2. The quotient is rounded at bit position 30. It is possible to cause overflow when shifting by means of a D instruction.

DECIMAL CONSTANTS IN A PROGRAM

Since the LGP-21 handles all data and internal computations in binary form, it is desirable to have a program which will read decimally-coded programs, convert them to binary form, and store them in designated locations. Such a program is called a program input routine and is provided to all LGP-21 users.

An LGP-21 program input routine does not accept constants entered in decimal format. They must be entered as instructions or hexadecimal words. For example, 1 at a q of 29 can be conveniently written as the instruction 20001, and 18 at a q of 23 as 21800. Constants which cannot be represented in this way must be written as hexadecimal words (leading zeros may be omitted). For example, 8.75 at a q of 4 must be written as 46000000 on the coding sheet.

Example problem: Calculate $5x^2 + 3x - 7.75 = y$ and store y in 6300 at a q of 10. The value x is less than 10 and is stored in 6301 at a q of 4. The constants 5 @ 3, 1 @ 1, 1 @ 4, 7.75 @ 10, and 3 @ 2 will be in the program.

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	X					
		0000	B	6301	'	Bring x@4	
		0001	M	6301	'	Multiply by x@4; x^2 @8	
		0002	M	0012	'	Multiply by 5@3; $5x^2$@11	
		0003	D	0013	' X	Shift left 1; $5x^2$@10	
		0004	S	0014	'	Subtract 7.75@10; $5x^2$ -7.75 @10	
		0005	H	6300	'	Hold $5x^2$ - 7.75@10	
		0006	B	6301	'	Bring x@4	
		0007	M	0015	' X	Multiply by 3@2; 3x@6	
		0008	M	0016	'	Shift Right 4; 3x@10	
		0009	A	6300	'	Add $5x^2$ -7.75@10; $5x^2$ -7.75 + 3x@10	
		0010	H	6300	'	Store result	
		0011	Z	0000	' X	Halt	
		0012	5000	0000*	'	5@3 in hexadecimal	
		0013	4000	0000	'	1@1 in hexadecimal	
		0014	00W8	0000	'	7.75 @ 10 in hexadecimal	
		0015	6000	0000	' X	3 @ 2 in hexadecimal	
		0016	800	0000	'	1@4 in hexadecimal	

* On the coding sheet, hexadecimal words must be preceded by a Program Input Code (see Figure 3.1) as required by whatever program input routine is being used. Since specific programs are not discussed in this manual, no input codes are shown in the above coding example.

THE M AND N INSTRUCTION

The N instruction multiplies the contents of the Accumulator by the contents of location m and leaves the least significant half of the result in the Accumulator. The M instruction also causes a multiply operation, but in this case the most significant half of the product is retained.

Multiplying two 30 bit numbers (plus sign) results in a 60 bit product (plus sign), which is in the Extended Accumulator (Figure 5.1). After an M instruction the Accumulator retains the sign and the 30 most significant bits of this product. The remaining bit in the far right of the Accumulator is the spacer bit which is always zero. After using the N instruction for multiplication, the 31st bit of the 60 bit product is in bit position zero of the Accumulator. Bit positions 30 and 31 of the Accumulator are always zero after an N multiply.

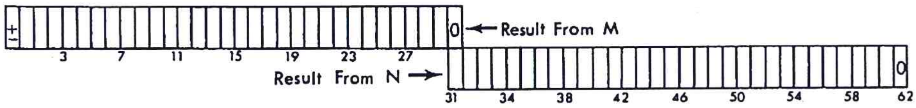


FIGURE 5.1 Extended Accumulator

Bit position zero of the Accumulator, after N, has no significance as a sign bit, unless the full 62 bit (plus sign) product contains all O's or all l's up to and including position zero of the result left in the Accumulator.

When an N-multiply instruction is used, the q of the result equals the q of the multiplicand plus the q of the multiplier, minus 31: (x @ 22) x (y @ 14) - 31 = xy @ 5. Examples of multiplication with the N instruction:

(19 @ 20) x (1 @ 28) = 19 @ 17

(120 @ 30) x (10 @ 31) = 1200 @ 30

The first example above demonstrates that N can be used for a left shift of S places by N-multiplying the number to be shifted by 1 at a q of 31 - S. To determine whether to shift left by N-multiplying or by dividing, the following general rule should be used:

> If the word to be shifted represents a binarized number and overflow is possible, divide. This allows for overflow detection, with the -Z instruction, which will be explained later.
>
> If the word has a logical meaning and overflow is possible, and if, in effect, a logical shift rather than an arithmetic division is desired, use N.
>
> If no overflow is likely to occur, either D or N can be used to shift left.

The second example above demonstrates that the N instruction can be used to obtain the product at the same q as the multiplicand in the Accumulator, if the multiplier is at a q of 31. However, only even numbers can be held at a q of 31 because position 31-the spacer bit-will be 0.

VARIATIONS OF THE Z INSTRUCTION

The Z instruction acts as a Halt, Sense Overflow, Sense Branch Switch, or No-operation.

When the address portion of a positive Z instruction is 0000 or 0100, the instruction is interpreted as a Halt. The computer stops in Phase 1 of the instruction following the 20000. The I register will contain the 20000, and the Counter will contain the address of the next instruction. If the track portion of the address is 02 or 03, the Z instruction is treated as a No-operation. That is, the instruction is brought into the I register but is not executed. The Counter is incremented, as usual, and the instruction following the Z is executed normally.

The negative Z instruction is interpreted as a test for overflow. The overflow indicator bit is recorded in the sign position (bit position zero) of the Counter Register. A "1" bit is recorded (i. e. , overflow indicator ON) when overflow occurs. A "0" bit (indicator OFF) signifies that no overflow has occurred. If the overflow bit (not to be confused with position zero of the Z instruction) is ON, the computer turns it OFF and executes the instruction immediately following the -Z; if the overflow bit is OFF, the instruction following the -Z is treated as a No-operation, after which the second instruction following the -Z is executed in the normal manner.

A negative Z instruction is programmed as 800Zn, which results in a 1 bit being placed in position zero of the binary instruction word in memory. This instruction is called the "eight hundred Z" or "minus Z" instruction.

Example of testing for overflow: If the contents of 6308 plus the contents of 6311 results in overflow, go to 0325 for the next instruction. Otherwise, go to 0236.

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		1520	B	6308	'		Bring (6308)
		1521	A	6311	'		Add (6311)
		1522	800Z	0200	'		Test for Overflow
		1523	U	0325	' ☒		Go to 0325 if Overflow ON.
		1524	U	0236	'		Go to 0236 if Overflow OFF.

In the example above, the track address 02 or 03 should be used in the -Z instruction because:

1. If 00 or 01 is used, the computer will halt after turning off overflow. A manual depression of the START switch would then be required to send the computer to Location 1523 for the next instruction. At this time the U0325 would be treated as a No-operation or executed as a branch instruction, depending upon the result of the test.

2. If a track address of 04 or greater is used, the instruction becomes a Sense Overflow and Branch Switch instruction, as explained below.

There are four Branch Switches on the computer console, labeled BS-4, BS-8, BS-16, and BS-32. These switches, which can be manually positioned ON or OFF, operate in conjunction with the track portion of the Z instruction. The computer will test the setting of the Branch Switch indicated by the track address of the Z instruction.

If the Branch Switch is ON, the computer will execute the next instruction in sequence; if the Branch Switch is OFF, the next instruction is treated as a No-operation, and the one following it is executed. For example, if the instruction is 20800 and BS-8 is ON, the instruction following the Z is executed. If BS-8 is OFF, the instruction following the Z is treated as a No-operation, and the one following that is executed normally. Several Branch Switches may be tested with one Z instruction by making the track address equal to the sum of the numbers of the switches. For example, 22800 will cause the computer to test BS-16, BS-8, and BS-4. If all the switches are ON, the next instruction will be executed; if any switch is OFF, the next instruction will be treated as a No-operation.

The Sense Overflow and Sense Branch Switch can be combined (-Z, plus track portion which specifies a Branch Switch). In this case, the next instruction will be executed if the overflow bit and all referenced Branch Switches are ON. If any one of these is OFF, the next instruction will be treated as a No-operation.

This discussion can be summarized as follows:

Instruction	Interpretation
z0000, Z0100	Halt
20200, 20300	No-operation
20400 through 26000	Sense Branch Switches; skip on no match.
-20000, -Z0100	Sense Overflow and halt.
- 20200, -20300	Sense Overflow only; skip on no overflow.
- 20400 through - 26000	Sense Overflow and Branch Switches

TRANSFER CONTROL SWITCH AND THE T INSTRUCTION

On the computer console is a switch labeled "TC" (Transfer Control), which can be manually positioned to ON or OFF. This switch operates in conjunction with the negative T instruction, programmed 800T, which results in a "1" bit in position zero of the T instruction. This instruction is referred to as the "eight hundred T" or "minus T" instruction. When a -T instruction is executed, a transfer to m will occur to obtain the next instruction if the Transfer Control switch is ON or the Accumulator contains a negative word. If the Transfer Control switch is OFF, the -T instruction will operate normally; that is, a transfer to m will occur if the Accumulator contains a negative word. Therefore, if this instruction is to be used for testing the position of the Transfer Control switch only, the Accumulator *must* contain a positive word at the time the instruction is executed.

THE E INSTRUCTION

The Extract instruction allows "masking" part of the word in the Accumulator, so that only the desired portion is retained. The masked out portion of the word is set to binary zeros; the word in location m is called the mask. Where the mask contains O's, the corresponding Accumulator bits are set to O's; where the mask contains l's, the corresponding Accumulator bits are left unchanged.

Examples: Assuming that the initial contents of the Accumulator and the contents of 2315 are as shown and that the computer executes an E2315, these would be the results:

First Example

Contents of 2315 (the Mask)	00000000000000001111111111111110
Initial contents of Acc.	00110110101011000101000011101100
Final contents of Acc.	00000000000000000101000011101100

Second Example

Contents of 2315 (the Mask)	11110000111100001111000011110000
Initial contents of Acc.	01101001000100000000111111110100
Final contents of Acc.	01100000000100000000000011110000

This instruction enables the programmer to "pack" and "unpack" computer words which contain more than one field of data.

Coding example: Location 0523 contains rate of pay in pennies at a q of 15 and hours worked at a q of 30. Compute gross pay (rate x hours) in pennies and store the result in 0563 at a q of 15.

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0000		B0523	'		Bring word from 0523
		0001		E0009	'		Keep hours worked
		0002		H6300	'		Save hours worked in temporary
					'	☒	storage
		0003		B0523	'		Bring word from 0523
		0004		E0010	'		Keep rate of pay @ 15
		0005		M0011	'		Shift right 1 to q = 16
		0006		N6300	'	☒	(Rate of pay @ 16) x (hours @ 30) =
					'		pay @ 46 - 31 = 15
		0007		H0563	'		Store gross pay in 0563 @ q = 15
		0008		Z0000	'		Halt
		0009		WWWQ	'	☒	Hexadecimal representation of mask
		0010	WWWW	0000	'		Hexadecimal representation of mask
		0011	4000	0000	'		1 @ q = 1.
					'		
						☒	

Notice that the instruction sequence B0523, E0009 yields exactly the same result as B0009, E0523. The mask can be in the Accumulator, and the word which is to be edited can be "extracted" from it, if this is more convenient.

INPUT/OUTPUT 6

The standard input/output device for the LGP-21 is the Model 121 Tape Typewriter unit. It consists of an electric typewriter, a paper-tape reader, and a paper-tape punch. The reader and punch cannot be used separately from the typewriter, but the typewriter may be used alone. That is, the typewriter is normally dependent upon the computer for electrical power, and therefore can be used only when the computer is ON. However, an extra cable is provided for connecting the typewriter to a standard outlet instead of to the computer. While this connection is used, the typewriter functions as an off-line device.

The typewriter has a standard keyboard which has been modified so that it can use the LGP-21 codes shown in Appendix C. Keys which represent commands are of a different color than the others. There is one additional key: the CONDITIONAL STOP CODE ('). It produces a code on tape which has two functions: to stop the paper-tape reader, and to send the "start" signal to the computer. In addition to the keys, a number of levers are part of the tape-typewriter unit. Their functions are described in Appendix B.

While optional input/output equipment is also available which provides higher operating speeds, if desired, the following discussion will be restricted to the standard unit entirely.

Information may be input to the computer from the typewriter keyboard (the manual mode of entering information), or it may be read from tape. In either case, a typed or "hard" copy of the information is produced. Similarly, information may be output from the computer to the typewriter, which will produce a hard copy and, if desired, a punched tape.

INPUT INFORMATION

When information is input to the computer, it enters the low end (i. e. , bit position 31) of the Accumulator in binary-coded decimal format. Each new character moves the preceding character to the left until the Accumulator is filled. If too many characters are entered, the left-most characters in the Accumulator will be lost. This, however, does not cause overflow. Before studying how the characters enter the Accumulator, their representation on punched tape shall be discussed.

CHARACTER REPRESENTATION ON TAPE

When a typewriter character is typed while the PUNCH ON lever on the typewriter is depressed, a pattern of holes-unique for each character-is punched across the six positions or channels of the tape.

For instance, if the characters 7 and M are punched consecutively, the pattern of holes on the tape would appear as shown in Figure 6.1.

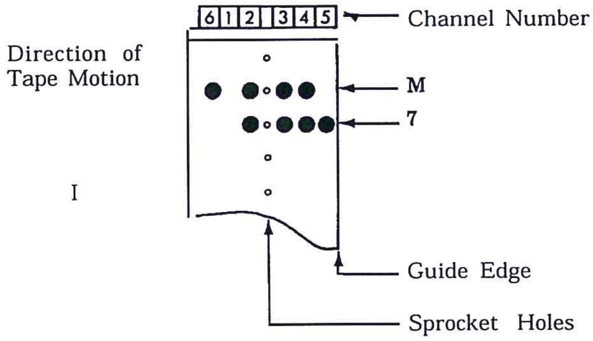


FIGURE 6.1 Character Representation on Tape

All the holes for a given character are punched simultaneously. Note that channel 6 is located next to channel 1.

Channels 5 and 6, on opposite edges of the tape, are called the zone channels; channels 1 through 4, the numeric channels. Thus the example in Figure 6.1 shows that the tape codes for 7 and M differ only in their zoning. The numeral 7 is one of the 16 hexadecimal characters (0 through 9, F, G, J, K, Q, W); M is one of the 16 letters which denote commands. The tape codes for all hexadecimal characters and all commands are given in the following table, Figure 6.2. Holes are presented by 1's; unpunched channels by 0's.

Character	Tape Code. 6 1 2 3 4 5	Decimal Value	Tape Code 6 1 2 3 4 5	Character
0	0 0 0 0 0 1	0	1 0 0 0 0 0	Z
1	0 0 0 0 1 1	1	1 0 0 0 1 0	B
2	0 0 0 1 0 1	2	1 0 0 1 0 0	Y
3	0 0 0 1 1 1	3	1 0 0 1 1 0	R
4	0 0 1 0 0 1	4	1 0 1 0 0 0	I
5	0 0 1 0 1 1	5	1 0 1 0 1 0	D
6	0 0 1 1 0 1	6	1 0 1 1 0 0	N
7	0 0 1 1 1 1	7	1 0 1 1 1 0	M
8	0 1 0 0 0 1	8	1 1 0 0 0 0	P
9	0 1 0 0 1 1	9	1 1 0 0 1 0	E
F	0 1 0 1 0 1	'10	1 1 0 1 0 0	U
G	0 1 0 1 1 1	11	1 1 0 1 1 0	T
J	0 1 1 0 0 1	12	1 1 1 0 0 0	H
K	0 1 1 0 1 1	13	1 1 1 0 1 0	C
Q	0 1 1 1 0 1	14	1 1 1 1 0 0	A
W	0 1 1 1 1 1	15	1 1 1 1 1 0	S

FIGURE 6.2 List of Tape Codes

The second column of this table shows that the hexadecimal characters have zones 0----1. Column 4 shows that the letters used for commands have zones 1----0. Column 3 gives the decimal value of the binary number formed by the holes in the numeric channels for both types of characters. A look at K and C in this table shows that both have numeric punches 1101 (8 + 4 + 1 = 7); they can be distinguished only by their zones.

The remaining characters have either zone 0----0 or 1----1. While sixteen codes are possible with each zone, only those actually used with the LGP-21 typewriter are listed in Figure 6.3.

Character	Tape Code 6 1 2 3 4 5	Decimal Value	Tape Code 6 1 2 3 4 5	Character
(Blank Tape)	0 0 0 0 0 0	0	1 0 0 0 0 1	Space
Lower Case	0 0 0 0 1 0		1 0 0 0 1 1	_
Upper Case	0 0 0 1 0 0	2	1 0 0 1 0 1	+ =
Color Shift	0 0 0 1 1 0	3	1 0 0 1 1 1	; :
Carriage Return	0 0 1 0 0 0	4	1 0 1 0 0 1	/ ?
Back Space	0 0 1 0 1 0	5	1 0 1 0 1 1	. 1
Tabulate	0 0 1 1 0 0	6	1 0 1 1 0 1	, [
		7	1 0 1 1 1 1	V
Cond. Stop	0 1 0 0 0 0	8	1 1 0 0 0 1	0
		9	1 1 0 0 1 1	X
		15	1 1 1 1 1 1	Delete

FIGURE 6.3 LGP-21 Special Character Codes

When information is entered into the Accumulator, the 4 numeric punch characters always enter, but the zone punches enter only if the programmer specifies the 6-bit mode of input. The bits corresponding to the six channels enter in 1-2-3-4-5-6 order, not 6-1-2-3-4-5 as they appear on tape. The 4- or B-bit mode is optional for every character, but the convention for the LGP-21 is to enter decimal and hexadecimal data in 4-bit mode, and alphanumeric data in B-bit mode.

THE INPUT INSTRUCTION

The I instruction determines the mode of input as follows: a negative Input instruction (8001) selects 4-bit mode; a positive Input instruction (I) selects 6-bit mode. The track address of the Input instruction determines what input device will be used: track address 00 selects the Model 141 Tape Reader; track address 02, the Model 121 Typewriter. The sector portion of the address has no effect on the Input instruction. When the typewriter is selected for input, information may be typed from the keyboard or read from tape, depending on how the MANUAL INPUT lever is positioned (see Appendix B).

Up to 31 input devices may be connected to a single LGP-21 system. Each device has an individual track address which is assigned at the time of its installation.

Examples of Input Instructions:

Instruction	Explanation
10200	B-bit input from Typewriter
10000	6-bit input from 141 Reader
80010200	4-bit input from Typewriter
80010000	4-bit input from 141 Reader

Examples:

1. Illustrated below is the contents of the Accumulator before and after reading the decimal digits 125' in 4-bit mode. (Note: The apostrophe represents the stop code which must be present to inform the reader when to sto reading, but it does not enter the Accumulator.) The X's represent the original bits in the accumulator before input (normally, all zeros).

```
before input  |X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|

after input   |X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|X|0|0|0|0|0|0|0|1|0|0|1|0|0|1|0|1|
                                                           1       2       5
```

   The 125 enters in binary-coded decimal in the twelve right-most positions of the Accumulator after causing its original contents to shift left in 4-bit increments to accomodate each of the incoming characters. In addition, four binary zeros have been inserted between the first character read and the pre-input contents of the Accumulator.

2. Illustrated below is the contents of the Accumulator after reading the alpha numeric characters LGP' in 6-bit mode.

```
XXXXXXXX000000000110101110100001
                  L     G     P
```

   The original contents of the Accumulator was shifted left, and the 6-bit code for each of the incoming characters was inserted in the far right positions. Also, six binary zeros have been inserted between the first character read and the pre-input contents of the Accumulator.

In both examples a "1" is in bit position 31. This can occur only immediately following input. If the word is stored in memory, the 1 bit in position 31 in memory will be lost (that is, set to 0). Significance in that bit position can only be saved by shifting the contents of the Accumulator left at least 1 place (using either the N, D, or the shift instruction explained in the next chapter) before the value is stored in memory.

Describing the input instruction in more general terms, the computer performs the following operations for each input instruction:

1. It shifts the contents of the Accumulator left 4 or 6 places, depending on the specified mode of input, inserting zeros in the vacated positions.

2. It reads a character from the selected device and holds the B-bit code for this character in a 6-bit register.

3. If the character in the 6-bit register is a stop code (binary code 100000), reading terminates, and the computer proceeds to the instruction following the input instruction. If a non-entering character other than the stop code is in the register, the computer returns to step (2); otherwise it goes to step (4).

4. It shifts the contents of the Accumulator left 4 or 6 places and inserts the 4-bit or 6-bit code for the character read into the low order 4- or 6-bit positions. Then it returns to step (2) above.

NON-ENTERING CHARACTERS

When input is through the 121 Typewriter, the following conditions are true:

1. In the 4-bit mode all bit combinations enter the Accumulator except the O----O zone combinations, Delete, and those combinations not specifically listed in Figure 6.3.

2. In the B-bit mode only legal codes enter the Accumulator.

When input is through the 141 Reader, these conditions are true:

1. In the 4-bit mode all bit combinations which have a "1" zone bit, enter the Accumulator. Thus, tape codes such as 110101 and 111101 are legal input codes for the 141 Reader but can not be read on the 121 Typewriter.

2. In the 6-bit mode the only bit configurations which do not enter the Accumulator are Delete and Conditional Stop.

THE PRINT INSTRUCTION

The Print Instruction selects the output device to be used and the mode of output, and causes one character to be recorded by the selected output device. A negative Print instruction (800P) selects the 4-bit mode of output; a positive Print instruction (P), the 6-bit mode of output. Decimal and hexadecimal data are output in 4-bit mode; alphanumeric data in 6-bit mode. The track address selects the output device: 02 selects the Model 121 Typewriter, 06 selects the Model 151 Tape Punch. The sector portion of the address has no effect on the Print instruction. If 4-bit output is selected the upper 4 bits of the accumulator are output through the selected device with zone bit 10 in channels 5 and 6. If 6-bit output is selected, the upper six bits of the accumulator are output through the selected device.

Examples of Print instructions:

Instruction	Explanation
800P0200	Record via typewriter the character whose zone bits are 1----O and whose numeric bits are in positions 0 through 3 of the Accumulator (4-bit output).
800P0600	Record via 151 Punch the character whose zone bits are 1----O and whose numeric bits are in positions 0 through 3 of the Accumulator (4-bit output).

Instruction	Explanation
PO200	Record via typewriter the character whose 6-bit code is in positions 0 through 5 of the Accumulator (6-bit output).
PO600	Record via 151 Punch the character whose 6-bit code is in positions 0 through 5 of the Accumulator (6-bit output).

If binary-coded decimal information in the computer is to be recorded in decimal; the 800P instruction is used. It is also used to record information in hexadecimal format, regardless of the internal representation of such information. The P instruction is used to record alphanumeric data which is represented internally in this form.

THE SHIFT INSTRUCTION

The I instruction is available in two special forms which can be used to shift the contents of the Accumulator. The negative form, 80016200 causes a 4-place shift, the positive form, 16200, a 6-place shift. The bits which are shifted out of the Accumulator at the extreme left are lost, while thevacated positions on the right are filled with zeros. The track-address portion of the Shift instruction is 62; the sector portion has no effect on the instruction.

Instruction	Explanation
80016200	Shift left 4
16200	Shift left 6

EXAMPLES OF OUTPUT OPERATIONS

1. Location 1806 contains the 3-digit, binary-coded decimal number 125 at a q of 30. Print this number, in decimal: via the typewriter.

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0000		B 1806	'		Bring 125@q = 30
		0001		N 0008	'		Shift the 3 digits into positions 0-11
		0002	800P	0200	'		Print "1"
		0003	800I	6200	'	☒	Shift left 4.
		0004	800P	0200	'		Print "2"
		0005	800I	6200	'		Shift left 4
		0006	800P	0200	'		Print "5"
		0007		Z 0000	'	☒	Halt
		0008		80000	'		1@q=12 in hexadecimal
					'		

2. Location 2753 contains the number 724 in binary at a q of 30. Print this number in decimal via the typewriter.

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	X					
		0000	B	2753	'	Bring 724@30	
		0001	D	0011	'	(724@30)÷(1000@27)= .724@3	
		0002	N	0012	'	(.724@3) N-Multiplied by (10@31) =	
					'	X 7.24@3	
		0003	800P	0200	'	Print "7"	
		0004	E	0013	'	Leaves .24@3	
		0005	N	0012	'	(.24@3) N-Multiplied by (10@31)=2.4@3	
		0006	800P	0200	'	X Print "2"	
		0007	E	0013	'	Leaves .4@3	
		0008	N	0012	'	(.4@3) N-Multiplied by (10@31)=4.0@3	
		0009	800P	0200	'	Print "4"	
		0010	Z	0000	'	X Halt	
		0011		3080	'	1000@27 in hexadecimal	
		0012		F	'	10@31 in hexadecimal	
		0013	WWW	WWWQ	'	Mask in hexadecimal	
					'	X	

3. Record the contents of Location 5513 in hexadecimal on the 151 Punch.

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	X					
		0000	B	5513	'	Bring the word from 5513	
		0001	800P	0600	'	Record first hexadecimal character	
		0002	800I	6200	'	Shift left 4	
		0003	800P	0600	'	X Record second hexadecimal character	
		0004	800I	6200	'	Shift left 4	
		0005	800P	0600	'	Record third hexadecimal character	
		0006	800I	6200	'	Shift left 4	
		0007	800P	0600	'	X Record fourth hexadecimal character	
		0008	800I	6200	'	Shift left 4	
		0009	800P	0600	'	Record fifth hexadecimal character	
		0010	800I	6200	'	Shift left 4	
		0011	800P	0600	'	X Record sixth hexadecimal character	
		0012	800I	6200	'	Shift left 4	
		0013	800P	0600	'	Record seventh hexadecimal character	
		0014	800I	6200	'	Shift left 4	
		0015	800P	0600	'	X Record eighth hexadecimal character	
		0016	Z	0000	'	Halt	
					'		

4. Location 0555 contains the 5-character, alphanumeric word LGP21 at a q of 29. Perform a carriage return and print this word via the typewriter.

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0000		B0012	'	Bring 01000 into positions 0-5 of	
					'	Accumulator	
		0001		P0200	'	Execute carriage return	
		0002		B0555	' ☒	Bring the alphanumeric word	
		0003		P0200	'	Print "L"	
		0004		I6200	'	Shift left 6	
		0005		P0200	'	Print "G"	
		0006		I6200	' ☒	Shift left 6	
		0007		P0200	'	Print "P"	
		0008		I6200	'	Shift left 6	
		0009		P0200	'	Print "2"	
		0010		I6200	' ☒	Shift left 6	
		0011		P0200	'	Print "1"	
		0012	4000	0000	'	HALT – entered as a hexadecimal word.	
					'	Will also cause a carriage return, since	
					' ☒	the 6-bit code for this function (010000)	
					'	is in bits 0 through 5 of this word. To get	
					'	this dual-purpose effect with an instruc-	
					'	tion whose address is not 0000, the add-	
					' ☒	ress must be written in hexadecimal.	

In the above example, "LPG" will print in lower case, as no provision has been made to change to upper case. The alphanumeric information could have specified upper case preceding the "L" and a change to lower case between the "P" and "2". However, this would have resulted in seven alphanumeric characters, requiring representation as two words in memory.

INPUT TO THE COMPUTER

Information may be input to the computer manually or under program control. Both methods will be discussed here.

Manual input

If the typewriter and computer are ON and the Mode switch is positioned to MANUAL INPUT, typing a character on the keyboard causes the bits representing channels 1 through 4 of the character's tape code to appear in the last four bit positions (28, 29, 30, and 31) of the Accumulator. As was pointed out in the discussion of the Input instruction: the information-in binary-coded decimal format-enters the low-order portion of the Accumulator, one character at a time, and moves to the high-order portion as each additional character is entered. If more than eight characters are typed during such an input operation, only the last eight are preserved in the Accumulator, since it has only 32 bit positions. The same characters which enter the computer in response to an Input instruction can also be entered manually (see "Non-entering Characters").

Suppose that a punched tape, such as the one containing the codes for 7 and M which was illustrated earlier, is placed in the typewriter-reader. With the computer in Manual Input mode, depressing the START READ lever on the typewriter activates the reader. This causes "**7M**" to be printed and the four principal bits of each character's tape code to enter the Accumulator, just as if the characters had been typed by hand. Depressing the START READ lever once causes automatic successive reading of the characters punched on tape. Reading will continue until a stop code is read on the tape or until the STOP READ lever or the START COMPUTE lever is depressed.

Notice that the form of input discussed here allows information to enter the Accumulator only; nothing is stored in memory and no instructions are executed.

Program-Controlled Input

As has been shown, characters can be entered into the Accumulator from the keyboard or from tape when the computer is in Manual Input mode. Input can also be activated by programming. For this purpose, an Input instruction must be stored in memory and executed during program operation.

This presupposes that some information is already stored in the computer, namely an Input instruction. But even before such an instruction is in memory, there must be a way to enter information into memory.

STORING INFORMATION IN MEMORY

In the following discussion, a number of computer switches will be mentioned which are instrumental in entering information into any desired memory locations. Since the mechanical aspects of this process are of no particular concern within the context of this chapter, no attempt is made to introduce the subject at this point. A detailed discussion of the computer controls and their functions will be found in Appendix A; a similar discussion of the input/output controls in Appendix B.

Input to Memory from Typewriter

To enter 19 at a q of 21 into Location 2003, a **C2003** instruction must first be placed in the Instruction Register and executed. To do this, the MODE switch is set to MANUAL INPUT, and **C140J** (the hexadecimal form of the decimal instruction **C2003**) is typed. Next the FILL CLEAR switch is depressed. This copies the contents of the Accumulator (the **C2003** instruction) into the Instruction Register and sets the Counter to zero. Then 00004500 is typed. This places 19 at a q of 21 in the Accumulator. Now it only remains to execute the instruction in the Instruction Register. To do this, the EXECUTE switch on the computer must be depressed. However, this switch is not active when the computer is in Manual Input mode. Therefore, to complete the operation, position the MODE switch to ONE OPERATION and depress EXECUTE. The Clear instruction will be executed, and the number 19 at a q of 21 will be stored in Location 2003. If other words are to be stored in other memory locations, the MODE switch must first be positioned to MANUAL INPUT.

Input to Memory from Tape

Returning to the problem of input initiated by programming, suppose that the sequence of instructions 10200, C2003, Z0000, is in memory. If 19 at a q of 21 is to be entered into Location 2003 from tape, a tape punched 00004500' must be in the typewriter-reader. When the execution of the sequence of instructions is initiated, the reader will begin reading the tape almost immediately and will stop when the stop code (') is sensed. A fraction of a second later the computer stops on the ZOO00 instruction, with 19 at a q of 21 in Location 2003 and zero in the Accumulator . The computer had been stopped momentarily by the 10200 instruction, but the reading of the stop code restarted it automatically.

If no such sequence of instructions is in memory and the constant is to be stored into Location 2003 without any typing, a tape punched C140J'00004J00' must be put in the tape reader. To load this constant, the following steps are necessary:

1. Position the MODE switch to MANUAL INPUT.
2. Depress the START READ lever on the typewriter. (C140J enters the Accumulator.)
3. Depress the FILL CLEAR switch on the computer. (C140J, the hexadecimal form of the instruction C2003, is copied into the Instruction Register.)
4. Depress START READ on the typewriter. (00004J00 enters the Accumulator .)
5. Position the MODE switch to ONE OPERATION.
6. Depress the EXECUTE switch. (00004J00 is cleared into Location 2003.)

LGP-21 BOOTSTRAP

Once a pair of I, C instructions is stored in memory, the programmer can store other words under program control. The manual operationof storing the original instructions in memory is called a bootstrap procedure, and the sequence of instructions which is stored is called a bootstrap. A bootstrap consists of a set of instructions which, when stored in memory: transfers control to itself in order to input a hexadecimal fill sequence, which in turn loads a program. While there are several ways of programming a bootstrap, the manual procedure remains the same for all. The discussion in this manual describes the bootstrap which loads Program Input 2 (program J1-10.1).

The bootstrap program consists of three instructions which are stored in Locations 0002, 0003, and 0004, and a fourth instruction which transfers control to Location 0002. The hexadecimal fill sequence consists of eleven instructions, stored in Track 63, and a twelfth instruction to transfer control to the beginning of this sequence.

One reason for using a program input routine in the LGP-21 is to convert decimal instructions to binary. Without such a routine, decimal instructions can not be entered. Consequently, the bootstrap, hexadecimal fill sequence, and the program input routines themselves must be written in hexadecimal. The following discussion will explain the bootstrap, its function, and how it is loaded.

The basic bootstrap consists of three instructions, shown here in decimal notation, to be loaded in Track 00.

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	INSTRUCTION ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0000			'		
		0001			'		
		0002	8001	0200	'		
		0003	C	0005	' ☒		
		0004	8001	0200	'		
					'		

These instructions must be stored in the computer manually. Therefore, each must be preceded by a Clear instruction which will enter the Instruction Register and, when executed, will store a word in the appropriate memory location. Finally, this set of instructions must be followed by an Unconditional Transfer instruction which is executed but not stored in memory. Thus, it takes eight instructions to actually store the bootstrap and transfer control to it. Figure 6-4, below, lists these instructions in proper sequence. Column one contains the decimal equivalent of each instruction; column two, the hexadecimal word as it appears on tape; and column three, the designation of the switches which must be activated, as well as the resultant action.

Decimal nstruction	Hexadecimal Word	Console Switch and Interpretation
		Turn computer and typewriter ON.
		Position MODE switch to MANUAL INPUT.
		Depress START READ. The following instruction enters the Accumulator.
coo02	000C0008'	Depress FILL CLEAR. Places COO02 in the Instruction Register. Depress START READ.
-10200	80010200'	The instruction -10200 enters the Accumulator. Position MODE switch to ONE OPERATION; depress. EXECUTE. Clears -10200 into Location 0002.
		Position MODE switch to MANUAL INPUT. Depress START READ to enter the following instruction into the Accumulator.
coo03	000C000J'	Depress FILL CLEAR. Places COO03 in the Instruction Register. Depress START READ.
coo05	000C0014'	COO05 enters the Accumulator. Position MODE switch to ONE OPERATION; depress EXECUTE Clears COO05 into Location 0003.
		Position MODE switch to MANUAL INPUT. Depress START READ to enter the following instruction into the Accumulator.
coo04	000C0010'	Depress FILL CLEAR. Places COO04 in the Instruction Register. Depress START READ.
-10200	80010200'	-10200 enters the Accumulator. Position MODE switch to ONE OPERATION; depress EXECUTE Clears -10200 into Location 0004.
		Position MODE switch to MANUAL INPUT. Depress START READ to enter the following instruction into the Accumulator.
U0002	000U0008'	Depress FILL CLEAR. Places UOO02 in the Instruction Register. Depress START READ.
Z0000	00020000'	ZOO00 enters the Accumulator. Position MODE switch to NORMAL.

FIGURE 6.4 Basic Bootstrap

At this point the Counter Register contains the address 0002. This indicates that, when the START switch is depressed, the computer will execute instructions beginning in 0002. After the START switch is depressed, the hexadecimal fill sequence is loaded in Track 63, and control is transferred to it.

The decimal coding for the hexadecimal fill sequence is given in Figure 6.5, below:

Location	Command	Address
6300	**8001**	**0200**
6301	GWC	0000
6302	U	**6308**
...	...	...
6308	B	6301
6309	S	**6317**
6310	T	6313
6311	C	6301
6312	U	**6300**
6313	**Z**	0000
6314	U	0000
...	...	...
6317	W	WWWJ

FIGURE 6.5 Decimal Coding for Hexadecimal Fill Sequence

The hexadecimal words which appear on the tape,together with their decimal equivalents, are listed in Figure 6.6. The bootstrap will store this sequence in memory and transfer control to it. The sequence of events is as follows: The first of each pair of instructions (except the last pair) is a Clear instruction. Thus, the instruction in Location 0002 reads it into the Accumulator. Then the instruction in 0003 places the Clear instruction into Location 0005. Next, the instruction in 0004 reads into the Accumulator the instruction which is actually to be stored. The instruction in Location 0005 stores the contents of the Accumulator into the proper location. Finally, the instruction in 0006 transfers control back to 0002 to repeat the process for the next pair of instructions.

Decimal Equivalent	Hexadecimal Word
C0006	000C0018'
uooo2	U0008'
C6300	C3W00'
80010200	80010200'
C6301	C3W04'
191coooo	GWCOOOO'
C6302	C3W08'
U6308	U3W20'
C6308	C3W20'
B6301	B3W04'
C6309	C3W24'
S6317	S3W44'
C6310	C3W28'
T6313	T3W34'
C6311	C3W2J'
C6301	C3W04'
C6312	C3W30'
U6300	U3W00'
C6313	C3W34'
Z0000	
C6314	C3W38'
U0000	U0000'
C6317	C3W44'
WWWWJ	WWWWJ'
U6300	U3W00'
zoooo	

FIGURE 6.6 Hexadecimal Fill Sequence

Notice the last pair of instructions. The **U3W00** is read into the Accumulator by the instruction in 0002; then the instruction in 0003 places it in Location 0005. The instruction in 0004 reads a zero (conditional stop code). The instruction in 0005 **(U6300)** then transfers control to 6300, and Program Input 2 is loadedinto Tracks 00, 01, and 02.

PROGRAM TAPE PREPARATION 7

When a program tape is prepared from a coding sheet (see Figure 3.1) only the information in the "Program Input Codes" and "Instruction" columns and the appropriate stop codes are punched. The information in the "Location," "Contents of Address," and "Notes" columns is not punched as part of a program tape.

TAPE PREPARATION

The procedure for punching a program tape is as follows:

1. Turn the typewriter POWER switch ON.
2. Depress the POWER switch on the computer console.
3. Position the Mode switch to MANUAL INPUT to protect the memory from accidental recording.
4. Depress PUNCH ON.
5. Hold the TAPE FEED lever down long enough to produce a few inches of tape with sprocket holes (a leader).
6. The first punch on every tape should be a Carriage Return code, so that, when the tape is read, information will not start printing in the middle of a line.
7. Program information is entered in this sequence:
   a. Type the entries in the "Program Input Codes" column. There are normally two, each followed by a Conditional Stop Code ('), before the first instruction is encountered. A carriage return is indicated on the coding sheet following the second input code. If there is a further entry in this column, preceding the first "Instruction" on the same line, it is punched next and again followed by a stop code. A stop code is not punched if there is no "Program Input Codes" entry on this line.
   b. Punch the first entry in the "Instruction" column, and follow it with a stop code.
   c. Continue punching information from the "Program Input Codes" and 'Instruction" columns as encountered in left-to-right order. If a line is left blank in this sequence, only a stop code is punched.

There are a number of general rules for punching the above information which, briefly, are as follows:

(1) Leading zeros need not be punched. For example, the hexadecimal word 00013W8J' can be punched without the leading zeros as 13W8J'. (However, the instruction T0016 is punched as T0016, since no leading zeros precede the "T".)

(2) Brackets are considered as containing zeros unless otherwise indicated. That is, for B[. . . .]' = we must punch B0000'. In the exceptional case where an entire word is bracketed which consists entirely of zeros, that is [. . . . .]' = [00000000]', only the stop code need be punched. In all other cases, zeros must be punched to assure that data appears in the proper positions.

(3) All characters may be punched in lower case. (In this manual capital letters are used to indicate operations. This is merely for ease of identification.) In printout, B0627' will appear as b0627'.

(4) Carriage returns, color shifts, back spaces, upper case, lower case, and sections of blank tape (tape feeds) may be punched as desired and will not affect the input operation. On the coding sheet, a carriage return is indicated after every fourth word.

(5) A heading may precede a punched program to identify the tape. It may contain any characters except stop codes. As the tape is read during input through the typewriter, the heading will print but will not affect the input operation.

8. After the last instruction in the program has been punched, depressthe TAPE FEED lever and allow a few inches of tape to pass through the punch to produce a trailer. Tear off the tape.

9. Each tape should be verified in the following manner, after it is punched:

   a. Place the tape in the reader.

   b. Raise the PUNCH ON lever.

   c. Depress the CONDITIONAL STOP lever.

   d. Depress START READ.

   As the tape passes through the reader, a typed copy is produced. The reader will not stop at stop codes, but these codes will appear as apostrophes in the hard copy. This copy may then be checked against the coding sheets. If errors are found, they should be corrected before the program is stored in memory.

CORRECTION OF ERRORS

Errors on punched tape may be corrected in various ways, depending upon the type of error and when it is noticed. Three correction techniques are explained here.

The easiest correction is for an error which is detected immediately after the wrong key has been depressed. In this case, one need only

1. Turn the FEED KNOB on the left side of the punch back one notch.
2. Depress the CODE DELETE lever once.
3. Continue punching by depressing the proper key on the keyboard.

If a wrong key is depressed whose tape code is a portion of the desired combination, the operator need only back the tape until the incorrect character is under the punch head, and overpunch it with the proper key. This method is particularly useful when the error is detected after characters have been punched beyond the error. However, it can only be used when the erroneous and correct tape codes are related in the proper way. For example, a "6" can be overpunched on a "0", "2", or "4", but not on "5" since the tape code for 5 has a punch in channel 4 while that for "6" does not.

A more time-consuming correction method is to reproduce the tape up to the error, punch the correct word, and then continue duplicating. The procedure is as follows:

1. Place the original tape in the reader.
2. Depress the PUNCH ON lever.
3. Depress the TAPE FEED lever and produce a tape leader.
4. Depress the CONDITIONAL STOP lever.
5. Depress the START READ lever. The tape will be read and a duplicate made. When the tape in the reader nears the error, raise CONDITIONAL STOP. The reader will halt when it encounters a stop code. Depress START READ each time another word is to be read.
6. When the last word prior to the error has been read and copied, raise PUNCH ON.
7. Depress START READ once to read past the error.
8. Depress PUNCH ON. Type the correct word.
9. Depress CONDITIONAL STOP and START READ to continue duplicating the original tape.

Appendix B contains a description of the basic input/output unit, including an explanation of the function of each lever.

TIMING AND OPTIMIZATION 8

Optimization is a programming technique which provides access to data and instructions with a minimum of nonproductive searching time. When a program is optimized for the LGP-21, the programmer utilizes the interlace arrangement of sectors around the disc in a manner which will be explained after instruction timing is fully understood.

TIMING

Generally, an instruction is said to be optimum if its four phases can be executed before the disc turns past the location of the next instruction in sequence. However, some instructions-such as multiply, divide, and input-require more than 18 word-times for their operations.

Since timing is an integral part of optimization, the 4-phase instruction **cycle—** already discussed in Chapter 3-is summarized here once more. Each phase is measured in computer word-times. During Phases 1 and 3 the computer searches for a specified sector and then activates the appropriate read/write head. This may take from one to several word-times. Phase 2 always requires one word-time; Phase 4 takes one word-time for all instructions except **N**-multiply, M-multiply, and Divide, which require 63, 65, and 66 word-times respectively.

The time required for the computer to read an instruction, execute it, and be ready to read the next instruction depends on whether an instruction is optimum or not. Figure 8.1 shows the various timing requirements:

Instruction	Optimum	Non-Optimum
Bring, Add, Subtract, Hold, Clear, Extract, Set Return Address, Store Address, and Shift	7.26 ms	58.11 ms
N-multiply, M-multiply, and Divide	58.11 ms	108.96 ms
Unconditional Transfer and Conditional Transfer	1.59 ms	Each sector beyond optimum adds **.40 ms**
Sense	7.26 or 14.52	

FIGURE Optimum Timing

An instruction can be optimum only if its operand is located within a certain number of word-times. Figure 8.2 lists the range of optimum sectors for all instructions which can be optimized.

Instruction	Distance from Instruction Location to Optimum Operand in Word-Times
Bring: Add, Subtract, Hold, Clear, Extract, Set Return Address, and Store Address	2 through 16
N-multiply	2 through 81
M-multiply	2 through 79
Divide	2 through 78
Unconditional Transfer and Conditional Transfer	4 or more if transfer is active
Others	always optimum

FIGURE 8.2 Range of Optimum Sectors

Input Timing

An input instruction is held in Phase 3 of its cycle until a start signal is received from the input device. Therefore, the computer is not free to perform internal calculations during an input operation. The total time for executing any input instruction consists of three word-times plus the Phase 1 search and the time required for reading or typing.

Output Timing

Because of the buffering system built into the LGP-21, an output operation will not delay the computer unless the selected device is already in use. Thus, the computer is free to perform other internal calculations while the information is being output. During the time the output device is busy, the interlock for that device is turned ON. If that device is selected for output a second time while its interlock is ON, the computer will delay execution of the second print until the interlock is turned OFF. The interlock is turned OFF automatically when the device is free. If, during the time an output device is busy, a second instruction selects a different device for input or output, the second instruction will be executed without delay if that device is not busy. Therefore, the operation of two or more input/output devices can overlap in time. For example, if an output to the tape typewriter is followed by an output to the 151 Tape Punch, the computer will not delay on the second output instruction even though the typewriter is still busy. If the first and second output instructions both select the typewriter, the computer may delay on the second instruction until the typewriter is ready.

Print instructions must be 1/3 revolution apart for the 151 Tape Punch and not more than 2 revolutions apart for the 121 Tape Typewriter, if these devices are to operate at their rated speeds.

OPTIMIZATION

At the beginning of this manual, it was briefly mentioned that the physical characteristics of the memory disc are disregarded for general programming purposes as they vary from the 64 track/64 sectors concept used. Actually, the disc consists of 32 tracks with 128 sectors each. These sectors are not numbered sequentially within a track although the pattern of numbering is the same for all tracks. This system is based on an 18-word interlace pattern which positions consecutive words 18 sectors apart-a feature which aids in optimizing of instructions which are executed in sequence.

Fortunately, the programmer does not have to memorize this complex pattern in order to utilize optimizing for his programs. He can use the device illustrated in Figure 8.3, which will let him determine at a glance the optimum sectors for the operand of any instruction.

The device-called the Optimum Address Locator-shows the interlace pattern of the sectors on the memory disc. Each sector is represented by threedigits of which the second and third represent an actual LGP-21 sector number. The initial digit-which is either 0 or l-indicates whether the track used in conjunction with the sector is even- or odd-numbered. A 0 indicates that the sector is on an even-numbered track; a 1 an odd-numbered track.

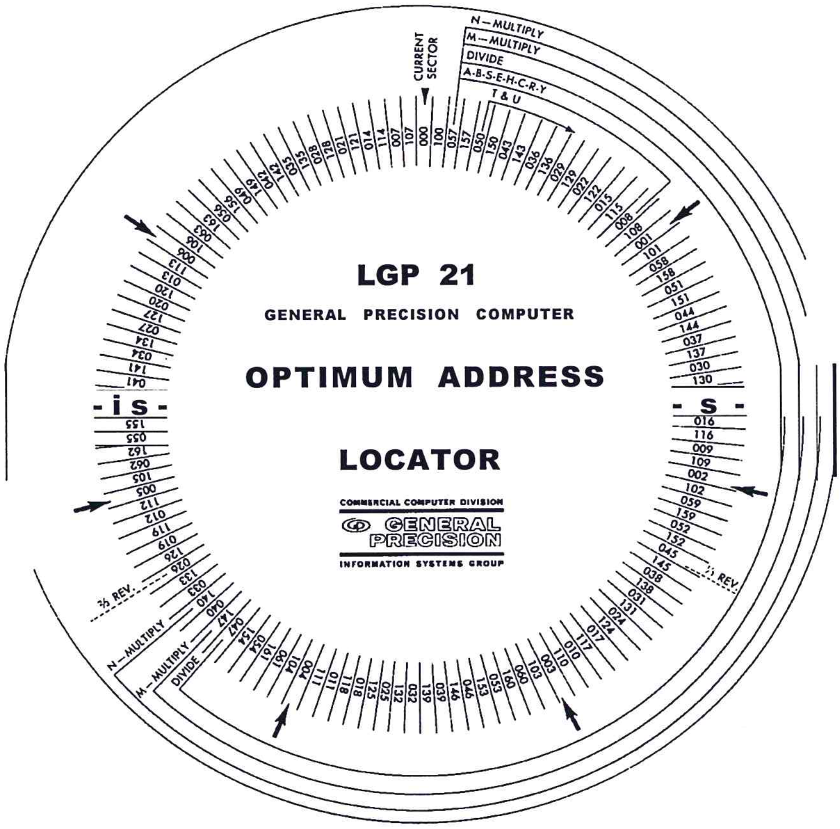


FIGURE 8.3 Optimum Address Locator

To determine the optimum operand address for an instruction with the help of the Optimum Address Locator, one must first locate on it the sector address of the instruction.

Assuming a Hold instruction is in location 1400, sector 00 must be found on the rotating center wheel of the device. Since track 14 is even-numbered, the sector address 00 must be preceded by another 0. The next step is to center 000 above the "Current Sector" indicator (on the larger, outside wheel). Adjacent to this indicator are five lines which delimit as many segments; each segment being headed by the name (or names) of the LGP-21 command to which it pertains. For example, the first segment applies to the T and U instructions; the fifth to N-Multiply.

The "H" for the Hold instruction can be found above the second segment. This indicates that the optimum operand sectors for that command are contained within the second segment area, namely between 057 and 008. In addition, all other sectors within this range also yield an optimum operand address for the Hold instruction above; namely, 057, 157, 050, 150, 043, 143, 036, 136, 029, 129, 022, 122, 015, 115 or 008.

If the Hold instruction had been in an odd-numbered location, say 1500, sector 00 on the center wheel would have to be preceded by a 1 to indicate the odd track number. With 100 centered underneath the "Current Sector" indicator, and the H command again located above the 2nd segment of the larger, outside wheel, the first possible optimum operand address for the instruction would have been 157, and the last 108. In addition, all sectors listed between 157 and 108 would be optimum for H1500.

At this point, one further general rule for using the Optimum Address Locator should be adopted as good practice: that is, to never use the outer limits of the optimum range for any instruction, if possible. The reasons for this rule will be explained later in this chapter, when a few additional concepts have been understood. Finally, the larger outside wheel of the Optimum Address Locator provides the sector location in which the next instruction to be executed will be found. This can be obtained from the black arrows which are placed, for clockwise reading, around the periphery of this disc. In other words, when the Current Sector indicator for H 1400 is placed on 000, the black arrows show that the next instruction would be located in 1401, etc.

Thus, by using the Optimum Address Locator, an operand address can be chosen which permits execution of the Hold instruction before the next instruction (in Location 1401) passes the read head. On the other hand,if an H1658 is in Location 1400, it is not an optimum instruction since sector 01 would have passed the read head before the computer could complete the execution of H1658; therefore, the disc would make a full revolution before Location 1401 could be read. Most instructions can be executed in approximately 1/7 of a disc revolution when operand addresses are optimum. This constitutes a significant saving in machine-time.

The U and T instructions are optimized somewhat differently than other instructions. If a U instruction is in sector 00 of an even-numbered track, the fastest transfer will be to sector 50 on an even or odd track; if the U instruction is in sector 00 of an odd-numbered track (e.g. 0100), the fastest transfer will be to Sector 50 on an odd-numbered track (e.g., 0150). Each subsequent sector will add one word-time to the minimum transfer time. The search for the sector of a T instruction occurs only during an actual transfer. If T1251 is given in Location 1700 and a transfer is not actually made, then the instruction in 1701 will be read as it passes the read head on that disc revolution.

For one more example of optimum programming, consider the following sequence of instructions.

PROGRAM INPUT CODES	STOP	LOCATION	INSTRUCTION OPERATION	ADDRESS	STOP	CONTENTS OF ADDRESS	NOTES
	'						
	'	☒					
		0000	B	1943	'		
		0001	A	2251	'		
		0002	C	1943	'		
						☒	

The instructions B1943, A2251, and C1943 must be brought into the Instruction Register before they can be executed. This can happen only when Locations 0000, 0001, and 0002 respectively are under the read head. If the read head is over sector 00 and is placing the instruction B1943 in the Instruction Register, then, if this instruction can be executed before the disc turns to sector 01, it will be an optimum instruction. Figure 8.3 shows that sector 43 for an odd-numbered track is between 00 and 01 and is within the optimum range for sector 00. The same logic applies to the next instruction, A2251 at Location 0001. Sector 51 for an even-numbered track lies between sector 01 and sector 02 and is within the optimum range for sector 01. Therefore, both these instructions are optimum. However, the next instruction, C1943 in Location 0002, will not be optimum because sector 43 does not lie between sectors 02 and 03.

Two further concepts need to be introduced now, to explain why optimum addresses should not be selected on the extreme outer limits of the optimum range, whenever possible. The first pertains to the difference between relocatable and non-relocatable programs. A non-relocatable program is coded for storage in a particular area of memory and will not operate properly if stored anywhere else. A relocatable program can be stored anywhere in memory. It is normally written relative to Location 0000; that is, as if the first instruction will be stored in Location 0000. In actual operation, it can then be stored by a program input routine beginning with any location the programmer specifies. (The program input routine description explains how this is done.)

The other new concept pertains to address modification. As a relocatable program is being loaded, some operand addresses must be modified by the program input routine for proper operation of the program. However, ***some*** **addresses—** for example, those in Input and Print instructions-can not be changed because they represent standard selection codes for certain input or output devices. The same is true for sense Branch Switch instructions, since their address determines which Branch Switch is tested. Consequently, the program input routine must be informed of any addresses which are not to be modified. This is done by preceding such commands with an **"X"**:

```
 XZ0800'
80X10200'
 XA2001'
 XR1011'
 XU1000
```

NOTE: The X is recognized by the program input routine, but it is not stored in memory.

Now it can be shown that, if a non-modifiable instruction has been written with an extreme outer-limit optimum operand address, and the program is relocated by an odd number of tracks, the instruction would no longer be optimum. For example, if the instruction

Location	Instruction
0200	XA0457

is part of a relocatable program, and if the program is relocated upward by three tracks, the instruction would appear as

Location	Instruction
0500	XA0457

and would not be optimum. For this reason, optimum addresses should not be chosen on the outer limits of the optimum range.

In closing, it might be mentioned that all LGP-21 subroutines programmed by the General Precision Commercial Computer Division have been optimized to effect savings in machine-time for the user. This is, however a more time-consuming effort than straightforward programming. Consequently, if a programmer decides to optimize a program, he should first compare the possible savings in machine-time with the added programming costs.

APPENDIX A

COMPUTER CONTROL PANEL

The LGP-21 computer is operated through switches which are located on the control panel. These switches are clearly identified by function or related action. Figure A. 1 illustrates the control switches.

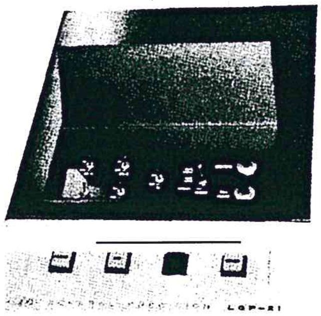


FIGURE A.1 LGP-21 Control Panel

MODE

This is a three-position toggle switch.

NORMAL - When the MODE switch is set to this position, the computer executes instructions at high speed. When the START switch is depressed, the execution of instructions will begin with the instruction whose address is in the Counter.

ONE OPERATION - If the computer is operating in Normal mode and the MODE switch is moved to ONE OPERATION, the computer will stop after the next Phase 4, the execute phase. If the computer is stopped in One Operation mode, depressing the START switch will start the computer cycling through the instruction whose address is in the Counter Register, and computation will stop after the execute phase for that instruction. The EXECUTE switch is operative only in One Operation mode. Changing from Manual Input to One Operation mode will deselect the tape typewriter.

MANUAL INPUT - This position sets the Accumulator to receive input. It also selects the typewriter for 4-bit input, but does not deselect other devices. If other devices are selected, the I/O switch should be depressed to deselect them or information may not enter correctly. When the computer is in Manual Input mode, all typed characters, except non-entering characters, enter the Accumulator. No instruction can be executed in Manual Input mode, since the START switch is inoperative.

FILL CLEAR

FILL CLEAR is a momentary switch. In Manual Input mode it transfers the contents of the Accumulator to the Instruction Register and resets the Counter Register to zero; in One Operation mode it only sets the Counter Register to zero. This switch is inoperative in Normal mode.

EXECUTE

This momentary switch causes the instruction in the Instruction Register to be executed. It is operative only in One Operation mode.

TRANSFERCONTROL

The TC switch can be set ON or OFF. This switch is used in conjunction with the negative T (Conditional Transfer) command. A negative T instruction will cause the computer to get the next instruction from the location designated by the operand address if the TC switch is ON, or if the contents of the Accumulator is negative. If the contents of the Accumulator is positive and the TC switch is OFF, the computer will continue to the next instruction in sequence.

BRANCH SWITCHES

The four branch switches are labeled BS-32, BS-16, BS-8, and BS-4. Each is a two-position switch which can be set ON or OFF. These switches are used in conjunction with the **Z** (Sense and Transfer) command. A **Z** instruction whose track-address corresponds to one or more of the branch switches will cause the computer to skip the next instruction if any designated switches are OFF, or to execute the next instruction if all designated switches are ON.

POWER

This switch turns power ON or OFF. Power for all units in the system is in series with this switch. Any units previously set ON will have their power turned ON as the switch is depressed. About thirty seconds after power is turned ON, the POWER switch lights to indicate the machine has attained full speed.

I/O

I/O is a momentary switchwhich clears the Accumulator and deselects all input/output devices. If the computer is in Manual Input mode, depressing I/O will not deselect the typewriter. The switch is lighted and operative during input and output and when the computer is in Manual Input mode.

STOP

This indicator lamp lights immediately when the computer is turned ON and is lit whenever the computer is not executing instructions.

START

START is a momentary switch which causes the computer to execute the instruction specified by the Counter Register. In Normal mode this will begin the full-speed execution of instructions. In One Operation mode only one instruction will be executed. The switch is not operative in Manual Input mode. The light beneath the switch is ON whenever the computer is operating.

In addition to the control panel switches, there are two toggle switches on the back of the computer:

INTERLOCK

The LGP-21 has a circuit breaker to interrupt operation if the air-flow from the fan becomes blocked. This interruption stops computer operation to prevent overheating. Following such an operation, the condition that caused it should be corrected; then the circuit may be reset by moving the INTERLOCK switch from the up position down and up again. It should be noted that, depending upon the operation in effect when the interruption occurred, information stored in memory may have been destroyed and may have to be re-entered.

RECORD ENABLE

This switch may be set ON or OFF. When it is ON, reading from and recording in all sectors of all tracks may occur. When it is OFF, 1024 words—specifically Tracks 00 through 15-are protected. That is, information may be transferred from any word within this area to the Instruction Register (to be executed as an instruction) or to the Accumulator (to be acted upon), but no information can be recorded in any word in this area. This feature allows the operator to lock a program in memory so that it cannot be destroyed inadvertently.

The primary input/output for the LGP-21 is the Model 121 Tape Typewriter. In addition to a standard typewriter keyboard, the unit has a paper-tape punch, paper-tape reader, and various levers for controlling their operations.

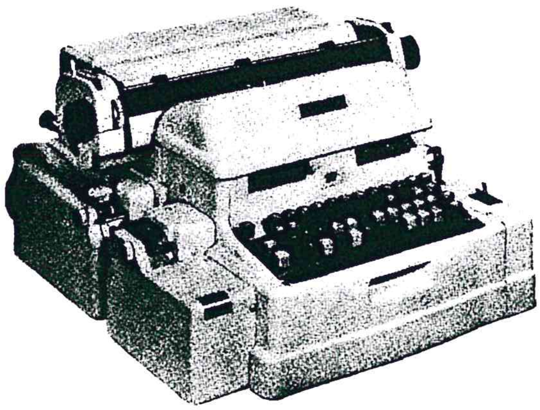


FIGURE B.1 Model 121 Tape Typewriter

POWER ON-OFF

This switch, in the lower right-hand corner adjacent to the keyboard, sets the typewriter so that power will be turned ON or OFF when the, computer power is ON. The carriage is interlocked and should not be moved when power is OFF.

START COMP

When the computer has selected the typewriter for input, the START COMP lever terminates input and allows the computer to proceed to the next instruction. The START COMP lever will stop the paper-tape reader whenever it is running on-line or off-line. The functions of the START COMP and the STOP READ levers are identical.

MANUAL INPUT

If this lever is down and the typewriter is selected for input, information can be transmitted to the computer from the keyboard only. If this lever is raised, information is transmitted from the tape reader when the typewriter is selected for input.

CODE DELETE

Operative only when the PUNCH ON lever is depressed, this lever is used to delete an error in the tape by punching holes in all six channels. One delete code is punched each time this lever is depressed. However, by holding down TAPE FEED and CODE DELETE at the same time, the operator can produce a series of code deletes. If, while the punch is ON, a tape interlock occurs because of a tight tape condition, depressing the CODE DELETE lever will release the typewriter.

TAPE FEED

This lever feeds tape into the punch, which then punches only sprocket (feed) holes. TAPE FEED is operative only when PUNCH ON is depressed.

MANUAL INPUT LIGHT

This light is on when the typewriter is selected for input to the computer and the MANUAL INPUT lever is down.

PUNCH ON

The PUNCH ON lever activates the tape punch, allowing any character typed from the keyboard or read from the tape reader to be punched. TAPE FEED and CODE DELETE are operative only when PUNCH ON is depressed. Raising the lever turns the punch OFF.

STOP READ

This lever is used interchangeably with START COMP.

START READ

This lever starts the tape reader providing the MANUAL INPUT lever is UP; otherwise, it turns ON the Manual Input light. Reading will continue until a Conditional Stop code is read, providing the COND STOP lever is raised, or until the STOP READ, START COMP, or MANUAL INPUT lever is depressed.

COND STOP

This lever, when depressed, allows the tape reader to read without being stopped by the Conditional Stop codes. This lever must be raised during input to the computer from the tape reader.

PAPER GUIDE

Located just to the rear of the platen, this guide should be adjusted horizontally so that it touches the left edge of the paper form.

TAB STOP

Under the cover to which the paper guide is attached is the tab rack, numbered 8 through 136 in increments of 4. A tab stop is a metal positioner that can be inserted in any notch along the tab rack. When the TAB key is depressed, a Tab code read from tape, or a tab output from the computer, the carriage will move to the next position containing a tab stop.

LEFT MARGIN STOP

In front of the tab rack is the margin rack, numbered 0 through 68 in increments of 4. The margin stop is the sliding assembly mounted on the margin rack. To move this assembly, press down on its center and slide it along the rack. The right end of the stop is the indicator. The setting of the margin stop determines the left margin position.

AUTOMATIC CARRIAGE RETURN

Behind the tab rack is a carriage return plate. An automatic carriage return positioner can be placed anywhere along the plate. An automatic return occurs when the carriage reaches this return positioner as the result of a tab jump; i.e., because the Tab key is depressed, a Tab code is read from tape, or the computer outputs a Tab code. If this positioner is reached as a result of single-character steps, the typewriter may jam. This condition may be cleared by striking the Carriage Return Key manually. However, any input or output that occurred at the time of the jamming may be invalid.

PAPER SCALE

The paper scale is printed on the metal shield in front of the platen. By viewing the paper scale through the type guide, one can determine the exact position of the carriage and where characters will print.

TYPE GUIDE

This guide indicates the position of the carriage and the location where the characters will print.

WRITING LINE

The bottom of the typed line will be exactly above the top edge of the writing line finder. It is used to align a previously typed page in the platen for additional typing.

PAPER RELEASE

The paper release is located at the top left-hand corner of the movable carriage assembly. When this lever is pulled forward, the paper can be straightened or removed.

LINE SPACER

To the right of the paper release lever is a lever which permits selection of single-, one and one-half-, or double-spacing between lines.

PLATEN

The platen is a roller-type device which holds the paper against which the type bars strike.

CARRIAGE RELEASE (Right and Left)

There are two Carriage Release buttons, one located to the right and one to the left of the platen. When either or both are held down, the entire carriage assembly can be freely moved. The carriage should not be moved when power is OFF.

PLATEN KNOBS (Right and Left)

The platen knobs, located at each end of the platen, are used for turning the platen forward or backward.

PLATEN VARIABLE

When this button, located in the center of the left platen knob, is depressed, the platen is released to allow the operator to position the paper at other than standard line spacing. Releasing the button restores standard line spacing.

MARGIN RELEASE

This lever, which-is located behind the left platen knob, can be raised to move the carriage to the left of the margin stop.

RIBBON POSITIONER

The ribbon position lever, located on the right side of the typewriter below the carriage, positions the ribbon for typing through its upper or lower part or for typing stencils.

SPACE

This bar moves the carriage forward one character space.

COND STOP (')

This key is used to punch a Conditional Stop code (') into paper tape. When sensed by the tape reader, this code stops the reader and sends a start signal to the computer.

TAB

This key moves the carriage to the next established tab position.

COLOR SHIFT

This key shifts and locks the ribbon for typing through its upper or lower half.

CAR RET

This key returns the carriage to the left margin and spaces the paper to the next typing line.

BACK SPACE

This key moves the carriage back one character space each time it is depressed.

LOWER CASE, UPPER CASE

These keys lock the keyboard in position for typing lower or upper case characters. LOWER and UPPER CASE keys are provided on both sides of the keyboard.

TAPE INTERLOCK

The punch contains a tape interlock that stops the device if the tape breaks or if the supply is exhausted.

FEED KNOBS (Reader and Punch)

The reader and punch feed knobs are located to the left of the read and punch heads, respectively. These knobs can be used to manually move tape forward or backward.

TABLE I Command and Address Equivalences

TABLE Ia Command Equivalences

Symbol	Command	Binary	Hexadecimal	Decimal
Z	Stop, Sense and Transfer	0000	0	1
B	Bring	0001	1	2
Y	Store Address	0010	2	3
R	Set Return Address	0011	3	4
I	Input, Shift Left	0100	4	5
D	Divide	0101	5	6
N	Multiply: Save Right	0110	6	7
M	Multiply: Save Left	0111	7	8
P	Print	1000	8	9
E	Extract	1001	9	10
U	Unconditional Transfer	1010	F	11
T	Conditional Transfer	1011	G	12
H	Hold	1100	J	13
C	Clear	1101	K	14
A	Add	1110	Q	15
S	Subtract	1111	W	16

TABLE Ib Address Equivalences

DECIMAL	HEXADECIMAL Track	HEXADECIMAL Sector	DECIMAL	HEXADECIMAL Track	HEXADECIMAL Sector
0	00	00	32	20	80
1	01	04	33	21	84
2	02	08	34	22	88
3	03	0j	35	23	8j
4	04	10	36	24	90
5	05	14	37	25	94
6	06	18	38	26	98
7	07	1j	39	27	9j
8	08	20	40	28	f0
9	09	24	41	29	f4
10	0f	28	42	2f	f8
11	0g	2j	43	2g	fj
12	0j	30	44	2j	g0
13	0k	34	45	2k	g4
14	0q	38	46	2q	g8
15	ow	3j	47	2w	gj
16	10	40	48	30	j0
17	11	44	49	31	j4
18	12	48	50	32	j8
19	13	4j	51	33	jj
20	14	50	52	34	k0
21	15	54	53	35	k4
22	16	58	54	36	k8
23	17	5j	55	37	kj
24	18	60	56	38	q0
25	19	64	57	39	q4
26	1f	68	58	3f	q8
27	1g	6j	59	3g	qj
28	1j	70	60	3j	wo
29	1k	74	61	3k	w4
30	1q	78	62	3q	w8
31	1w	7j	63	3w	wj

TABLE II Powers of 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 786	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5

TABLE III Input/Output Codes

TABLE IIIa Input/Output Codes for the 121 Typewriter

Character Codes			Tape Codes 6 1234 5	Input Codes 1234 56	Output Codes 1234 56
Tape Feed			0 0000 0	*oooo 00	
)		0	0 0000 1	0000 10	0000 10
L		1	0 0001 1	0001 10	0001 10
*		2	0 0010 1	0010 10	0010 10
"		3			
Δ		4	11 0011 0100 [illegible]	0011 0100 10 10	0011 0100 10 10
%					
$	5	6	11 0101 0110 11	0101 0110 10 10	0101 0110 10 10
π					
Σ	7	8	11 0111 1000 11	0111 1000 10 10	0111 1000 10 10
(		9	0 1001 1	1001 10	1001 10
F		f	0 1010 1	1010 10	1010 10
G		g	0 1011 1	1011 10	1011 10
J		.	0 1100 1	1100 10	1100 10
K	J				
Q	k	q	11 1101 1110 [illegible]	1101 1110 10 10	1101 1110 10 10
W		w	0 1111 1	1111 10	1111 10
Z		z	1 0000 0	0000 01	0000 01
B		b	1 0001 0	0001 01	0001 01
Y		y	1 0010 0	0010 01	0010 01
R		r	1 0011 0	0011 01	0011 01
I		i	1 0100 0	0100 01	0100 01
D		d	1 0101 0	0101 01	0101 01
N		n	1 0110 0	0110 01	0110 01
M		m	1 0111 0	0111 01	0111 01
P		p	1 1000 0	1000 01	1000 01
E		e	1 1001 0	1001 01	1001 01
U		u	1 1010 0	1010 01	1010 01
T		t	1 1011 0	1011 01	1011 01
H		h	1 1100 0	1100 01	1100 01
C		c	1 1101 0	1101 01	1101 01
A		a	1 1110 0	1110 01	1110 01
S		s	1 1111 0	1111 01	1111 01
Lower Case			0 0001 0	"0001 00	0001 00
Upper Case			0 0010 0	*0010 00	0010 00
Color Shift			0 0011 0	*0011 00	0011 00
Car. Return			0 0100 0	*0100 00	0100 00
Back Space			0 0101 0	*0101 00	0101 00
Tab 00			0 0110 0	*0110 00	0110 00
Cond. Stop			0 1000 0		1000 00
Space			1 0000 1	0000 11	0000 11
—			1 0001 1	0001 11	0001 11
=		+	1 0010 1	0010 11	0010 11
			1 0011 1	0011 11	0011 11
?	;	,			
]	/	' .	[illegible]	0110 [illegible]	[illegible]
√		v	1 0111 1	0111 11	0111 11
O		o	1 1000 1	1000 11	1000 11
X		x	1 1001 1	1001 11	1001 11
Delete			1 1111 1		

*6-bit input only

TABLE IIIb Additional Input/Output Codes

In addition to the codes in Table IIIa, the following codes can be output by the computer through the 151 Punch and input through the 141 Reader. They cannot be input or output via the 121 Typewriter.

	Tape Code	Input Code	Output Code
	6 12345	1234 56	1234 56
Tape Feed	0 00000	*0000 00	0000 00
	0 01110	*0111 00	0111 00
	0 10010	*1001 00	1001 00
	0 10100	*1010 00	1010 00
	0 10110	'1011 00	1011 00
	0 11000	*1100 00	1100 00
	0 11010	*1101 00	1101 00
	0 11100	*1110 00	1110 00
	0 11110	*1111 00	1111 00
	1 10101	1010 11	1010 11
	1 10111	1011 11	1011 11
	1 11001	1100 11	1100 11
	1 11011	1101 11	1101 11
	1 11101	1110 11	1110 11
Code Delete	1 11111		1111 11

*6-bit input only