General Precision Electronic Computer

LGP-30

PROGRAMMING CLASS NOTES

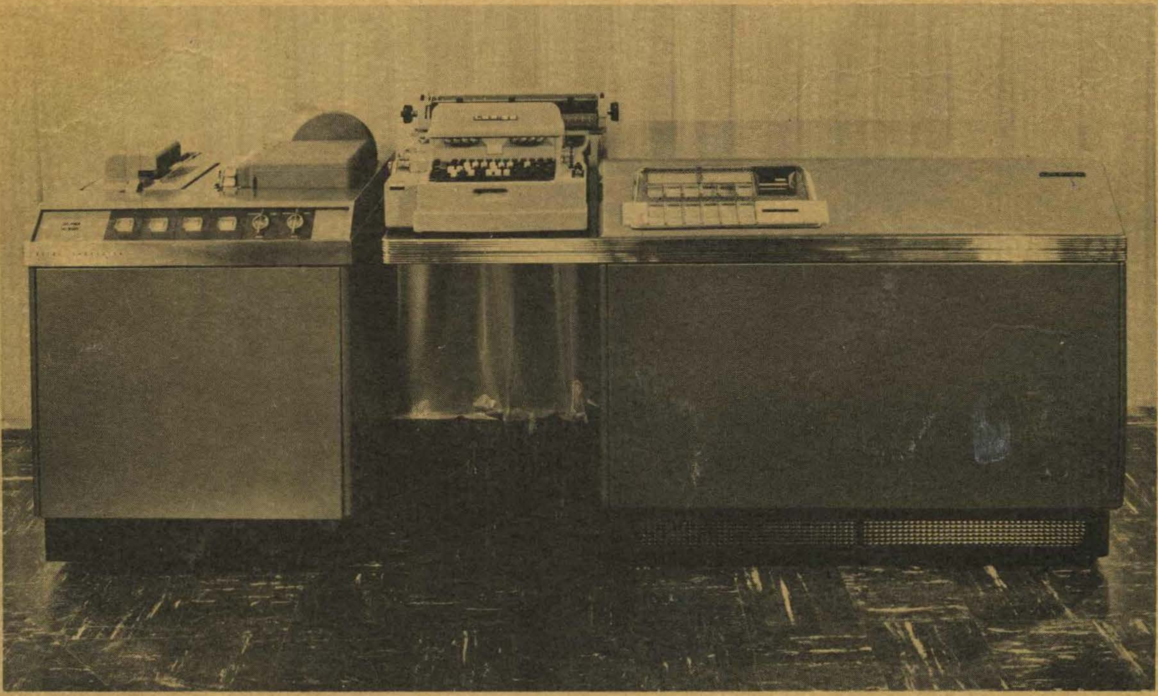


TABLE OF CONTENTS

Page

INTRODUCTION

SECTION I .... LGP-30 PHYSICAL DESCRIPTION 1

SECTION II .... MAGNETIC DRUM AND WORD STRUCTURE 2

SECTION III .... SOME BASIC ORDERS 7

SECTION IV .... SCALING 9

SECTION V .... PROGRAMMING A SIMPLE PROBLEM 12

SECTION VI .... TAPE CODES 14

SECTION VII .... TAPE READ IN 17

SECTION VIII .... CONVERSION OF DECIMAL INSTRUCTION AND DATA TO HEXADECIMAL 19

SECTION IX .... OSCILLOSCOPE 22

SECTION X .... CONSOLE BUTTONS 23

SECTION XI .... TYPEWRITER CONTROLS 30

SECTION XII .... PROGRAMMING A LOOP 35

SECTION XIII .... BOOTSTRAP PROGRAM 38

SECTION XIV .... PROGRAM INPUT ROUTINE 42

SECTION XV .... OPERATING TIMES-OPTIMIZING 48

SECTION XVI .... PRINT INSTRUCTION 52

SECTION XVII .... BINARIZATION OF DECIMAL ADDRESS 54

SECTION XVIII .... CORRECTING TAPES 57

SECTION XIX .... SYNOPSES OF LGP-30 SUBROUTINES 59

INTRODUCTION

These class notes are designed to dispense with the necessity for note taking and to increase the general efficiency of the instruction period. They do not constitute the Royal McBee LGP-30 Programming Manual.

The following paragraphs briefly summarize the organization of the major topics covered by the sections of this manual.

After disposing of the basic physical description and "Computer Control" the concept of data and instruction words are introduced. Some basic LGP-30 orders along with scaling are introduced before proceeding to a specific example. Having programmed a basic problem, the next step is to show how this program is represented on tape and read into the LGP-30.

The sections covering conversion to hexadecimal, the oscilloscope, console buttons, typewriter controls, and programming a loop are presented to prepare the reader to master the section concerning the bootstrap program.

The section devoted to the program input routine is possibly the most important as it is most widely used.

A section covering operating times and optimizing is presented to give the reader the background required to fully grasp the theory of the print instruction.

The remaining sections contain material with which LGP-30 operators and programmers should be acquainted.

SECTION I

LGP-30 PHYSICAL DESCRIPTION

The LGP-30 is commonly referred to as a desk computer. Obviously, this phrase originated because the size of the LGP-30 closely parallels the size of a standard office desk. The LGP-30 is 26" deep, 33" high, and 44" long, exclusive of the typewriter shelf. The computer weighs approximately 800 pounds and is mounted on sturdy casters which facilitates movement of the computer.

Another asset in regard to the computer's mobility is its power requirement. The LGP-30 requires 1500 watts when operating under full load. The power inlet cord may be plugged into any standard 115 volt 60 cycle single phase line. The computer contains internal voltage regulation of all voltages against power line variations from 95 to 130 volts. In addition to regulation of power line variations, the computer also contains the circuitry required to permit a warm-up stage. This warm-up stage minimizes thermal shock to tubes and insures long component life. The computer contains its own blower unit and directs filtered air, through ducts, to tubes and diodes, in order to again insure long component life and proper operation. No expensive air conditioning needs to be installed if the room is kept within a reasonable temperature range.

The computer contains 113 electronic tubes and 1450 diodes. The 113 electronic tubes are mounted on 34 etched circuit pluggable cards which also contain associated components. Although 34 pluggable cards are used, there are only 12 different types of such cards. Card extenders are available to permit dynamic testing of all machine functions. Six hundred and eighty of the 1450 diodes are mounted on one pluggable logic board. This logic board may be quickly removed and is very accessible for service personnel's use.

The main memory consists of a magnetic drum that contains 4096 words. An instruction or data word may be stored in each one of the 4096 available words. When instructions are placed on the drum, they will be executed in sequence until one of the instructions "transfers control" to a new set of instructions.

SECTION II

MAGNETIC DRUM AND WORD STRUCTURE

Since we will be concerned with programming for the computer, the most important component from our standpoint will be the magnetic drum. Physically the drum is approximately 6.5" in diameter and 7" in length. The 4096 word main memory is contained on this drum. To facilitate machine design (as we shall soon see) this main memory was broken into 64 tracks or rings around the circumference of the drum. These tracks are not visible, actually each track is only .040" in width. There is .075" between each track. Approximately .001 of an inch above each track is mounted a head. This head is capable of reading and recording magnetic spots on the surface of the drum. Since these heads are much wider than the .040" track width, it is impossible to place all 64 heads in one continual line above the drum. Instead they are spaced at various points around the drum. The heads are manufactured by Librascope Inc. and each head is encapsuled for moisture resistance and dimensional stability. Low resistance windings and sintered ferrite cores with low capacitance assure stable operation.

The 64 tracks on the drum are numbered from 00 through 63. At first this may seem odd but shortly you will see the reason for numbering the tracks in this fashion. Each track is broken into 64 equal segments or sectors as we shall refer to them. This gives us 64 tracks each containing 64 sectors or a total of 4096 sectors. As you already know, the LGP-30 computer has a 4096 word memory; consequently, each sector may also be referred to as a word. Again as you already know, our computer is a stored program computer, i.e. we may use each word of memory to store an instruction of our program or a numerical value (word of data).

The computer interprets a word of data in an entirely different manner than it does a word containing an instruction. So let us first see how the computer interprets a word of data. If we were able to pick up any sector (or word) of memory and examine it as the computer does, we would find that the word was composed of 32 equal pieces or bits. The read record head is capable of recording or magnetizing each one of these individual bits independently and later is capable of reading those same magnetized spots. In the case of a data word, if the high order bit (i.e. bit at the extreme left) is magnetized, it indicates that the data word is negative. Conversely an unmagnetized high order bit indicates the presence of a positive number. Perhaps you have seen in the blue brochure that the LGP-30 number range is from -1 to almost +1. If this were really true, I am sure you all might have a little difficulty finding a job to run on the computer.

The computer represents all numbers in binary and assumes the decimal point (or binary point -- since we have a binary number) to be located between the sign bit and the bit just to the right of it. The computer considers the bit to the right of the sign bit to have a value of 2^{-1} and the bit to the right of it to have a value of 2^{-2}, the next 2^{-3}, 2^{-4}, etc.

We shall represent a magnetized bit as a 1 and an unmagnetized bit as zero. Consequently, if we have the 4 bits to the right of the sign bit magnetized and all others unmagnetized (including sign bit) the computer would interpret this as $2^{-1} + 2^{-2} + 2^{-3} + 2^{-4}$ or 1/2 + 1/4 + 1/8 + 1/16 = .5 + .25 + .125 + .0625 = .9375. With this notation it would not be possible to ever express a positive integer. Since there is never an actual binary point placed on the drum, the machine does not know whether we consider the binary point to be located between the sign bit and the bit to the right of it or not. So we may consider the binary point to be located anywhere we choose.

As an example, let us suppose we consider the binary point to be located X positions to the right of the standard position. Rather than saying each time that "the binary point is X places to the right of the standard position of the binary point", we have adopted the notation "at a 'q' of +X." Conversely, "a 'q' of -X" indicates that the binary point is considered to be X places to the left of the standard binary point location. As an example, if we consider "q" to be + 3, we have 3 bits between the sign bit and the binary point. The bit just to the left of the binary point represents 2^0, the bit to the left of that 2^{+1} and the one to the left of that 2^{+2}. Therefore, if all 3 of these bits are magnetized, we are representing the integer $2^0 + 2^1 + 2^2 = 1 + 2 + 4 = 7$. In this example, the bits to the right of our binary point represent 2^{-1}, 2^{-2}, 2^{-3}, etc. So if we choose to carry a number at a q of 3, we may represent any number less than $2^3 = 8$.

We cannot represent $2^3 = 8$ or any larger number at a q of 3 so in a binary fixed point machine you must have some idea of the range of all of your data. Once you know the range (maximum value in particular) of a variable it is a simple matter to refer to a table of powers of 2 and determine what "q" (or power of 2) the number must be "carried at". By following these basic rules, one may easily express positive numbers in the range $0 < N < 2^{30}$ with nearly 9 decimal digits accuracy. You will note that we say 2^{30} and not 2^{31} even though it appears that we have 31 bits to the right of the standard binary position. Actually the low order bit (extreme right) is <u>always</u> recorded on the drum as a zero. It is usually referred to as a spacer bit since it effectively gives us a space between the words (or sectors) on the drum.

Negative numbers are represented in the LGP-30 as two's complements. We shall discuss here two methods that we may use to obtain the two's complement of a number. First, if we express the negative number we wish to convert as a positive binary number, then make all the 1's zeros and all the zeros 1's and finally add a 1 in the least significant bit, we will obtain the proper complement. For example, suppose we wish to represent -6.75 in binary at q = 3. First we represent +6.75 at q = 3 in binary:

```
±                          sp
0110.110000000000000000000000000

                                  Then reverse all bits:
1001.001111111111111111111111110
                              1   Add a bit to the least significant bit.
1001.010000000000000000000000000
```

The preceding arrangment of bits represents a -6.75 at q = 3.

We shall discuss the second method of representing a negative number through inspection of the first example. If we examine the +6.75 at q = 3 and our final answer, we can readily see that all bits to the left of our least significant bit have been reversed. This is the second rule: represent the number as a positive number at the appropriate q and then reverse all bits to the left of the least significant "1" present. The bits to the right of the least significant bit are unchanged. The least significant bit is unchanged.

Perhaps you have decided by now that you do not like this method of converting negative decimal numbers in this manner. As a matter of fact you undoubtedly do not like the method of converting your positive numbers to binary. Well we did not like it either, we all prefer to be able to write down our decimal number in good old decimal form and tell the machine to convert it for us and place it on the drum at the proper place. The LGP-30 Data Input Subroutines were developed to perform this function for us. As an example, suppose we desire to place the following numbers into the computer.

1. Place +96.40236 in drum location 6234 at q of + 7.
2. Place -.000000597 in drum location 2363 at q of -14.
3. Place +330000. in drum location 2100 at q of +30.

All we need to do is enter these numbers on a load sheet, and punch the data tape from it. These three numbers would be entered on the load sheet in the following manner.

Quan.	P	±	q	Location		± Number		Car. Ret.
	5	+	07	6 2 3 4	'	9 6 4 0 2 3 6	'	
	9	-	14	2 3 6 3	'	-0 0 0 0 5 9 7	'	☒
	0	+	30	2 1 0 0	'	3 3 0 0 0 0	'	

Later we shall learn how to use the Data Input Subroutine in order to have it read in our decimal numbers and place them on the drum.

Now let us consider a word on the drum that is used to store an instruction. The 12 bits at the left are ignored entirely by the computer while it is interpreting the instruction. The one exception to this rule is the transfer control instruction which will be discussed later. The next 4 bits are very important, since the order (command or operation) is represented in binary here. If we consider these four bits in binary, we see that we can represent the integers 1 through 15 by magnetizing various combinations. If no bits are magnetized, we may represent a 16th order. Since our computer has 16 basic orders (or operations) we may assign a number (0 through 15) to each order. The complete table of orders with their equivalent code is given on the following page.

DEC. CODE:	ORDER		BINARY CODE 2^3	2^2	2^1	2^0
0	Z	Stop	0	0	0	0
1	B	Bring	0	0	0	1
2	Y	Store Address	0	0	1	0
3	R	Return Address	0	0	1	1
4	I	Input	0	1	0	0
5	D	Divide	0	1	0	1
6	N	Multiply	0	1	1	0
7	M	Multiply	0	1	1	1
8	P	Print	1	0	0	0
9	E	Extract	1	0	0	1
10	U	Unconditional Transfer	1	0	1	0
11	T	Test	1	0	1	1
12	H	Hold	1	1	0	0
13	C	Clear	1	1	0	1
14	A	Add	1	1	1	0
15	S	Subtract	1	1	1	1

Later we shall explain each instruction in detail but for now let us consider the meaning of other bits present in the word when it is used to represent an instruction.

Just to the right of the 4 order bits we find 2 bits that are also ignored by the computer when it is interpreting an instruction. The 12 bits just to the right of these 2 bits are used to represent the address (operand) of the instruction. This operand refers to one word of the 4096 word memory. As we found earlier, each one of the 4096 words may be identified by giving it a specific track and sector address. Since we have 12 bits to identify this address, it follows that the first 6 bits should be used to represent the track portion of the address and the last 6 bits be used to represent the sector portion. If we analyze the 6 bits of the track, we will find that we may represent in binary any integer in the range 1 to 63 (inclusive) by magnetizing various combinations of bits. Again, if we do not magnetize any of the track bits we may represent track 00. Now we see why the tracks were originally number 00 through 63.

The same logic applies to the six bits used to represent sector. You will note that if we add a 1 to the binary representation of sector 63 we obtain a carry into the track portion and the sector becomes zero. Thus address modification from one track to another is easily obtained. Attention is also called to the fact that the address portion of an instruction is carried at a q of 29. The bit following the address is ignored and the bit to the right of it is our old friend the spacer bit.

In the discussion thus far we have referred to a bit either being magnetized or not being magnetized. This is not actually the case; instead, zeros are magnetized with their north pole in one direction and 1's are magnetized in the opposite direction.

As we now examine the word of instruction, we find that our order must be represented in the binary number system; for example, add is represented as 14. In addition, we find that our address must have its decimal track and sector represented in the binary number system. Here again, we prefer to do all of our programming in the familiar decimal system, so it became necessary to have a program input routine to convert our decimal instructions to binary.

It is obvious that since the computer is a binary computer and consequently does all its computing in binary, it is necessary to have a Data Output Subroutine convert the data from binary to decimal and perform the decimal printing. So to perform any problem we must have a Data Input Subroutine, Program Input Subroutine, and Data Output Subroutine stored in the computer. If we had the Program Input Subroutine in the computer, we could use it to read in the Data Input and Output Programs. So our problem now is to develop a routine that will load the Program Input Subroutine into the computer. We call the program that does this the "bootstrap" program. Later we shall see in detail exactly how this bootstrap loads the Program Input Routine into the computer.

SECTION III
SOME BASIC ORDERS

The LGP-30 is a single address computer. That is, with each order there must be an address. In a three address computer there are 3 addresses with each order. For example, in a three address computer, one would give an add and 3 addresses. The computer would add the contents of the first address to the contents of the second address and place the sum in the third address. In a single address computer this same addition would be programmed with the add order and the one address. Obviously, 2 addresses are implied. In the case of the LGP-30 these 2 addresses both refer to the accumulator. In the case of an add, the computer adds the contents of the accumulator to the contents of the single given address and places the sum in the accumulator. The following presentation explains the basic LGP-30 orders. The R (code 3), P (code 8), I (code 4), N (code 6) and E (code 9) orders are explained later.

In the explanation below m represents any one of the 4096 words of memory.

CODE	ORDER	ADDRESS	INTERPRETATION
1	B	m	Bring -- Place the contents of memory location m in the accumulator after clearing out its previous contents. Memory location m is unaltered.
14	A	m	Add -- Add the contents of memory location m to the contents of the accumulator and place their algebraic sum in the accumulator. Memory location m is unaltered.
15	S	m	Subtract -- Subtract the contents of memory location m from the contents of the accumulator and leave their algebraic difference in the accumulator. Memory location m is unaltered.
7	M	m	Multiply -- Multiply the contents of memory location m by the contents of the accumulator and place the most significant 30 bits of the algebraic product in the accumulator. Memory location m is unaltered.

5	D	m	Divide -- Divide the contents of the accumulator by the contents of memory location m and place the algebraic quotient in the accumulator. Memory location m is unaltered.
12	H	m	Hold --Store the contents of the accumulator in memory location m. The contents of the accumulator are unaltered.
13	C	m	Clear -- Store the contents of the accumulator in memory location m. Then clear the accumulator to zero.
2	Y	m	Store Address -- Store the address portion of the accumulator into the address portion of memory location m. The contents of the accumulator and all other portions of memory location m are unaltered.
10	U	m	Unconditional Transfer -- This instruction transfers control to memory location m. That is, it goes to memory location m and executes the instruction there and proceeds sequentially from there until given another transfer instruction.
11	T	m	Test -- This instruction tests the sign of the accumulator and if the accumulator is negative, control is transferred to memory location m. If the accumulator is positive, the instruction following T m is executed next. (See p.26 - Transfer Control Button). Zero is always "+".
0	Z	m	Stop -- This order is a conditional stop instruction. (See p. 25 - Break Point Buttons).

SECTION IV
SCALING

If we are adding two numbers, the decimal points must first be lined up. In performing an addition on the LGP-30, the binary points must also be lined up. That is to say that the "q" of both numbers must be the same. The "q" of the sum will be equal to this same q.

Similarly, the same principle holds for subtraction.

In performing a multiplication, the number of decimal places in the multiplier must be added to the number of decimal places in the multiplicand, to give the number of decimal places in the product. On the LGP-30, the binary points of the multiplier and multiplicand are added (i.e. "q_1", + "q_2" = "q_3") to give the binary point location of the product.

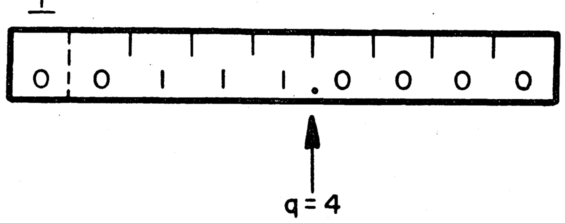
In division, the binary point of the divisor is subtracted from the binary point of the dividend (i.e. $q_1 - q_2 = q_3$) to give the binary point of the quotient.

Occasionally customers have an application in which all input data is extremely large or small. For instance, the input might all be in multiples of one thousand. In this case, the input could be "scaled" down by 1000. This type of "scaling" is well known by everyone familiar with the operation of a desk calculator.

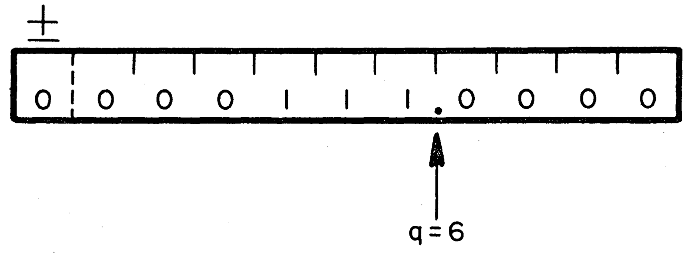
In this section, scaling within the computer will be considered. We saw above that before an addition or subtraction could be performed, the numbers involved had to be "scaled" to the same binary point location. That is to say, one number may have to be shifted to the right or left before performing the addition or subtraction.

First, we shall consider shifting a number to the right. Suppose we have the number 7 at a "q" of 4 and we wish to shift this number to a "q" of 6 in order to add it to another number already at a "q" of 6.

The number 7 at a "q" of 4 may be represented as:



If we multiply this 7 at "q" of 4 by a 1 at a "q" of 2, we obtain 7 at a "q" of 6 which may be represented as:



You will note that performing a multiplication by 1 at a "q" of 2 shifted the entire contents of the accumulator right by 2 places.

From this simple example it is apparent that we may shift the contents of the accumulator to the right "x" places by performing a multiplication by a 1 at a "q" of "x".

There are two ways to shift the accumulator left "x" places. First, we shall consider the method employing the use of a divide instruction. Let us reverse the previous example i.e. shift a 7 at a "q" of 6 to a 7 at "q" of 4.

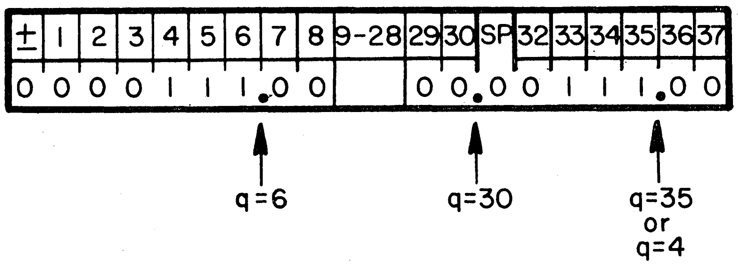
To accomplish this, we divide the 7 at a "q" of 6 by a 1 at a "q" of 2 and obtain a 7 at a "q" of 4.

It might be pointed out here that if we were to divide a 7 at a "q" of 6 by a 1 at "q" of 4 we might expect to obtain 7 at a "q" of 2. Such a thing is impossible since the quotient (7) may not be carried at a "q" of 2. If this division were attempted on the LGP-30, the computer would stop. This condition is referred to by programmers as a "divide check". A "divide check" will result if the quotient will not "fit" at the "q" resulting from the division. Another way to analyze the possibility of a divide check, is to analyze the dividend and divisor before dividing. If the divisor is greater than the dividend by at least one unit in the 30th binary position (i.e. the least significant binary digit), then a divide check will not occur.

The second method of shifting left "x" places involves the N multiply instruction. The N multiply instruction places the least significant 31 bits of the product in the accumulator. Attention is called to the fact that the high order bit appearing in the accumulator does not indicate the sign of the product. It actually represents the bit that would have appeared in the spacer bit position on the M multiply instruction if it were possible to display this bit then.

Thus far we have been speaking of an LGP-30 accumulator as if it were only 32 bits somewhere. This is not truly the case, but from the programmer's standpoint he may consider it this way if he desires. Actually the accumulator is a track located on the drum. The accumulator track consists of 64 separate 32 bit accumulators just as any one of the tracks 00 through 63 consists of 64 separate 32 bit sectors. The main difference is that the accumulator track is a "recirculating" track, i.e. whatever arrangement of bits are found in one 32 bit accumulator will be found in all of the other 32 bit accumulators. This "recirculating" feature enables the computer to obtain access to the accumulator value at 64 distinct locations during each drum revolution. So in general there are 64 separate accumulators on the accumulator track. The exception to this rule is that following an "M" or "N" multiply there are only 32 separate accumulators on the accumulator track. Then, each accumulator contains 64 bits consisting of 1 sign bit followed by 30 bits of accumulator value, a spacer bit, 31 additional bits of accumulator value and another spacer bit. Following an M multiply, the computer considers only the <u>first</u> half of the 64 bit accumulator, viz., the sign bit, 30 bits of accumulator value and a spacer bit. Following an N multiply, the computer considers only the <u>last</u> half of the 64 bit accumulator, viz., the 31 bits of actual accumulator value and the spacer bit.

The following illustration will help to clarify the effect that an "M" or "N" multiply has on the value of "q".



Suppose now that we have 7 at a "q" of 6 and desire to shift it left 2 places and obtain 7 at a "q" of 4, as shown in the illustration above. If we N multiply 7 at a "q" of 6 by 1 at a "q" of 29, we obtain 7 at a "q" of 35. You will notice that a "q" of 35 is equivalent to a "q" of 4 when the "q" in the latter case is referenced to the new accumulator.

To shift a number "n", left "x" places, N multiply "n" by 1 at a "q" of (31 - "x"). Care must be taken to be sure that the number "n" will fit at the new "q".

The q of a number following an N multiply is: $q3 = q1 + q2 - 3$.

SECTION V
PROGRAMMING A SIMPLE PROBLEM

Example No. 1. Given the following equation, write a program that would store Q at a "q" of 7 in memory location 1505.

$$Q = \frac{K\ (F + G - J)}{W}$$

We shall assume K, F, G, J, and W represent various quantities of some hypothetical problem that are stored as indicated below:

F in 1500 at q = 7
G in 1501 at q = 7
J in 1502 at q = 7
K in 1503 at q = 7
W in 1504 at q = 7

The program, consisting of the following instructions, may be placed on any track, (Example shows track 10 sector 02).

Loc.	Order	Address	Notes
1002	B	1500	F at 7 in acc.
1003	A	1501	(F + G) at 7 in acc.
1004	S	1502	(F + G - J) at 7 in acc.
1005	M	1503	K (F + G - J) at 14 in acc.
1006	D	1504	$K\ \frac{(F + G - J)}{W} = Q$ at 7 in acc.
1007	H	1505	Q at 7 in 1505 and Acc.
1008	Z	0000	Halt

Let us now consider how we would go about getting the data F, G, J, K, and W into the computer so our program could use them. We mentioned before that we have a set of instructions that will convert decimal data to binary numbers and store them anywhere in the computer at the desired q. This set of instructions is a program in the Royal McBee program library, and is called Data Input 1 (prog. J2-11.0 R). This program occupies 3 tracks and can be placed anywhere on the drum. Let us assume that another program called a "program input routine" has placed Data Input 1 on tracks 53, 54, and 55. What we would like to do is have Data Input 1 read in the data for our program just before we begin executing our program at location 1002. We can accomplish this by preceding our program with 2 instructions that will cause us to "transfer control" from our program to Data Input 1, and then return to our program to continue operating after Data Input 1 has finished its function. Before writing these two instructions we must know the "entry point" and "exit point" of Data Input 1. The entry point of a program is the location to which control is transferred to allow the program to start operating. The exit point of a program is the location from which control is transferred when the program has finished operating. The entry point for Data Input 1 is location 5300 and the exit point is 5308.

Now we are ready to examine the R (Return Address) instruction. The purpose of an R instruction is to place an address in the address portion of a word in memory. Usually it is used to place a return address in the address portion of a word in memory whose order portion already contains a U (Unconditional Transfer) instruction.

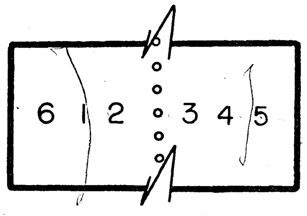
Let us suppose we write the instruction R 5308 in our program at location 1000. When the computer executes an R instruction, it adds 2 to the location of the instruction it is executing (in our case, 2 + 1000=1002) and places this sum in the address portion of the word in memory specified by the address portion of the R 5308 instruction. We have now effectively set the exit point of Data Input 1. When Data Input 1 finishes operating, it will transfer to its exit point at 5308 where it will encounter a U order and the return address 1002. (Address 1002 will be placed there when R 5308 in our program is executed). You will notice that the R order, when executed, does not alter the accumulator and does not perform a transfer. It only sets up an address. Having done this for us, the computer then executes the instructions at the next memory location, 1001. In memory location 1001 we will execute the instruction U 5300. Our completed program now becomes:

Loc.	Order	Address
1000	R	5308
1001	U	5300
1002	B	1500
1003	A	1501
1004	S	1502
1005	M	1503
1006	D	1504
1007	H	1505
1008	Z	0000

SECTION VI
TAPE CODES

In later sections we shall consider how information enters the computer and how the oscilloscope and console buttons may be used to perform various functions. In order to understand how information enters the computer we must first learn the form that this information is in. This section explains how each alphabetic, numeric or control character on the keyboard has its own unique 6 bit code and how the computer accepts 4 of these 6 bits.

The punched paper tape contains 6 channels and the channels are numbered as indicated below.



Machines with serial number 12 or less read 4 of the 6 channels into the computer. Namely, channels 1 2 3 and 4 read into the computer. For machines with serial numbers greater than 12 the same is normally true; however, on these machines if the "6 bit Input" button is depressed all 6 bits of the character represented on tape will read in. This entire write up is based on using the LGP-30 with the 4 bit input feature. As you will see later, this is normally the way the computer is used.

The following table denotes the tape codes associated with the typewriter keys 0 through 9. A "1" indicates a punched hole in the tape.

Key Code	Tape Code	Binary Code
	16 8 4 2 1 0 612345	
0	1	0
1	11	1
2	1 1	2
3	111	3
4	1 1	4
5	1 11	5
6	11 1	6
7	1111	7
8	1 1	8
9	1 11	9

Upon examining the tape codes, one will notice that channels 1, 2, 3, and 4 give the binary representation of the decimal number that has been punched. We discovered earlier in our discussion of the 4 order bits that it is possible to represent the binary numbers 0 through 15 in four consecutive bits. Upon inspection of the table, one immediately notices that some combinations are missing. In particular, the binary 10, 11, 12, 13, 14 and 15. Since these keys are not on the typewriter unit and it is necessary (at times) to be able to place these codes on tape, the following correspondence table was adapted.

Memorize this:

F = 10
G = 11
J = 12
K = 13
Q = 14
W = 15

Key Code	Tape Code						Binary Code
	6	1	2	3	4	5	
F		1		1		1	10
G		1		1	1	1	11
J		1	1			1	12
K		1	1		1	1	13
Q		1	1	1		1	14
W		1	1	1	1	1	15

You will note that these numbers (if we consider F, G, J, K, Q, W to represent binary numbers) all contain a channel 5 punch and never contain a channel 6 punch.

A table of tape codes for the 16 orders are given below. Attention is called to the fact that all order codes contain a punch in channel 6 and never a punch in channel 5. You will also notice that the binary representation of these codes (as given by channels 123 and 4) corresponds to the code assigned to each order in the discussion of "Basic Orders".

Key Code	Tape Code						Binary Code
	6	1	2	3	4	5	
Z	1						0
B	1				1		1
Y	1			1			2
R	1			1	1		3
I	1		1				4
D	1		1		1		5
N	1		1	1			6
M	1		1	1	1		7
P	1	1					8
E	1	1			1		9
U	1	1		1			10
T	1	1		1	1		11
H	1	1	1				12
C	1	1	1		1		13
A	1	1	1	1			14
S	1	1	1	1	1		15

The reader is referred to the complete keyboard code sheet for control codes and the balance of the keyboard codes. Attention is called to the fact that these codes either contain no punching in channels 5 and 6 or punching in both channel 5 and 6.

The reader should note that on the sheet giving the tabulation of tape code, that the channels are represented in the order 123456 instead of 612345 as they actually appear on tape.

The LGP-30 typewriter keyboard contains one key that is not found on conventional typewriter keyboards. This is the "conditional stop" key. (cond. stop). You will find that you will possibly use this one key more than any other one key on the keyboard. Its function is to place on tape a code that will stop the tape from reading.

SECTION VII
TAPE READ IN

The tape may be started in motion by programming two instructions. These two instructions are P0000 and I0000. Once the tape is set into motion, channels 1, 2, 3, and 4 enter the right hand side of the accumulator and move the previous contents of the accumulator to the left by four bits. This process continues until a conditional stop code is sensed on the tape. At this time the tape stops (the conditional stop code does not enter the accumulator) and the computer executes the instruction following the I0000 instruction.

We stated earlier that the accumulator and each one of the 4096 words of memory contained 32 bits and that the bit at the extreme right was a spacer bit that always contained zero. Now we find the only exception to this rule. During input the right hand bit of the accumulator may accept either a "1" or a zero. If this were not so, the right hand bit of each group of four would contain a zero. (e.g. 0001 would enter as 0000). As soon as the contents of the accumulator are stored the spacer bit becomes as "0". So if the bit in the right hand position of the accumulator is to be preserved, the contents of the accumulator must be shifted to the left before storing.

The accumulator has a 32 bit capacity and each character read in requires 4 bits. Under these circumstances, eight is the maximum number of characters that may be read in without the presence of a stop code to prevent the loss of some significant character.

It might also be mentioned here that if a clear order is given before the P0000 and I0000 orders, leading zeros need not be punched on the input tape.

Since the accumulator and each word of memory contains 32 bits, it would be very inconvenient to say the accumulator has a value of 10011101100010110111101011101100. At once it becomes apparent that some form of "shorthand" notation is needed. In view of the fact that the LGP-30 reads in 4 bits at a time, we find it very convenient to break the bits of a word into groups of 4. This form of notation is referred to as hexadecimal. The above arrangement of bits may be represented in hexadecimal notation as 9K8G7FQJ.

Before leaving the subject of tape read in it should be stated that all tape codes listed on the "keyboard code" sheet will be read into the computer by the tape reader except the control codes listed on the following page.

	Control Code	Tape Code 1 2 3 4 5 6
1.	Lower Case	0 0 0 1 0 0
2.	Upper Case	0 0 1 0 0 0
3.	Color Shift	0 0 1 1 0 0
4.	Carriage Return	0 1 0 0 0 0
5.	Back Space	0 1 0 1 0 0
6.	Conditional Stop	1 0 0 0 0 0
7.	Start Read	0 0 0 0 0 0
8.	Delete	1 1 1 1 1 1

The tab code (011000) is different from any other code in that it will not enter its code into the accumulator but will shift the contents of the accumulator left by four or six bits. The four or six bit shift will occur when using the four and six bit input feature respectively.

SECTION VIII
CONVERSION OF DECIMAL INSTRUCTIONS AND DATA TO HEXADECIMAL

As an example of reading in an instruction let us say we wish to read in the instruction B 4627. We said before that we were going to code in decimal and have the program input routine convert the entire instruction to binary and arrange it in the proper form for us. The method to be described below is not the method employed by the program input routine. The following method lends itself very well to mental manipulations where the method employed in the program input routine blends itself very nicely to machine manipulation. Later on in the course we shall describe in detail the latter method, but for now let us concern ourself with mentally performing this conversion.

If we were to read in the instruction B 4627, it would enter the accumulator in the following manner:

(a)	0 0 0 1	0 1 0 0	0 1 1 0	0 0 1 0	0 1 1 1
	(bring code 1)				Spacer bit

This form is known as Binary Coded Decimal since each decimal number is represented in binary. We know from our study of the structure of an instruction word that B 4627 must be represented in the computer in the following fashion.

(b)	Order		Track	Sector	
	0 0 0 1	0 0	1 0 1 1 1 0	0 1 1 0 1 1	0 0
	Bring		Binary	Binary	
	Code 1		46	27	

If we represented this same desired form in groups of four bits, it appears as:

(c)	0 0 0 1	0 0 1 0	1 1 1 0	0 1 1 0	1 1 0 0
	Bring	2	14=Q	6	12=J

Consequently, the computer must convert B 4627 in binary coded decimal to B 2Q6J. The B 2Q6J is known as hexadecimal. For an explanation of hexadecimal form the reader is referred to the section on "Tape Read In." You will note that the order portion needs no attention. This is always the case.

If we consider the track portion in illustration (b) above, we note that the last 4 bits of track consist of one group of 4 bits in illustration (c). This means that we may represent decimal tracks 0 through 15 in these 4 bits. If we desire to represent track 16, we must represent it as a bit just to the left of this group of 4 bits. Consequently, the number of bits required to the left of this group of four bits is obtained by dividing the decimal track by 16. In our case we would divide 46/16 and obtain 2 with remainder of 14. The 2 we represent to the left of the group of 4 bits and the 14 (equivalent to Q) we represent within the 4 bit group.

Upon analyzing the Sector portion of illustration (b), we find that the last 2 bits of sector fall in the last group of 4 bits in illustration (c). This means that we may represent decimal sectors 0 through 3 in the 2 high order bits of the last group of (c). To represent sector 4, we must represent it as a bit just to the left of the last 4 bits in illustration (c). Consequently the number of bits required to the left of this group of 4 bits is obtained by dividing the decimal sector by 4. In our case we would divide 27/4 and obtain 6 with a remainder of 3. The 6 we represent in the bits to the left of the last group of 4 bits in illustration (c). In our example it is easy to see that we desire to represent the remainder in the 2 high order bits of the last group of 4 bits. Now if we consider only the last 4 bits (1100), we see that the bit adjacent to the zero really represents a binary 4 and the bit to the left of this bit represents a binary 8. So instead of representing our remainder as a binary 3 in the last 4 bits we are representing it as a binary 12 or as 4 times its value. Consequently the remainder obtained by dividing 27/4 must be multiplied by 4 to permit us to enter it into the last 4 bits of the accumulator from the tape or keyboard.

The above conversions may be summarized in the following manner.

Decimal Track to hexadecimal track conversion

Divide the decimal track by 16. The integer represents the high order track hexadecimal integer and the remainder represents the low order hexadecimal character. If the low order hexadecimal character is 10, 11, 12, 13, 14, or 15 use the equivalent F, G, J, K, Q or W form.

Decimal sector to hexadecimal sector conversion

Divide the decimal sector by 4. The integer represents the high order sector hexadecimal character. The remainder is multiplied by 4 and this product becomes the low order hexadecimal character. If the high or low hexadecimal character is 10, 11, 12, 13, 14, or 15 use the equivalent F, G, J, K, Q or W form.

Consequently, if we preferred to go through this mental gymnastics for each instruction we would not need an input routine. We find the program input routine much faster and for most of us, far more reliable. This presentation is given here merely as an exercise to help the student grasp the problem of conversion and should not be interpreted as part of the programming technique. It may, however, be used if a situation should warrant it.

The method just described is a method which is very useful when converting decimal track and sector to hexadecimal. When it becomes necessary for a programmer to express decimal data in hexadecimal, the conversion is not always a mental one.

In previous sections of this write-up (page 3) simple decimal data was converted to binary by inspection. For example +6.75 at a "q" = 3 was quickly converted to binary by sketching an accumulator. Normally the data to be converted does not lend itself to this method.

A general method for converting a decimal data number "N" to a hexadecimal number with a binary point location of "q" is given below. This method expresses the converted number in hexadecimal rather than binary. Use of a table of powers of 2 and a desk calculator are required.

If the number to be converted is N and its binary point location is to be "q", the high order hexadecimal character (I_1) is found by the following formula.

$$N\,(2^{3} - q) = I_1\ F_1 \qquad \begin{array}{l}(I_1 \text{ is integer part of product})\\(F_1 \text{ is fractional part of product})\end{array}$$

The second hexadecimal character (I_2) is found by:

$$F_1\ (16) = I_2\ F_2$$

The third hexadecimal character (I_3) is found by:

$$F_2\ (16) = I_3\ F_3$$

The five remaining hexadecimal characters may be found by the equation:

$$F_{(i\ -\ 1)}\ (16) = I_i\ F_i$$

As an example of this method, let us convert the value (3.1415927) to hexadecimal at a "q" of 3.

$$\begin{array}{lll}
3.1415927\ (2^{3-3}) & = 3.1415927 & I_1 = 3\\
.1415927\ (16) & = 2.2654832 & I_2 = 2\\
.2654832\ (16) & = 4.2477312 & I_3 = 4\\
.2477312\ (16) & = 3.9636992 & I_4 = 3\\
.9636992\ (16) & =15.4191872 & I_5 = W\\
.4191872\ (16) & = 6.7069952 & I_6 = 6\\
.7069952\ (16) & =11.3119232 & I_7 = G\\
.3119232\ (16) & = 4.9907712 & I_8 = 4
\end{array}$$

Combining the 8 hexadecimal characters and rounding the last one (due to spacer bit), the 32 bit word appears as:

3243W664

Consulting the powers of 2 table enables us to check this hexadecimal representation against the number we started with.

SECTION IX
OSCILLOSCOPE

The oscilloscope, hereafter referred to as scope, along with the console buttons and typewriter keyboard, provide the operator a means of checking out ("debugging") a program. In this section we shall discuss the role played by only the scope. In the section on console buttons, the reader will acquire the concept of how the three components are necessary to "debug" a program.

The scope furnishes the operator with a visual representation of 3 registers. Each register is displayed in a separate window on the face of the scope.

The top window displays the contents of the counter register. The counter register always contains the location (binary track and sector) of the next instruction that the computer plans on executing. It tells the operator where the machine is "headed."

The center window displays the contents of the instruction register. When the computer stops, the instruction register displays (in binary) the last instruction executed by the computer. There is an exception to this general rule; namely, after the execution of a divide or multiply instruction, the multiply or divide instruction does not appear in the instruction register. In the section on the "console buttons", the reader will learn how to use this instruction register to "interrogate" a set of words (data or instructions) without executing any instructions. In the section on the "console buttons" the reader will also learn how the operator may execute an instruction that is in the instruction register.

The lower window displays the contents of the accumulator (in binary). We may use the window to view the contents of the accumulator after a series of operations or after manually entering information into the accumulator.

SECTION X
CONSOLE BUTTONS

In checking out programs the scope enables the operator to determine where the computer has stopped, the last instruction executed, and the contents of the accumulator. Through the use of the console buttons, the typewriter keyboard and the scope, the operator may interrogate a series of instructions, interrogate a table of data, enter a new instruction or data word into the computer, manually transfer control to another portion of memory or perform one operation of his program at a time.

POWER ON AND POWER OFF:

Let us now turn our attention to the console buttons and lights. First, there is a very important rule to always remember. This rule is the alpha and omega (the beginning and the end) rule which may be stated as "always to be certain that the manual input button (interrogate button on early machines) is depressed before depressing the 'Power On' or the 'Power Off' button." Violation of this rule "MAY" result in bits on the drum being altered. The worst thing that can happen is that you may have to load your routine into the computer again. Since the LGP-30 will retain programs or data on the drum for an indefinite period of time, it is a waste of time to reload these routines due to carelessness on the part of the operator.

So, after checking to insure that the "Manual Input" button is depressed, the operator may then depress the "Power On" button.

STAND BY:

After the Power On button is depressed, the red "Stand By" button will light up. For the following 50 seconds half voltage is applied to the filaments of all tubes and the drum is brought up to speed.

STAND BY TO OPERATE:

Following this 50 second warm-up period (providing the "operate" button has been previously depressed), the computer will automatically turn out the "Stand By" light and light the yellow "Stand By to Operate" light. At this time full voltage is applied to the filaments and all computer voltages are regulated to their proper value. This "Stand By to Operate" stage also requires 50 seconds.

OPERATE:

At the completion of this period the "Stand by to Operate" light goes out and the green "Operate" button will light. This is an indication to the operator that the computer is ready for use. Before leaving the subject of computer warm up we might also state that should the operator desire to place the computer in "Stand By" he may do so by depressing the "Stand By" button. This will reduce the filament voltages to half voltage. Later when the computer is needed, the green operate button may be depressed and the computer will again cycle through the

the 50 second "Stand By to Operate" stage before lighting the green "operate" button to indicate it is ready for use. You will notice that "Stand by to Operate" may not be depressed since it is not a push button.

MANUAL INPUT BUTTON:

Let us now consider the functions that may be performed while the "Manual Input" (interrogate on early machines) button is depressed. First, let us see why the word "interrogate" was originally given to this button. With this button depressed we may easily "interrogate" a series of words within the computer. If the start button is depressed while the "Manual Input" button is depressed, the contents of the track and sector represented in the counter (top window) will be displayed in the instruction register (middle window) and the contents of the counter register will be increased by one. You will notice that no instructions will actually be executed by the computer. Multiply and divide instructions will appear in the instruction register if they are present in the program being interrogated. Data will also appear in the instruction register if it is contained in the memory locations being interrogated.

FILL INSTRUCTION BUTTON:

Another function that may be performed while the "Manual Input" button is depressed is that of entering information into the computer from the keyboard. As a key of the keyboard is depressed, its 4 bit code is entered into the right hand side of the accumulator and the original contents of the accumulator shifted left by 4 bits. In this way the B 2Q6J could be typed into the accumulator to represent a B 4627 instruction. If we desire to execute this instruction, we must place this arrangement of bits into the instruction register. This transfer of information from the accumulator to instruction register may be accomplished by depressing the fill instruction button. After this button is depressed the contents of the accumulator and the contents of the instruction register are identical.

ONE OPERATION BUTTON -- EXECUTE INSTRUCTION BUTTON:

Having the instruction to be executed in the instruction register, one may depress the "one operation" button and then depress the "execute instruction" button. If B 2Q6J is in the Instruction register, the contents of 2Q6J (4627) will be brought to the accumulator. Depressing the "one operation" button before the "execute instruction" button caused the machine to halt after executing the one instruction.

NORMAL --ONE OPERATION -- MANUAL INPUT BUTTON:

Attention is called to the fact that the "normal", "one operation" and "manual input" buttons are mechanically interlocked. It is impossible to go from "normal" to "manual input" and conversely from "manual input" to "normal." It is necessary to first depress the "one operation" button in changing from one of these modes to the other. This interlocking feature was incorporated in the LGP-30 to prevent any operator from leaving the normal mode of operation before the operation being executed had been completely executed.

The readers' attention is also called to the lights under the various buttons. The lights under the "normal", "one operation" and "manual input" buttons permit the operator to tell at a glance which mode of operation the computer is in. While the computer is in the "manual input" mode of operation the light under the "fill instruction" button is lit. While in the "one operation" mode lights under the "start", "clear counter" and "execute instruction" buttons are lit. This feature assists the operator in remembering which auxiliary buttons are active when in the various modes of operation.

CLEAR COUNTER BUTTON:

When the "clear counter" button is depressed, the counter is reset to track zero sector zero. This feature allows us to readily get to memory location 0000. Later we will discuss the program input routine which we suggest you place in track 00, 01, 02, in order to be able to quickly get to its first instruction in memory location 0000.

STOP LIGHT:

The red "stop" (Blocked State on earlier machines) light will be on whenever the computer is not computing.

COMPUTE LIGHT:

The green "Compute" light is on while the machine is computing.

BREAK POINT BUTTONS:

Machines with serial numbers 12 or less contain five break point switches. Machines manufactured after that contain four break point switches.

The break point switches are used in conjunction with the Z (stop) order previously discussed. Mention was made at that time that the Z order was a conditional stop. Actually the stop depends on the "condition" (up or down) of the break point switch setting called for in the address portion of the Z order. The break points are numbered 32, 16, 8, 4, and 2 (2 is present only on early machines). When the stop instruction is programmed, the programmer must supply the Z order with a track and sector address in the same manner he would program any other instruction. If the programmer programs a "stop" instruction with a track address corresponding to one of the break point switches, the computer will not execute a stop until it has checked the condition of the corresponding break point switch. If the corresponding break point switch has previously been depressed, the computer will disregard the programmed stop instruction and execute the instruction following the programmed stop. If the computer finds the corresponding break point switch has not been depressed, it will execute the instruction as a stop instruction.

It might be mentioned that a Z 0000 is always a stop instruction, regardless of the break point switch settings.

The break point switches are mechanically engineered so that one depression will latch it in the down position and the following depression releases the latch. Lights under the break point switch buttons are lit while the switch is in the down or latched position.

The programmer will find it convenient to program Z instructions with either a zero track address or a track address corresponding to one of the break points. For instance, a Z 2000 is interpreted by the computer as a stop order if either break point switch 16 or 4 has not been previously depressed.

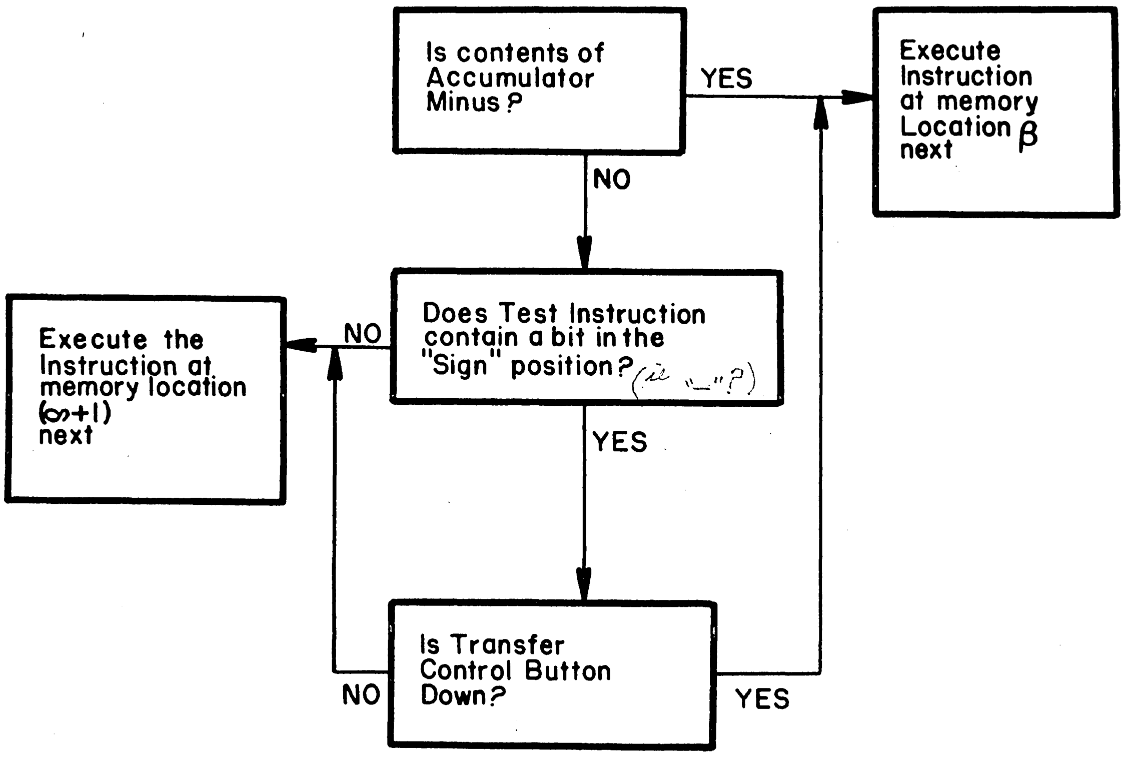
TRANSFER CONTROL BUTTON:

The transfer control button, like the break point switches, furnishes the computer additional information required for the execution of an instruction. In the case of the break point switches, we found that the Z (halt) order was the order associated with the break point switches. In the case of the transfer control button, a variation of the test (T) order is associated with the transfer control button.

You may recall that we stated earlier that the computer ignored the 12 leading bits of a word when executing an instruction. It was also stated that there was one exception to this general rule. Now we are ready to consider this one exception.

Whenever the LGP-30 executes a test (T) instruction, it tests the contents of the accumulator for being negative. If the accumulator is negative, the computer next executes the instruction whose address is given in the address portion of the test instruction. In the event that the accumulator is positive, the computer next checks the sign bit position of the test instruction. If the sign bit position of the instruction does not contain a bit (i.e. test instruction is plus), the computer next executes the instruction following the test instruction. When the accumulator is plus and the sign position of the test instruction contains a bit (i.e. test instruction is minus), the computer next examines the "condition" of the "transfer control" button. When the "transfer control" button is down, the computer will next execute the instruction whose address is given in the address portion of the test instruction. In the event that the "transfer control" button is up, the computer will next execute the instruction following the test instruction.

The above paragraph has been condensed to the block diagram form, (next page) to enable the student to better grasp the flexibility of this test instruction. If at memory location α (where α represents any memory location), the instruction "-Tβ" is given (where β represents some other memory address) the computer will follow the logic outlined below in executing the "-Tβ" instruction.


Is contents of Accumulator Minus?
YES
Execute Instruction at memory Location β next
NO
Does Test Instruction contain a bit in the "Sign" position? (ie "—"?)
NO
Execute the Instruction at memory location (∞+1) next
YES
Is Transfer Control Button Down?
NO
YES

The beginner may not desire the added flexibility that this instruction affords until he gains a little programming experience but he should be aware of the flexibility he has at his disposal.

SIX BIT INPUT BUTTON:

All machines having a serial number greater than 12 contain a "Six Bit Input Button" in place of Break Point Switch No. 2. When this "Six Bit Input Button" is depressed, all six tape channels will enter the accumulator. The reader is referred to Section VII for a more detailed explanation of the tape codes that actually enter the accumulator.

When the "Six Bit Input Button" is up, only channels 1, 2, 3, and 4 enter the accumulator.

NORMAL BUTTON:

After the normal button is depressed, the computer is prepared to execute instructions at high speed. If the "start" button is depressed, the computer will begin the execution of instructions with the instruction contained in the word whose address is displayed in the counter register.

START BUTTON:

Three possible interpretations are associated with depressions of the "start" button. They are summarized below as a function of the computer's mode of operation at the time of the "start" button depression.

Manual Input:

A depression of the start button will result in displaying the contents of the word, whose address is displayed in the counter register, in the instruction register. The counter register will be increased once for each depression of the start button. No instruction will be executed by depressing the start button in this mode of operation. For this reason the accumulator value will be unaltered.

One Operation:

A depression of the start button will execute the instruction in the word whose address is displayed in the counter register and stop. The contents of the counter register will be increased by one unless the instruction that was actually executed was a transfer instruction. In this case, the counter register will contain the address of the transfer instruction. The instruction register will contain the instruction just executed except for multiply and divide instructions. This is due to the fact that in the design of the LGP-30 the engineers utilized time sharing circuitry when economical to do so. During the execution of a multiply and divide instruction, it was found economical to utilize the instruction register for another function. After the execution of an instruction, the accumulator register displays (as always) the contents of the accumulator. Whether the accumulator content has been changed or not as a result of executing the instruction is a function of the instruction executed.

Normal:

A depression of the start button will start the computer executing instructions at high speed. The first instruction to be executed will be located at the address displayed in the counter register.

The following example is given to review the console button functions. This review is accomplished by executing an unconditional transfer (U) instruction.

To manually transfer to 4627, the operator must depress the "manual input" button, type in a U 2Q6J, depress the "fill instruction" button, depress the "one operation" and finally depress the "execute instruction" button. The counter window will then contain 2Q6J (4627) indicating that the machine is ready to transfer control to 4627 as soon as the "start" button is depressed. The "start" button may be depressed while in the "one operation" mode and only the one instruction contained in 4627 will be executed. If the operator so desires, he may depress the "normal" button and then the "start" button, in which case the computer will execute instructions, beginning with the instruction contained in 4627.

The best way to stop the computer while it is operating at high speed ("normal" button depressed) is to depress the "one operation" button. If the operator does this, he may later depress the "normal" button and then the "start" button and the computer will continue with its sequence of operations.

As a final example, let us manually enter the instructions B4953 in location 5614. The following sequence of operations is required:

1. Depress "manual input" button.
2. Type C3838 (C5614) on keyboard.
3. Depress "fill instruction" button.
4. Type 000B31K4 (B4953) on keyboard.
5. Depress "one operation" button.
6. Depress "execute instruction" button.

We will find later that this instruction (B4953) may be placed in 5614 by entering everything in decimal form.

SECTION XI
TYPEWRITER CONTROLS

Power - On - Off:

The typewriter unit contains its own main line switch. This switch is a toggle type switch clearly marked "power". The "on" and "off" positions are appropriately indicated.

Connect Switch:

In our discussion on tape read in, we stated that a "conditional stop code" on the tape would stop tape reading. In addition to stopping the tape motion, the "conditional stop" code provides a "start signal". This "start signal" is capable of starting the computer at the instruction following the I0000 instruction.

Many times it is advantageous for us to prevent this "start signal" from reaching the computer. In short, we would like to "disconnect" the wire that carries this "start" signal back to the computer. This may effectively be accomplished by placing the "connect" switch in the "off" position. Conversely the "start signal" reaches the computer when the "connect switch" is in the "on" position.

The other typewriter unit controls are mounted in two banks above and to the rear of the typewriter keyboard. Each bank contains 4 labeled levers. The function of each lever is presented below.

Start Read Lever:

We found that the programmed instruction P0000 and I0000 would set the tape reader into motion. When the typewriter unit is "disconnected" (connect switch to off), it very often becomes necessary to set the tape reader in motion manually. This may be accomplished by depressing the "start read" lever.

Stop Read Lever:

Whenever the tape reader is in motion, it may be stopped by depressing the "stop read" lever.

Punch On Lever:

The LGP-30 typewriter unit is equipped with two tape unit heads. One head is capable of reading the tape and the other is capable of punching a tape. The punch head is capable of feeding and punching holes only while the "punch on" lever is in the down position. Depressing the "punch on" lever prepares the punch unit for operation.

Tape Feed Lever:

When the "punch on" lever is in the down position, tape will feed through the punch head as long as the "tape feed" lever is held in a downward position. This tape feed lever will spring up as soon as it is released. The "tape feed" lever is very useful in placing blank tape on the front or rear of a tape that is being prepared in the punch unit. While the "tape feed" lever is held down, sprocket holes are cut in the tape but no other holes are cut. Depressing the "tape feed" lever does not cause the "read unit" to move.

<u>Code Delete Lever:</u>

After the "punch on" lever has been depressed, every key depression will result in that key's code being punched on tape. If the operator should depress the wrong key and detect his error immediately, he may manually roll the punch tape back one space and depress the "code delete" lever. Depressing the "code delete" lever punches holes in channels 612345. Later when this code is read by the read unit it will not enter the computer or be reproduced on to another tape.

<u>Conditional Stop Lever:</u>

Since the LGP-30 has a punch head and read head it often becomes convenient for an operator to reproduce a tape. To accomplish this, the operator must place the tape to be reproduced in the read head, Next, the "punch on" lever should be depressed and the "tape feed" lever held down to obtain a leader on the front of the new tape. If the "start read" lever is depressed, the tape read unit will start in motion and as characters appear under the read head the corresponding characters will be punched at the punching station. When the read head detects blank tape, the read head remains in motion but the punch unit stops feeding tape. Delete codes read by the read head are not punched by the punch head.

Once the tape read unit is set in motion by a depression of the "start read" lever, it will remain in motion until a conditional stop code is read. In order to start the tape reader again (when reproducing) another depression of the "start read" is required. When reproducing a long tape, depressing the "start read" after every Conditional Stop code would become quite a chore. It is obvious that the operator would like to avoid stopping after each conditional stop code when reproducing tapes. By depressing the "Conditional Stop" lever these stops will be ignored. Raising the "conditional stop" lever will cause the tape to stop after the next conditional stop code is read at the read head.

<u>Manual Input Lever:</u>

We found earlier that a programmed P0000 followed by an I0000 would call for information to be read into the computer. Normally this information will be on tape and be ready to pass under the tape read head. Occasionally though an operator may desire to enter a change or additional information into the computer. By depressing the "manual input" lever the operator will force the computer to stop following the execution of the P0000, I0000 instruction. This type of stop is indicated by a light on the typewriter unit being lit. At this time the operator may type information which will be entered into the accumulator.

Start Compute Lever:

The "start compute" lever is especially useful after the computer is brought to a stop by depressing the "manual input" lever and then having the machine execute the instructions P0000, I0000. Under this condition the light on the typewriter unit will glow and the computer will stop. Manual information may be entered via the typewriter keyboard and then a depression of the "start compute" lever will turn out the typewriter unit light and start computing within the computer.

Paper Guide:

The "paper guide" is located just to the rear of the platen. This guide should be adjusted horizontally so that it just touches the left edge of the paper form. This "paper guide" is mounted on a metal shield known as the "paper table".

Tab Rack:

If the "paper guide" is used to rotate the "paper table" to the rear, the "tab rack" is visible. The "tab rack" has the numbers 8 through 80 (in increments of 4) inscribed on it.

Margin Rack:

The "margin rack" is located just in front of the "tab rack". The "margin rack" has the numbers 0 through 44 (in increments of 4) inscribed on it.

Front Paper Scale:

The "front paper scale" is printed on the metal shield just to the front of the platen. By viewing the "front paper scale" through the aperture at the printing position, one may determine the exact position of the carriage. Note the hairline indicator in the aperture window numerically defining the carriage position.

Tab Stop:

A "tab stop" is a metal positioner that may be inserted in any notch along the "tab rack". If a tab stop is placed in a numbered notch, it will stop the carriage in a numbered position coinciding with this numbered notch. (The carriages numbered position is shown at the "front paper scale" indicator.) The carriages numbered position must be at a lower value than the tab stop's if the latter is to have any effect on the carriage's position.

Margin Stop:

The "margin stop" is the sliding assembly mounted on the margin rack. The "margin stop" assembly may be moved by pressing on the middle of the margin stop assembly and sliding it along the rack. The right end of the margin stop is the indicator. By setting the margin stop at a particular number on the margin rack, the carriage's left margin position will be determined.

Paper Release Lever:

The "paper release lever" is located at the top left hand corner of the movable carriage assembly. When this lever is pulled forward, pressure is released from the paper to allow straightening or removal.

Line Space Lever:

The "line space lever" is located just to the right of the paper release lever. It permits selection of single, double or triple spacing between lines.

Carriage Release Buttons (right and left):

The two buttons are located above and to the right and left of the platen. When either or both are held down, the entire carriage assembly may be easily moved rightward or leftward.

Feed Knob (read and punch):

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

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 Button:

The "platen variable button" is located in the center of the left platen knob. When this button is depressed the platen is released from discrete motion to allow the operator vernier line space adjustment through the use of the platen knob.

Carriage Return Rack:

The carriage return rack is the rack that is visible from the top of the typewriter unit. This movable rack passes a fixed contact lever mounted to the rear of it.

Carriage Return "Tabs":

The insertion of a carriage return "tab" stop is capable of automatically returning the carriage to its left most position. Care must be taken in inserting this tab in the proper carriage return rack slot. If the operator desires to obtain a carriage return when a particular position of the carriage is reached, he should do the following:

1. Position the carriage so that the last printing position of the line to be printed is at the printing station.
2. Sight the carriage rack notch that lines up with the contact levers to the rear of the rack.
3. Move the carriage assembly to the right or left to permit insertion of a carriage tab stop in this notch without touching the contact with the "tab" stop.
4. Insert the carriage tab stop in the notch.

This procedure will assure an automatic carriage return from this carriage position whenever a key is manually depressed on the keyboard while in this carriage position.

If a carriage return is desired from this carriage position while reading a tape or while carriage movement is controlled by the program, the above procedure is a necessary but not a sufficient one. To obtain an automatic carriage return under either of these two conditions a "tab stop" must also be placed in the corresponding "tab rack" notch; furthermore, a "tab" <u>into this position</u> must occur.

SECTION XII
PROGRAMMING A LOOP

One of the most basic and powerful techniques in programming is that of programming a set of instructions that apply to more than one set of memory addresses. That is, the set of instructions perform a basic function many times but use different values (i.e. different addresses) each time. Such a set of instructions is referred to by programmers as a loop.

As a basic example let us suppose we desire to move the words on track 14 to track 56. It is obvious that such a programming task could be accomplished by giving the following set of instructions on any track. (say track 39).

Loc.	Order	Address	Notes
3900	B	1400	Contents 1400 to Acc.
3901	C	5600	" Acc. to 5600
3902	B	1401	
3903	C	5601	
3904	B	1402	
3905	C	5602	
3906	B	1403	
3907	C	5603	
etc.	etc.	etc.	

Such a set of instructions, if expanded to use 64 brings and 64 clears (total 128 instructions), would accomplish the task. This type of programming is efficient when the machine time is the only consideration. When memory capacity is at a premium and we prefer to use as few instructions as possible, this type of programming is wasteful. It might also be mentioned here that programmers, in general, strive to reduce the number of instructions in a program, even though it might require a little more machine time.

This same programming task could be accomplished in the following manner:

Loc.	Order	Address	Notes
3900	B	3915	
3901	Y	3904	
3902	B	3916	
3903	Y	3905	
3904	B	[ ]	
3905	C	[ ]	
3906	B	3904	
3907	A	3918	
3908	Y	3904	
3909	B	3905	

Loc.	Order	Address	Notes
3910	A	3918	
3911	Y	3905	
3912	S	3917	
3913	T	3904	
3914	Z	0000	Stop - Finished
3915	Z	1400	
3916	Z	5600	
3917	C	5700	
3918	Z	0001	

The Y order stores the address portion of the accumulator (12 bits only) in the address portion of the instruction represented by the address of the Y instruction. For example, on step 3901 the accumulator would contain Z 1400. Executing the instruction at location 3901 would place the address that is contained in the accumulator in 3904. That is, the instruction at 3904 would then read B1400.

The instructions at locations 3900 through 3903 are referred to as "set up instructions". They set up initial addresses in memory locations 3904 and 3905. If these 4 set up instructions were not used and instructions 3904 and 3905 were made B 1400 and C 5600 the program would work--BUT ONLY FOR THE FIRST TIME AFTER RECORDING THE PROGRAM FROM TAPE: All experienced programmers use this "set up" policy. Failure to adopt this technique will not only brand a programmer as inexperienced; but what is even worse, his programs may frequently give incorrect results.

A third method for programming this problem is given below. This third method is rarely used in programming a loop but it should be mastered before attempting to understand the bootstrap program in the following section.

Loc.	Order	Address	Notes
3900	B	3914	
3901	Y	3904	
3902	B	3915	
3903	C	3905	
3904	B	[ ]	
3905	[	]	
3906	B	3904	
3907	A	3916	
3908	Y	3904	
3909	B	3905	
3910	S	3917	
3911	T	3918	
3912	H	3905	
3913	U	3904	
3914	Z	1400	
3915	03WC	5600	
3916	Z	0001	
3917	000W	WWWJ	
3918	Z	0000	Halt - Finished

Again instructions 3900 through 3903 are set up instructions. The instructions at 3904 and 3905 perform the actual shifting from track 14 to track 56. (Track 14 is unaltered). The address of instruction 3904 is modified as before by instructions 3906, 3907, and 3908. The entire instruction at 3905 is modified by instructions 3909 through 3912. This modification requires a detailed explanation.

Instruction 3905 is initially set up to be 03WC5600. Remembering that the computer ignores the leading 12 bits of an instruction, the computer will execute the instruction at 3905 as a standard C5600. The 03W in the first 12 bits may be expressed in binary as 0000 0011 1111 at a q of 11. This is just another way of expressing the decimal number 63. Our purpose in carrying 63 in this way is to furnish us with an (N - 1) counter which will tell us the number of times to execute the loop. We will subtract a 1 from this counter each time we execute a loop and after the 64th time through the loop the counter will turn negative.

If we analyze the bits in the accumulator after executing instruction 3909 the first time we find:

```
0000   0011   1111   1101   0011    1000    0000    0000
 0      3      W      C       Track               Sector
   63 Decimal                 56                  00
```

After subtracting the following arrangement of bits at instruction 3910.

```
0000   0000   0000   1111   1111    1111    1111    1100
 0      0      0      W      W       W       W       J
```

We obtain:

```
0000   0011   1110   1101   0011    1000    0000    0100
 0      3      Q      C       Track              Sector
                              56               01
```

This final form shows that by subtracting 000WWWWJ we have increased the sector portion by one and decreased the (N-1) counter by one. In this way we have accomplished two functions with one instruction. The remainder of the program is similar to the second example.

SECTION XIII
BOOTSTRAP PROGRAM

If the reader has read the preceding pages, he has had all the fundamentals required to proceed with the following description of how the bootstrap and program input are placed in the computer.

The LGP-30 "bootstrap program" consists of a set of instructions that will read itself into the computer and then transfer control to a section of itself which in turn will call in the "program input" routine.

So, one might say that the LGP-30 "bootstrap program" consists of two individual bootstraps. The first bootstrap (5 instructions) is put in track 63 manually. This first bootstrap is then used to read in the second bootstrap and automatically transfer control to this second bootstrap which reads the program input routine into tracks 00, 01, and 02.

If the reader will recall, one of the reasons we desire a "program input routine" in the LGP-30 is to convert our decimal instructions to binary. Without the "program input" routine we may not enter decimal instructions; consequently, the bootstrap and "program input routine" itself must be written in hexadecimal. In order to simplify matters, this hexadecimal bootstrap and hexadecimal program input are placed on one tape and supplied to all customers.

We shall first illustrate the basic bootstrap (5 instructions) in decimal. We chose to put it on track 63.

Loc.	Order	Address
6300	P	0000'
6301	I	0000'
6302	C	6305'
6303	P	0000'
6304	I	0000'

These 5 instructions must be placed in the computer manually. The first 12 hexadecimal instructions on the "bootstrap" program input tape are given below. The decimal equivalent of each is shown at the left. A "start read" depression is required for each of the 12 hexadecimal words. Before depressing the "start read" for the first time, the operator should be sure that the computer is in the "manual input" mode of operation and the typewriter "connect" switch is set to "off". The console button depressions required (following each "start read" depression) and their effect are given to the right of the hexadecimal representation. The first "start read" depression causes a note to print.

Dec. Equiv.	Hex. Tape	Console Button and Interpretation
C 6300	000C 3W00'	Depress fill instruction (a) Places C6300 in instruction register.
P 0000	000P 0000'	Depress one operation - depress execute instruction - depress manual input. (a) Clears P0000 into 6300.
C 6301	000C 3W04'	Depress fill instruction (a) Places C 6301 in instruction register.
I 0000	000I 0000'	Depress one operation - depress execute instruction - depress manual input. (a) Clears I0000 into 6301.
C 6302	000C 3W08'	Depress fill instruction. (a) Places C 6302 in instruction register
C 6305	000C 3W14'	Depress one operation - depress execute instruction - depress manual input (a) Clears C 6305 into 6302
C 6303	000C 3W0J'	Depress fill instruction. (a) Places C 6303 in instruction register
P 0000	000P 0000'	Depress one operation - depress execute instruction - depress manual input. (a) Clears P0000 into 6303
C 6304	000C 3W10'	Depress fill instruction (a) Places C 6304 in instruction register
I 0000	000I 0000'	Depress one operation - Depress execute instruction - depress manual input. (a) Clears I0000 into 6304.
U 6300	000U 3W00'	Depress fill instruction. (a) Places U 6300 into instruction register.
Z0000	000Z 0000'	Depress one operation - depress execute instruction - depress manual input. (a) The computer executes the instruction U 6300.

At this point the counter register (top scope window) reads track 63 sector 00. This indicates that when the "start" button is depressed (in normal mode of operation), the computer will execute instructions beginning in 6300 at high speed.

One more "start read" depression will cause a note to print informing the operator to place the "connect" switch to "on" and go into normal operation. After doing these two things the operator may depress the "start" button on the computer console. This depression will read in the second bootstrap and transfer control to it.

The decimal coding for the second bootstrap is given below.

Loc.	Order	Address
6306	U	6300'
6307	P	0000'
6308	I	0000'
6309	OGW C	0000'
6310	B	6346'
6311	S	6326'
6312	U	6322'
6313	Z	0000'
6314	U	0000'
6322	T	6313'
6323	H	6309'
6324	C	6346'
6325	U	6307'
6326	000 W	WWWJ'
6346	OGW C	0000'

The first bootstrap (5 instructions) will place this second bootstrap in the computer and transfer control to 6307. The basic concept of this is illustrated below. The hexadecimal tape is shown with the decimal equivalent given to the left of each instruction. The reader's attention is called to the fact that there is a clear order preceding each instruction that is actually to be placed on track 63. The clear order is the first (of each pair) to be read. The instructions at 6300 and 6301 reads this clear instruction into the accumulator. The instruction at 6302 then places the clear instruction into 6305. The instructions at 6303 and 6304 read the instruction that is to be actually stored into the accumulator. Then at instruction 6305 the instruction is actually stored in the proper sector on track 63. Instruction 6306 transfers control back to 6300 to repeat the process for the next pair on tape.

```
    C 6306          C 3W18'
    U 6300          U 3W00'
    C 6307          C 3W1J'
    P 0000          P 0000'
    C 6308          C 3W20'
    I 0000          I 0000'
    C 6309          C 3W24'
191 C 0000      OGW C 0000'
```

```
      C 6310       C 3W28'
      B 6346       B 3WG8'
      C 6311       C 3W2J'
      S 6326       S 3W68'
      C 6312       C 3W30'
      U 6322       U 3W58'
      C 6313       C 3W34'
      Z 0000       Z 0000'
      C 6314       C 3W38'
      U 0000       U 0000'
      C 6322       C 3W58'
      T 6313       T 3W34'
      C 6323       C 3W5J'
      H 6309       H 3W24'
      C 6324       C 3W60'
      C 6346       C 3WG8'
      C 6325       C 3W64'
      U 6307       U 3W1J'
      C 6326       C 3W68'
000   W WWWJ   000 W WWWJ'
      C 6346       C 3WG8'
191   C 0000   OGW C 0000'
      U 6307       U 3W1J'
```

The reader's attention is called to the last pair indicated above. The U3W1J is read into the accumulator by the instructions at 6300 and 6301 and then instruction 6302 places the U3W1J (U6307) into location 6305. The P0000 and I0000 at 6303 and 6304 reads a conditional stop code. Instruction 6305 then transfers control to 6307 and the hexadecimal "program input routine" is loaded into tracks 00, 01, and 02. Since this second bootstrap counts the number of instructions in the "program input routine", a stop is executed after loading. (stop at 6313) If the operator desires, he may depress the start button and control is transferred to track 00 sector 00.

SECTION XIV
PROGRAM INPUT ROUTINE
(Prog. 10.4)

This Program Input routine is punched in hexadecimal on the "Bootstrap - Program Input" tape and is supplied to all LGP-30 users. This tape will place the program input routine on tracks 00, 01, and 02.

Transferring to the first instruction of the program input routine will read a word into the computer. Every word read into the computer by the program input routine may be classified into one of seven basic categories. Classification into one of these categories is based upon the binary "code" represented in the four high order accumulator bits. For this reason each word read in by the program input routine is referred to as a code word. A decimal instruction is a code word whose 4 high order accumulator bits appear as 0000 (code zero).

In summarizing the above paragraph, transferring to the first instruction of the program input routine reads a "code word" into the accumulator. The program input routine then analyzes the "code" and branches in one of seven basic directions to execute the requirements defined by the code word.

It now becomes apparent that an operator must be able to transfer to the first instruction of the program input routine to "trigger this chain reaction." The program input routine's first instruction was placed on track 00 sector 00 to facilitate the operator's transferring control to this location. This may be accomplished by performing the following procedure:

1. Depress the "one operation" button.
2. Depress the "clear counter" button.
3. Depress the "normal" button.
4. Depress the "start" button.

These four operations are basic and the function of each should be clearly understood. The first three instructions of the program input routine (located on 0000, 0001, and 0002) are C 0143, P 0000, I 0000. These 3 instructions clear the accumulator to zero and supply a typewriter input order. It was explained in the section on Typewriter Controls that the "manual input" lever determines whether the tape reader is set in motion or the computer stops after the execution of the P0000 and I0000 instructions. So by setting the "manual input" lever prior to giving four basic operations listed above, the operator may select whether the "code word" is to come from keyboard or tape.

The computer's counter register (top scope window) will always be at 0003 while this word is being entered into the computer, whether it be from keyboard or tape. Control will be transferred to 0003 when a conditional stop code is reached on tape or when the "start compute" lever is depressed in the case of a keyboard entry. At location 0003 the program input routine begins to analyze the word placed in the accumulator. This analysis is based on the "code" placed in the four high order accumulator bits. Control is transferred to one of seven sets of instructions.

The seven basic "codes" that may be represented in the 4 high order accumulator bits are given below:

Typewriter Key Code	Type Codes	Interpretation
(None)	0000	Instruction
+	0010	Command
;	0011	Start Fill
/	0100	Set Modifier
.	0101	Stop and Transfer
,	0110	Hexadecimal Words
V	0111	Hexadecimal Fill

These seven basic "codes" will now be discussed individually in detail.

(;) Start Fill:

One location (word) of the program input routine is used as a counter to indicate to the routine where the next instruction or hexadecimal be stored. This counter is commonly referred to as the "start fill" counter. The "start fill code" word is used to place an initial value into this counter. Each time another word is read in and stored the"start fill"counter is increased by one. The entire start fill word will consist of ; 000$m_1m_2m_3m_4$. Where: $m_1m_2m_3m_4$ represents in decimal the initial address into which the next word stored will be placed. The binarized $m_1m_2m_3m_4$ will be placed in the"start fill"counter.

(/) Set Modifier:

When programming small problems for the LGP-30, the programmer may program a set of instructions and know (while he programs) exactly where this program will be stored in memory. If a large problem is being programmed by many programmers, this may not be the case. It then becomes necessary to be able to program instructions and later decide exactly where the set of instructions should be placed in memory.

For instance, the program at the bottom of page 35 and top of page 36 will work properly only if placed on track 39. However, suppose that the track number (39) had not been chosen at the time this program had been devised. In that event that programmer could have used a tentative track number, for example "00", and could have completed the writing of his program with respect to this "00" track. Then, later, when the actual track number had been selected, the programmer could use the program input routine to modify the tentative track "00" number of those instructions requiring such changes. The following program is equivalent to the one on pages 35 and 36. The program on the following page is preferred over the other one since it may be placed on any other track by just changing the start fill and set modifier instructions. The modifier (/0003900) will add 3900 to the address of every instruction whose order is not preceded by an "X".

Prog. Input Code	Loc.	Order	Address
;0003900			
/0003900			
	0000	B	0015
	0001	Y	0004
	0002	B	0016
	0003	Y	0005
	0004	B	[ ]
	0005	C	[ ]
	0006	B	0004
	0007	A	0018
	0008	Y	0004
	0009	B	0005
	0010	A	0018
	0011	Y	0005
	0012	S	0017
	0013	T	0004
	0014	XZ	0000
	0015	XZ	1400
	0016	XZ	5600
	0017	XC	5700
	0018	XZ	0001

The addresses at 0015, 0016, and 0017 are fixed by the statement of the problem and will not vary when we relocate our program. For this reason an "X" should precede the order. The 1 at 29 (instruction 0018) is a constant and the "X" prevents this constant from being altered. The "X" was placed before the order in instruction 0014, to prevent the program input routine from changing it to an instruction dependent upon a break point switch setting.

Programmers are urged to use the modifying feature in order to permit shifting the program to another memory location at a later date.

The /0003900 will set the "modifier" of the program input routine at 3900. The "modifier" will remain at 3900 until changed by a new (/) code. If a programmer is not using the modifying feature he must set the modifier to zero by preceding his instructions with the /000 0000 code. Failure to do this will result in all of his non "X" instructions having their address increased by whatever happens to be in the program input routine's modifier.

(None) Instruction

Since the program input routine clears the accumulator to zero before giving the P0000, I0000 instructions, all instruction words on tape will leave a zero code (0000) in the 4 high order accumulator bits. When the program input routine finds this zero code, it converts the decimal track and sector to binary. Then it places the binary representation of the instruction in the memory location given by the "start fill" counter. A "-T" instruction is handled in the same manner even though it contains a (1000) code in the 4 high order accumulator bits.

After the instruction is properly stored, the program input routine increases the "start fill" counter by one.

(,) Hexadecimal Words

In Section VIII we converted the number π to hexadecimal. At times it is desirable to enter a hexadecimal number (such as π) into the computer. The number π at a q = 3 was found to be 3243W6G4. Since we have already converted the decimal number to binary, we would like to prevent the program input routine from altering this word. Use of the "program input code" ,0000001 will cause the program input routine to place the next word on tape in the memory location given by the "start fill" counter. The "start fill" counter is increased by one after storing each hexadecimal word.

One hexadecimal code word may be used to place as many as 63 hexadecimal words in memory. The number of hexadecimal words following the "hexadecimal word - program input code" is denoted (in decimal) by the last two digits of the code word. That is ,0000056 would denote that 56 hexadecimal words were to follow. The reader is referred to the Subroutine Manual (any subroutine) for examples of the hexadecimal word.

(V) Hexadecimal Fill

From the discussion of the "hexadecimal word" code above, the reader should have surmised that inputing a hexadecimal word is accomplished in less time than standard decimal words. This is true because the computer does not need to perform the decimal to binary conversion.

In order to take advantage of this faster input, hexadecimal punch out routines were developed. These routines may be used to punch out an entire program in hexadecimal. These punched out hexadecimal tapes always contain a $vn_1n_2n_3$ $m_1m_2m_3m_4$ (8 digit) code. Later, when the tape is used as an input tape, the program input routine will detect the "V" and branch to the "hexadecimal fill" portion of the program input routine. The $n_1n_2n_3$ represents in hexadecimal the number of hexadecimal words to be read in. The $m_1m_2m_3m_4$ denotes in hexadecimal where the first hexadecimal word on tape is to be stored. The $n_1n_2n_3$ words will be filled consecutively.

Use of the hexadecimal tape (V code) is preferred over the decimal tape, once the program has been checked out.

(.) Stop and Transfer

In writing the program input routine, it was felt that a code to transfer control to any memory location would be desirable. The period code serves this purpose. It may be placed at the end of the program tape. After the program input routine detects this period code on the program tape, a stop is executed.

The period code is of the following form:

.000 $m_1m_2m_3m_4$

Where: $m_1m_2m_3m_4$ represents in decimal the drum location to which a transfer is to be made.

After execution of the stop, depressing the "start button" will transfer control to $m_1m_2m_3m_4$. The stop portion of the stop and transfer code will be ignored whenever break point switch 32 is depressed.

(+) Command

This program input code is a very handy one to use when manually changing an instruction or word of data in the computer. It may also be used to manually execute any instruction.

The "+ command" is a two part "program input code". The first part will store the command that the operator desires to execute. The second part consists of the execution of this instruction.

The following two examples should illustrate the flexibility obtainable through the use of the "+ Command" code.

Example 1.

Place the instruction A 2317 into memory location 4649.

Procedure:

1. Depress "one operation" button.
2. Depress "clear counter" button.
3. Depress "normal" button.
4. Depress typewriter "manual input" lever.
5. Depress console "start" button.
6. After typewriter light glows, type +00C4649, (the instruction to be executed) on the keyboard.
7. Depress typewriter "start compute" lever. (This stores the instruction to be executed).
8. After typewriter light glows again, type A 1744 (A 2317) on the keyboard.
9. Depress typewriter "start compute" lever (this actually executes the instruction stored by part 1).

The instruction A 2317 could be stored in memory location 4649 through proper use of the "start fill" and "set modifier" codes. This method is longer but the use of a hexadecimal address is avoided.

Example 2.

Bring the contents of memory location 3761 to the accumulator.

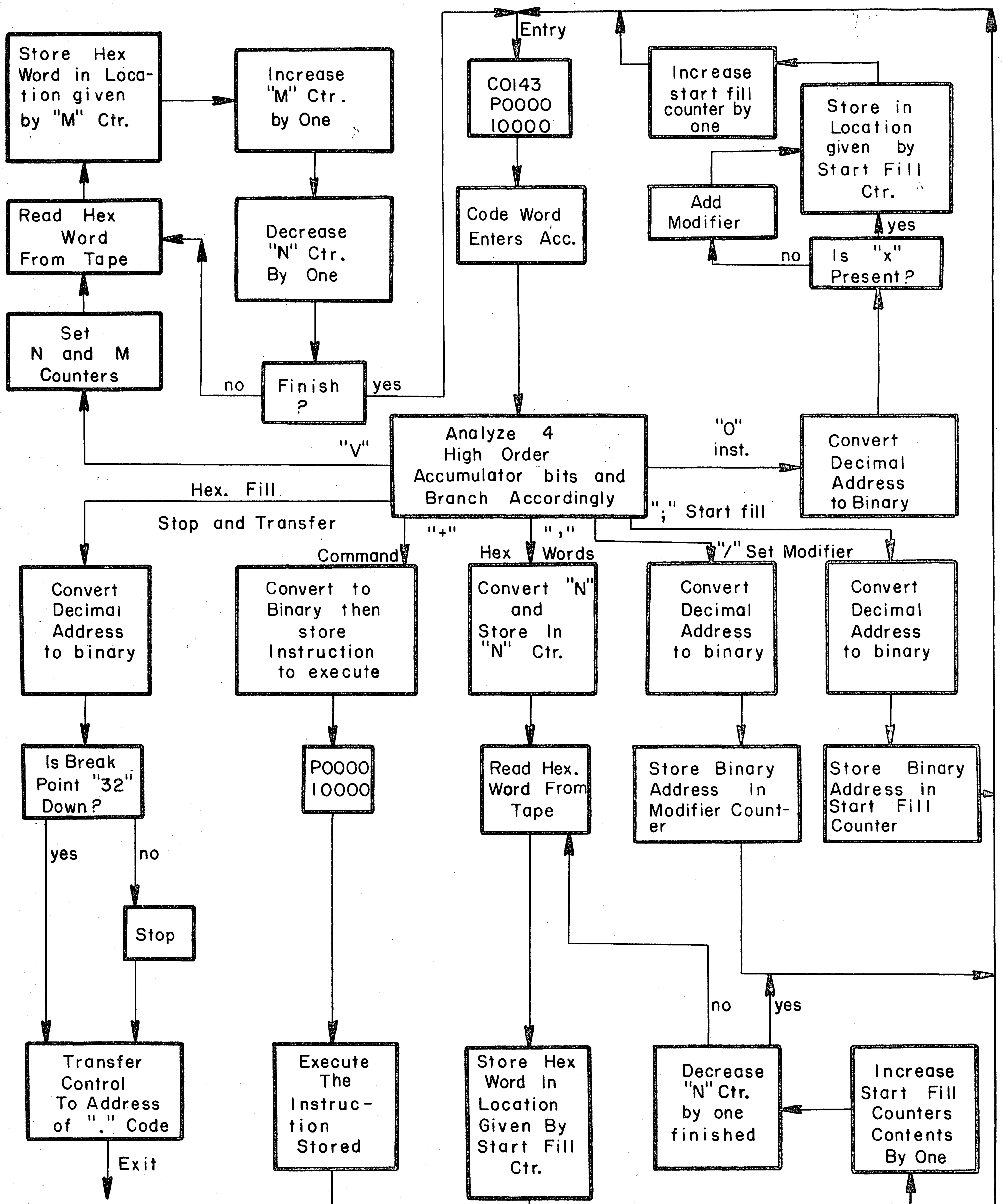
Procedure:

Steps 1 through 5 of example 1 are identical.

6. After typewriter light glows, type +00B3761 (the instruction to be executed) on the keyboard.
7. Depress typewriter "start compute" lever. (this stores the instruction to be executed).
8. Depress "one operation" button.
9. Depress typewriter "start compute" lever. (this actually executes the instruction stored by part 1)

Following the execution of step 9, the contents of 3761 will be displayed in the scope accumulator window.

The flow chart on the following page may help tie together the basic program input code words.

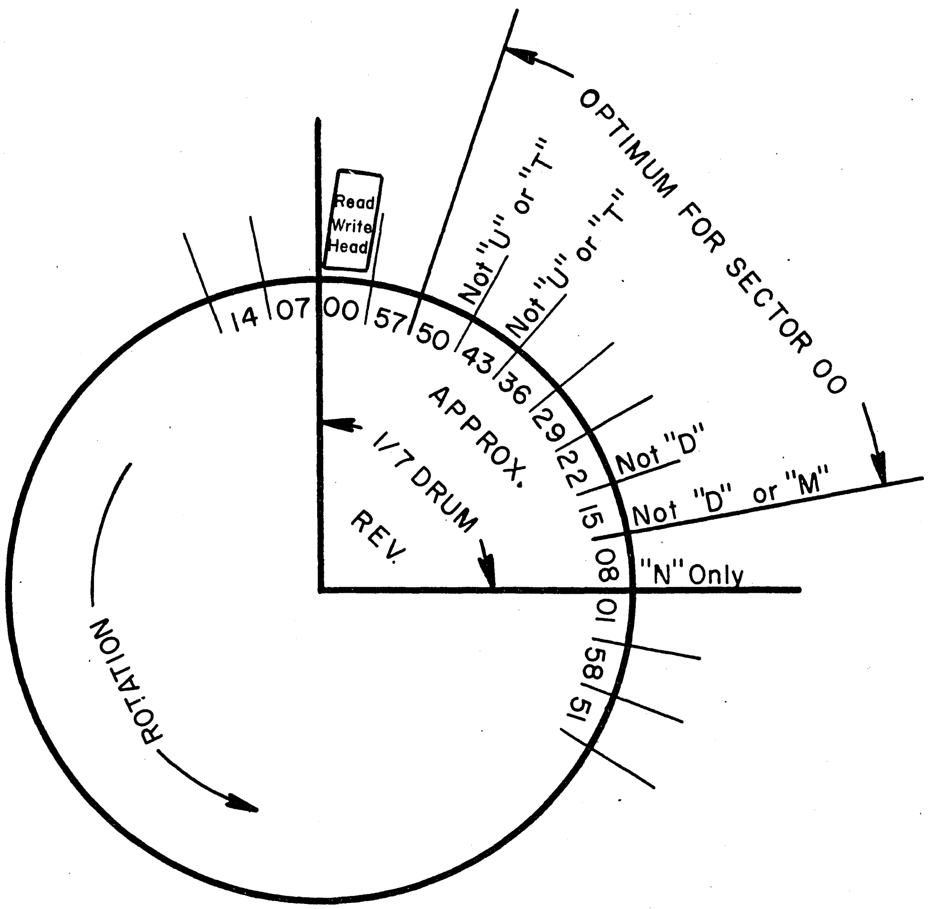


Entry
C0143 P0000 10000
Code Word Enters Acc.
Analyze 4 High Order Accumulator bits and Branch Accordingly
"V"
Set N and M Counters
Read Hex Word From Tape
Store Hex Word in Location given by "M" Ctr.
Increase "M" Ctr. by One
Decrease "N" Ctr. By One
Finish ?
no
yes
"0" inst.
Convert Decimal Address to Binary
Is "x" Present?
yes
no
Add Modifier
Store in Location given by Start Fill Ctr.
Increase start fill counter by one
Hex. Fill
Stop and Transfer
"+" Command
"," Hex Words
";" Start fill
"/" Set Modifier
Convert Decimal Address to binary
Is Break Point "32" Down?
yes
no
Stop
Transfer Control To Address of "." Code
Exit
Convert to Binary then store Instruction to execute
P0000 10000
Execute The Instruction Stored
Convert "N" and Store In "N" Ctr.
Read Hex. Word From Tape
Store Hex Word In Location Given By Start Fill Ctr.
Convert Decimal Address to binary
Store Binary Address In Modifier Counter
Convert Decimal Address to binary
Store Binary Address in Start Fill Counter
Increase Start Fill Counters Contents By One
Decrease "N" Ctr. by one finished
no
yes

SECTION XV
OPERATING TIMES - OPTIMIZING

The LGP-30 will execute an addition or subtraction in .26 ms. (excluding access time). The same time applies to all other instructions except multiply, divide and transfer instructions. The execution of a multiply or divide instruction requires 16.66 ms. (exluding access time). The minimum execution time for a transfer instruction is 1 ms.

The following discussion explains the idea of "access time". Two columns of figures will be found along the right hand edge of the powers of 2 table sheet. The right most column indicates in decimal the sequence of sectors found on each track. The column to the left of it gives the hexadecimal representation of these sectors.



The sketch above shows a cross sectional view of the drum at any track.

The particular track that this cross section depicts is immaterial but for purposes of illustration let us say it shows track 15. The drum rotation is indicated to be counter clock-wise. The time required for each sector to pass under the head is approximately .25 ms. This is commonly referred to as one word time. The time required per drum revolution is approximately 17 ms. (actually 16.66+). In the following discussion the approximate times are used since the figures are easier to work with. In practice these approximate times are adequate.

The memory location 1500 could contain a "bring" instruction. After sector 00 passes the head the computer analyzes the order bits of sector 00 and determines a "bring" has to be executed. By this time sector 57 has partially passed beyond the head and a B 1557 could not be executed without waiting for an entire drum revolution. The LGP-30 is capable of "bringing" sector 50, 43, 36, 29, 22, or 15 in time to accept the next instruction from location 1501 as it passes the head. In this way, an instruction may be "optimally" executed in approximately 1/7 of a drum revolution. If a B1508 were located in memory location 1500, it would not be possible in an optimum program to read 1508 and execute the B1508 in time to accept the next instruction from 1501.

From this discussion it should be clear to the reader that the sector portion of an instruction's address determines whether or not the instruction is an "optimum" one. It should be clear that the track portion of the "instruction" does not affect the time required to execute an instruction.

The following table illustrates the breakdown in ms. for all instructions except transfer, multiply, and divide when the instruction is given at sector 00.

Address Sector of Instr.	Time Req'd. to Read Instr.	Access Time to Operand	Time Req'd. to Read Operand	Exec. Time (.26)	Access Time to next Inst.	Total Time
57	.25	17.00	.25	.25	1.50	19.25
50	.25	.25	.25	.25	1.25	2.25
43	.25	.50	.25	.25	1.00	2.25
36	.25	.75	.25	.25	.75	2.25
29	.25	1.00	.25	.25	.50	2.25
22	.25	1.25	.25	.25	.25	2.25
15	.25	1.50	.25	.25	.00	2.25
08	.25	1.75	.25	.25	16.75	19.25

From this table it is apparent that for all instructions except transfer, multiply, and divide, either 2.25 ms. or 19.25 ms. are required to read any instruction, execute it and be in a position to read the next instruction. In short, an instruction is either optimum or unoptimum.

From the sketch, it can be seen that if a transfer (U) instruction is given in sector 00, the fastest transfer will be one to sector 36. Each sector following sector 36 will add approximately .25 ms. to the minimum transfer time of 1 ms. Search for the sector of a "T" instruction is made only when an actual transfer is being executed. If a "T" instruction is given in sector 00 and a transfer is not actually made, the instruction at sector 01 will be read (for execution) as it passes the head on that drum revolution.

Execution of the "M" instruction requires 66 word times, the "N" instruction 64 word times, and the divide 67 word times. All three of these instructions are said to require 17 ms. in various sales brochures. Since we are dealing here with optimizing, we must consider the instructions with a little more care.

If we consider the case where sector 00 contains an "N" order, sectors 50, 43, 36, 29, 22, 15 and 8 are all optimum sectors. The table given below illustrates the times required when an "N" instruction is given in sector 00.

In the event that an "M" instruction is given in sector 00, sectors 50, 43, 36, 29, and 22 are optimum locations. The table indicates the times required when an "M" instruction is given in sector 00. If a divide instruction is given in sector 00, sectors 50, 43, 36, and 29 are optimum locations. The table indicates the times required when a divide instruction is given in sector 00.

"N" INSTRUCTION

Address Sector of Instr.	Time Req'd. to Read Instr.	Access Time to Operand	Time Req'd. to Read Operand	Exec. Time	Access Time To Next Inst.	Total Time
57	.25	17.00	.25	17.00	1.75	36.25
50	.25	.25	.25	17.00	1.50	19.25
43	.25	.50	.25	17.00	1.25	19.25
36	.25	.75	.25	17.00	1.00	19.25
29	.25	1.00	.25	17.00	.75	19.25
22	.25	1.25	.25	17.00	.50	19.25
15	.25	1.50	.25	17.00	.25	19.25
08	.25	1.75	.25	17.00	.00	19.25

"M" INSTRUCTION

Address Sector of Instr.	Time Req'd. to Read Instr.	Access Time to Operand	Time Req'd. to Read Operand	Exec. Time	Access Time To Next Inst.	Total Time
57	.25	17.00	.25	17.50	1.25	36.25
50	.25	.25	.25	17.50	1.00	19.25
43	.25	.50	.25	17.50	.75	19.25
36	.25	.75	.25	17.50	.50	19.25
29	.25	1.00	.25	17.50	.25	19.25
22	.25	1.25	.25	17.50	.00	19.25
15	.25	1.50	.25	17.50	16.75	36.25
08	.25	1.75	.25	17.50	16.50	36.25

"D" INSTRUCTION

Address Sector of Instr.	Time Req'd. to Read Instr.	Access Time to Operand	Time Req'd. to Read Operand	Exec. Time	Access Time to next Inst.	Total Time
57	.25	17.00	.25	17.75	1.00	36.25
50	.25	.25	.25	17.75	.75	19.25
43	.25	.50	.25	17.75	.50	19.25
36	.25	.75	.25	17.75	.25	19.25
29	.25	1.00	.25	17.75	.00	19.25
22	.25	1.25	.25	17.75	16.75	36.25
15	.25	1.50	.25	17.75	16.50	36.25
08	.25	1.75	.25	17.75	16.25	36.25

The above discussion has all dealt with instructions in sector 00. When an instruction is given in any other sector the basic concepts still apply but the address to which they apply changes. As an example, instructions located in sector 43 can be analyzed in the following way:

1. The fastest transfer is to sector 15. Sector 08 requires an extra .25 ms. etc.
2. Optimum "N" multiply locations are 29, 22, 15, 08, 01, 58, and 51.
3. Optimum "M" multiply locations are 29, 22, 15, 08, and 01.
4. Optimum divide locations are 29, 22, 15, and 08.
5. Optimum locations for all other orders are 29, 22, 15, 08, 01, and 58.

In closing, it might be stated that all Subroutines programmed by the Royal McBee programming group have been optimized. It was felt that the time saved by optimizing these basic routines was justified. Programmers are cautioned to carefully compare the savings in machine time cost realized by an optimum program against the added programming costs required to obtain the program. In most cases the added programming costs will exceed the saving in machine time costs. This is, of course, a function of the application, but extreme care should be taken before deciding to optimize any program.

SECTION XVI
PRINT INSTRUCTION

The explanation of the print instruction has been delayed until the reader has acquired knowledge of operating times and the "x" notation of the program input routine.

When a print instruction is executed, the typewriter will print a character or perform another typewriter function. The function performed is dependent upon the arrangement of the six bits in the track portion of the print instruction. To print a character on the typewriter a "P" instruction with an arrangement of track bits corresponding to this character's bit code must be given.

Normally the Data Output, Hexadecimal Print or Punch out, or Alpha-numeric output subroutines automatically set up the track portion of the print instruction for the programmer. Occasionally the programmer decides to set up the track portion of the print instruction himself and it is for this reason that the following discussion is included.

To simplify the programmer's task of determining the arrangement of track bits required to execute a "P" instruction, a table is supplied on the powers of 2 sheet, and is labeled "Keyboard Codes". This table lists all typewriter characters and functions in the first and fourth columns. The entries LC, UC, CS, CR, and BS represent lower case, upper case, color shift, carriage return and back space respectively. For all other entries that consist of two characters, the first character indicates the upper case code and the second the lower case code. The entry appearing two columns to the right of each typewriter character entry is the decimal track representation required to print that character.

For instance, the decimal track representation for a 5 is shown to be 22. This means that when programming, to print a 5 the decimal instruction P 2200 should be given. This track 22 will be converted by the program input routine to binary 22 and will appear as 010110. This is the keyboard code for a 5 and may be verified on page 73. All characters are printed and all keyboard functions are executed in this manner.

The Sector portion of the "P" instruction is used by the computer in the manner discussed in the Section on Operating Times and Optimizing. That is, any Sector may be given but an optimum sector will start the typewriter function one drum revolution sooner.

After the "P" instruction supplies the typewriter unit with the required information to execute a print or control function, control is transferred to the instruction following the "P" instruction. The programmer may feel free to use any instruction except another "P" instruction. If a "P" instruction is to be given within 100 ms. after giving a previous "P" instruction, the "Z" (stop) instruction must precede this second "P" instruction. This "Z" instruction causes the computer to stop until the first "P" instruction has been completely executed. After the typewriter unit completes the execution of every "P" instruction, it supplies a "start signal" back to the computer. If the computer is stopped, this start signal will cause the computer to proceed to the instruction whose address is given by the counter register. (Normally this is the instruction following the "Z" instruction). If the computer is running when this start signal arrives, the start signal will have no effect. Programmers are cautioned not to use the "Z" instruction more than 100 ms. after giving a "P" instruction unless a stop is really desired.

The carriage return and tab require more than 100 ms. and of course are a function of the distance the carriage must travel. Long carriage returns can require as many as 1100 ms. and tabs 800 ms. It is advisable to follow carriage return and tab instructions with a "Z" instruction if the programmer is in doubt as to whether another print instruction will be given in the next second or so.

In programming "P" and "Z" instructions it is advisable to place an "X" before the order on the programming sheet. This prevents the program input routine from modifying the address portion of the instruction.

SECTION XVII
BINARIZATION OF DECIMAL ADDRESS

The following method of binarizing a decimal track and sector to its corresponding binary track and sector lends itself very nicely to the LGP-30. This method was derived and first used by Dr. Tom Kampe of Librascope. The following presentation of the derivation is not very rigorous. A formal derivation may be obtained by contacting Dr. Tom Kampe.

Since we have found this procedure so useful, we included it to enable LGP-30 users to incorporate it in programs of their own, should the occasion arise. The presentation is slanted toward the application of converting track and sector but the same general approach applies to data also. The derivation deals with converting a decimal track or sector but in the explanation that follows track and sector are handled simultaneously.

Let C represent the 2 decimal digit track (or sector) to be converted. This number C may then be represented as:

(1) $C = R \cdot A + B$ where A and B are two contiguous digits to the base R, each binary coded with n binary digits.

As an example of (1), let us consider track 23 then:

$A = 2$
$B = 3$
$C = 23 = 10\,(2) + 3$
C is expressed in binary coded decimal as 0010 0011.

Let D be this same binary coded decimal arrangement of bits represented in pure binary.

(2) $D = A2^n + B$
For the example track 23
$A = 2$
$B = 3$
$D = 35 = (2)\,2^n + 3$

If we subtract (1) from (2) we obtain:

$$D - C = A\,(2^n - R)$$
$$C = D - A\,(2^n - R) = D + A\,(-6) \text{ when : } \quad R = 10,\; n = 4$$

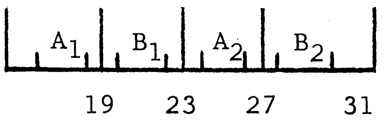
This means that B (low order 4 bits) is not needed in evaluating A(-6). That is, we would like to "extract" off the last four bits of the accumulator before multiplying by -6. The LGP-30 has an "extract" instruction which is capable of "extracting off" (setting to zero) any bit or combination of bits present in the accumulator.

The "extract" instruction consists of an E order and a 4 decimal digit address just as any other instruction does. The address of the extract instruction indicates the location of what is known as the "extractor". In executing the extract instruction the LGP-30 places an "0" bit in every accumulator bit for which the corresponding extractor bit is zero. Conversely, the LGP-30 leaves every accumulator bit unaltered if the corresponding extractor bit contains a "1".

Another way of thinking of the extract instruction is to consider a bit for bit multiplication between each accumulator bit and the corresponding extractor bit. Carries in this multiplication are ignored.

With this explanation of the extract instruction the reader is prepared to proceed with the detailed program of binarization of a decimal track and sector. The method of binarizing track and sector simultaneously is identical to the preceding discussion of track or sector with the exception that the track and sector portions occupy different portions of the accumulator.

When the 2 decimal track and 2 decimal sector enter the accumulator, they occupy the last 16 bits. We may consider the track portion as A_1 and B_1 and the sector portion as A_2 and B_2. This may be indicated in the following manner.



A_1 is at q = 19	A_2 is at q = 27
B_1 is at q = 23	B_2 is at q = 31
D_1 is at q = 23	D_2 is at q = 31

The following program will binarize:

Loc.	Order	Address	Notes
0000	N	0009	This instruction shifts accumulator value left 2 bits. A_1 now at q = 17; A_2 now at q = 25; B_1 now at q = 21; B_2 now at q = 29; D_1 now at q = 21; D_2 now at q = 29.
0001	H	0010	Save D_1 and D_2 at q = 21 and 29 respectively.
0002	E	0011	Extract off B_1 and B_2 since A (-6) does not involve B's. A_1 still at q = 17; A_2 still at q = 25.
0003	M	0012	By multiplying -6 at q = 4 by A_1 at a q = 17 and A_2 at a q = 25, we obtain A_1 (-6) at 21 and A_2 (-6) at 29.
0004	A	0010	Performing this addition gives D_1 + A_1 (-6) at q = 21 and D_2 + A_2 (-6) at q = 29. At this point both track and sector have been binarized.

Loc.	Order	Address	Notes
			The sector position is at a q of 29 just as it is to be stored. The track portion is at a q of 21 and should be stored at a q of 23. That is, we would like to shift only the track portion of the accumulator right by 2 bits. This is equivalent to multiplying the track portion by $2^{-2} = 1/4$. Since we desire then 1/4 of the track value that we have in the accumulator, we may obtain this 1/4 by subtracting 3/4 of the track value from the track value itself.
0005	H	0013	Save track value at q = 21 and sector value at q = 29
0006	E	0014	Extract off sector portion leaving track value at a q = 21 in accumulator.
0007	M	0015	By multiplying track value at q = 21 by -3/4 at q = 0 we obtain -3/4 of track value at q = 21 in the Accumulator.
0008	A	0013	This gives [track value at q = 21] + [-3/4 track value at q = 21] = [1/4 track value at q = 21] = [track value at a q = 23]. Of course the sector value is still at a q = 29.
0009	XZ	0001	1 at a q of 29
0010	[	]	temp. N_1
0011	3J3	J3J0	Extractor
0012	K000	0000	-6 at a q = 4
0013	[	]	Temp. N_2
0014	OWWW	WWOO	
0015	F000	0000	-3/4 at a q = 0.

SECTION XVIII
CORRECTING TAPES

Correcting an error on punched paper tape may be accomplished in various ways. The method of correcting the error depends generally upon the type of error. The purpose of the following discussion is to explain the basic correction techniques and encourage the reader to analyze his particular problems closely before deciding on the best method.

1. Perhaps the easiest error to correct is one which is detected immediately after the wrong key has been depressed. In this case all that needs to be done is:
   A. Turn the "feed knob" at the left side of the tape punch back one notch.
   B. Press the "code delete" lever once.
   C. Continue by depressing the proper key on the keyboard.

2. At times the operator will depress the wrong key and punch a particular combination of holes. This combination of holes might be a portion of the combination desired. If this is the case, the operator need only turn the "punch feed knob" back one notch and continue by depressing the proper key. This method is particularly useful when the error is detected after characters have been punched beyond the error.

3. Another type of error is the one in which an incorrect word is punched on tape and cannot be corrected by either of the methods given above. As an example, suppose the instruction S1234 were punched instead of the correct instruction A1234. The operator may correct this error by punching two additional words on the end of this program tape. If the A1234 instruction were to be placed in memory location 3617, the two words to be added to the tape would be:
   1) +00C3617 (see Section XIV)
   2) AOJ88 (A1234)

   This S1234 would be entered into 3617 but the A1234 would later replace it.

4. The example discussed in (3) could also be corrected by placing the following three words on tape.
   1) ;000 3617'
   2) /000 0000'
   3) A1234'

5. The most obvious method of correcting a word on tape is to reproduce the tape up to the error, punch the correct word, and then continue reproducing.

SYNOPSES OF LGP-30 SUBROUTINES

BOOTSTRAP (Program J1-09.0)

This routine loads the "Program Input Routine" on tracks 00, 01, and 02. After completing this procedure, a halt is executed at track 63 sector 13. Depressing the START switch transfers control to the first instruction of the "Program Input Routine".

Storage:

The Bootstrap routine uses 21 words on track 63. (Sectors 00 through 14, 22 through 26, and sector 46.)

Time:

The time required for storing both Program J1-09.0 and Program J1-10.0 is approximately three minutes. They are considered collectively since they are supplied to LGP-30 Computing Installations on a single tape.

PROGRAM INPUT 1 (Program J1-10.0)

This routine reads in classified or coded words which it interprets, binarizes (if necessary), and stores in chosen memory locations. The coding is accomplished in the four high-order binary bits of the words. Most coded words are "instructions" which are expressed in terms of the machine's engineered order structure and addresses of locations on the magnetic drum. Other coded words have the following associated symbols and capabilities:

Symbol		Function Performed
,	------------------	Storage of from one to sixty-three hexadecimal words.
v	------------------	Storage of an entire program-tape which has been punched previously in hexadecimal.
.	------------------	Stops the computer and transfers to a chosen memory location.
+	------------------	Permits the voluntary changing of an instruction or data-word or the execution of any instruction by keyboard direction.

Storage:

Three tracks, or 192 words.

HEXADECIMAL INPUT 1 (Program J1-10.1)

This routine reads in a hexadecimal tape that has been prepared by Program J4-13.1. This routine is superior to the Program Input 1 for storing information from hexadecimal tapes because it computes a check sum based on tape information stored in memory and compares it with a previously computed and punched sum appearing on the tape. If the two sums disagree, the word "error" is printed and a stop is executed. A hexadecimal tape is preferable to a tape with decimally typed addresses since the former requires no binarization and, hence, is faster.

Storage:

Ninety-six locations of instructions and constants.

Time:

Reading from tape: One track per minute. Computing the check sum: ten tracks per minute.

DATA INPUT 1 (Program J2-11.0R)

This routine reads punched paper tape having binary-coded configurations of decimal data. It converts these configurations to pure binary, introduces required scale factors, and stores the results in specified drum memory locations. Each data word stored is preceded by an identification word which supplies the necessary decimal and binary point location information as well as the drum location for storing the data word.

Storage:

192 locations of instructions and constants, and eight locations of temporary storage (Track 63).

Time:

20-25 words per minute.

DATA INPUT 2 (Program J2-11.1)

The useful feature of this routine is its ability to store a set of numbers (having the same "p" and "q") in consecutive memory locations. The "p", "q", and first memory location are specified by a single code word contained in memory. Use of this routine requires one data word on tape for each word to be stored in memory.

Storage:

89 locations of instructions and constants.

Time:

45-55 words per minute.

DATA INPUT 3 (Program J2-11.2)

This routine is efficient for storing several groups of data when the several numbers within a given group have the same "q" and the same number of fractional decimal digits. The numbers within a given group are stored in consecutive memory locations. The "p" and/or "q" and/or first memory location may be changed without leaving the routine. One identification word must accompany each set of data words.

Storage:

192 locations of instructions and constants. Five locations of temporary.

Time:

45-55 words per minute.

DATA INPUT 4 (Program J2-11.3)

This program is an integral part of Program H1-24.0. See program description of latter.

DATA INPUT 5 (Program J2-11.4)

This routine inputs, converts, and scales nine decimal digit data words. It functions similarly to Data Input 3 (Program J2-11.2).

Storage:

3-1/2 tracks.

Time:

40-50 words per minute.

DATA OUTPUT 1 (Program J3-12.0)

This routine will print a nine decimal digit number complete with decimal point and sign (sign if negative). Each time a number is to be printed it must first be placed in the accumulator before giving the corresponding "R-U" instructions. The programmer must scale the number to be printed to one of ten particular "q's" and indicate which "q" is being used by a code.

DATA OUTPUT 1 Modified (Program J3-12.0A)

This data output routine differs from its unmodified counterpart J3-12.0 in only one respect: it suppresses leading integral zeros when printing.

Storage:

Nineteen locations in addition to Data Output No. 1 storage.

DATA OUTPUT 2 (Program J3-12.1)

This routine will print one or more groups of numbers in decimal form. Each group has the same binary point location (q), and all numbers are printed from consecutive memory locations.

Storage:

160 locations of instructions and constants.

Time:

Approximately 30 words per minute.

DATA OUTPUT 2A (Program J3-12.1A)

This routine differs from its unmodified counterpart J3-12.1 in only one respect: it suppresses leading integral zeros when printing.

Storage:

Nineteen locations in addition to Data Output No. 2 storage.

DATA OUTPUT 3 (Program J3-12.2)

This output routine prints the contents of the accumulator as if it were in dollars and cents at q 30. This routine is primarily useful for business problems. It provides exact conversion, with leading zeroes suppressed.

Storage:

Two tracks.

Time:

1.2 seconds per word.

DATA OUTPUT 4 (Program J3-12.3)

This program is an integral part of Program H1-24.0. See program description of the latter.

DATA OUTPUT 5 (Program J3-12.4)

This subroutine has suppression of leading integral zeroes as its primary feature. It is achieved by executing spacing operations instead of printing the zeroes. It will print one or more groups of numbers in decimal form, each group consisting of one or more numbers stored in consecutive memory locations. All numbers in each group are assigned a specified scale factor "q" in the calling sequence to fix their binary points. Each output number will consist of a decimal point and eight (or more) rounded decimal digits. Each number is followed by the sign if the number is negative. A tab is executed after each number is printed.

Storage:

3-1/2 tracks.

Time:

30-35 words per minute.

HEXADECIMAL PUNCH OR PRINT 1 (Program J4-13.0)

This routine is used to punch out a group of memory locations in hexadecimal. The hex tape is subsequently read into the computer through the agency of Program Input 1 (J1-10.0).

Storage:

158 locations and instructions.

Time:

Approximately 45 words per minute.

HEXADECIMAL PUNCH OR PRINT 2 (Program J4-13.1)

This routine functions and operates in the same general manner as program J4-13.0 with the exception that after punching has been completed this routine computes a check sum and punches it as the last word on tape. The Hexadecimal Input 1 program (J1-10.1) is used when subsequently reading and storing the hexadecimalized information from tape. The check sum is intended for use by program J1-10.1 in determining whether or not the hexadecimalized tape information has been correctly stored on the drum.

Storage:

204 locations of instructions and constants. (Three tracks and 12 sectors).

Time:

The check sum is computed at approximately 7 seconds per track.

SINE-COSINE 1 (Program B1-14.0)

This fixed point routine computes the sine or cosine of the angle value, argument, or independent variable placed in the accumulator. A 9th degree polynomial approximation is used. The angle value (or independent variable) must be expressed in degrees and will be reduced to the first quadrant equivalent before computation of the function value.

Storage:

64 locations of instructions and constants. Six locations to temporary storage.

Time:

250 to 275 ms.

SINE-COSINE 2 Floating Point (Program B1-14.1)

This routine is an integral part of Program H1-24.0. See program description of the latter.

SINE-COSINE 3 Radian Argument (Program B1-14.2)

This is a fixed point routine to find Sin x, or Cos x, when x is in radians. "x" is expressed with a scale factor of three, with the result that

$$/x/ \quad < 8 \text{ radians} \qquad < 458.3 \text{ degrees.}$$

The routine uses Taylor's series for Cos x, with

$$\sin x = \cos\ (x + \frac{3}{2}\ \pi) \text{ or } \sin x = \cos\ (x - \frac{\pi}{2}).$$

The maximum error is 8×10^{-9}.

Storage:

64 words and 2 words of temporary storage on track 63.

SQUARE ROOT 1 (Program B4-15.0)

This fixed point routine computes the square root of a number placed in the accumulator. The number whose root is extracted must be scaled by a scale factor whose exponent "q" is a positive even integer.

Storage:

64 locations of instructions and constants. Five locations of temporary storage.

Time:

Square Root computed in from 500-750 ms.

ARCTANGENT 1 (Program B1-16.0)

This fixed point routine computes the arctangent of the number placed in accumulator. A 15th degree polynomial is used in the approximation process. The "principle" value (i.e. a value lying in the first or fourth quadrant) expressed in degrees will be given as output.

Storage:

64 locations of instructions and constants. Ten locations of temporary storage.

Time:

320 milliseconds.

ARCTANGENT - VECTOR RESOLUTION (Program B1-16.1)

This routine computes the arctangent of y/x where x and y are numbers scaled by the same scale factor. There are no overflow stops, and resolution in all four quadrants is possible.

Storage:

145 locations.

ARCTANGENT 2 - Floating Point (Program B1-16.2)

This routine is an integral part of Program H1-24.0. See program description of the latter.

EXPONENTIAL 1 (Program B3-17.0)

This fixed point routine will evaluate the functions e^x, or k^x where k may be "2" or "10".

Storage:

63 locations of instructions and constants.

Time:

255 to 285 ms. per evaluation.

EXPONENTIAL 2 - Floating Point (Program B3-17.1)

This routine is an integral part of Program H1-24.0. See program description of the latter.

LOG 1 (Program B3-18.0)

This fixed point routine computes the logarithm of a positive number in terms of any of the three bases "2", "e", or "10". A 7th degree polynomial is used in the approximation process. The base chosen is explicitly indicated in the routine's calling sequence.

Storage:

122 locations of instructions and constants.

Time:

Approximately (445 - 30 N) ms., where N is the number of leading zeros.

LOG 3 - Floating Point (Program B3-18.1)

This subroutine is an integral part of Program H1-24.0. See program description of the latter.

ALPHANUMERIC OUTPUT 1 (Program J4-19.0)

This routine is useful for printing page headings or distributing data at desired locations in alphabetical titles. The routine enables the programmer to "write" or express four typewriter commands in a single instruction word of the program.

Storage:

58 locations and constants.

Time:

About 400 characters per minute.

ARCSINE-ARCCOSINE 1 (Program B1-20.0)

This fixed point routine computes the arcsine or arccosine of any given value between $-1<|X<1$. A 7th degree polynomial is used in the approximation process.

Storage:

160 locations of instructions and constants. 9 locations of temporary storage are also used.

Time:

850-900 ms per evaluation.

DECIMAL MEMORY PRINTOUT 1 (Program K2-21.0)

This routine is useful when performing Program Checkout procedures. The routine causes the computer to print out the contents of any designated part of drum memory in decimal. Instruction words print in the form in which they customarily appear on the coding sheet and data print in either decimal (at q of zero) or hexadecimal, depending on the programmer's wishes.

Storage:

256 locations of instructions and constants (four tracks).

Time:

Approximately 60 words per minute.

COMPLEX OPERATIONS INTERPRETIVE ROUTINE 1 - (Program H1-22.0)

This fixed point interpretive program permits the programmer to program complex number arithmetic operations with the ease of corresponding real number arithmetic operations.

Storage: 192 locations of instructions and constants.

Time: 102 - 1697 ms. per instruction.

TRACE AND MEMORY PRINTOUT 1 (Program K1-23.0)

This routine is a member of the general class of "Program Checkout" routines. It is capable both of performing tracing operations and of executing memory printout processes. During tracing operations the routine prints out logically necessary information contained in the LGP-30 coding sheets, such as:

1. "Locations" or Instruction Counter
2. "Instruction" or Order-Address words of the program
3. "Contents of Address"
4. "Contents of Accumulator".

Each of these four items is not necessarily printed for each instruction.

Storage:

Seven tracks.

TRACE AND MEMORY PRINTOUT 2 (Program K1-23.1)

This routine, like K1-23.0, is a Program Checkout routine for tracing fixed point programs. It will perform a selective printing of items from the following logical four:

1. "Location" or Address of Instruction.
2. "Instruction" or Order-Address words of the program.
3. Contents of the Address contained in the address part of the instruction.
4. Contents of the Accumulator.

It is possible, when using this routine, to define the memory interval over which tracing and printing is to occur when executing a given problem program. The address in the problem program at which computation is to begin is also specifiable. The program description for this routine classifies the printed output on the basis of the defined memory interval, the order structure, and the time at which the printing is done (i.e. whether the printing occurs before or after the computer's execution of the instruction being traced.)

Storage:

Nine tracks.

Time:

About 3.8 seconds per instruction when monitor-printing, and about 0.7 seconds per instruction when not printing.

TRACE AND MEMORY PRINTOUT 3 (Program K1-23.2)

This routine will facilitate program-checkout for floating point programs by providing a printed record of the program's instructions together with the numerical results obtained at the completion of each instruction execution. The actual printed information given by this routine appears in five columns and gives:

1. The Address of the floating point instruction to be executed.
2. The floating point instruction to be executed.
3. The contents of the address accumulator or floating point accumulator after the execution of an instruction causing a change.
4. The contents of the memory location specified by the address appearing in the floating point instruction.
5. The contents of the multiplier register after the execution of a given floating point instruction.

Provision is made for omitting this customary printing within the traced interval for:

(a) a given sequence of instruction and/or
(b) a sequence of instruction constituting a loop.

Storage:

Eight tracks.

Time:

600-700 instructions per hour.

FLOATING POINT INTERPRETIVE SYSTEM (Program H1-24.0)

This interpretive routine redefines the LGP-30 engineered order structure so that it may be programmed as a floating point computer. The original order structure of sixteen orders has been expanded to thirty-three to include additional typical orders as well as data input, data output, and function evaluation orders. Along with the changed order structure the routine has its new format for numbers in memory as well as a set of defined registers. Numbers are represented in memory by 24 binary bits and sign for the fractional part and five bits and sign for the Exponential Part.

The newly defined registers consist of a floating point accumulator and a Multiplier Register each consisting of two words of memory, and an Address Accumulator consisting of a single word. This new system of registers permits intermediate calculations to be executed with 30 bits of Fraction and 30 bits of Exponent for the floating point numbers involved.

Twenty-six instructions covering the arithmetic, logical, address modification, auxiliary, and square root orders have been coded as a single "Floating Point Interpretive Routine", numbered as Program H1-24.0. The Data Input and Data Output Subroutines (two instructions) are customarily used in conjunction with H1-24.0; thus sixteen tracks (1024 words) of memory provide 28 instructions for a wide range of floating point problems. With this arrangement 3072 words of memory are available for storage of problem-programs. In other cases the remaining five instructions of the Interpretive Order Structure, namely the Function Evaluation instructions (excluding square root) may be needed. In those cases 23 tracks of memory provide the entire 33 instructions leaving 2624 words of memory for problem-program and data storage.

Only those routines actually used need be stored on the drum. These required routines must be stored on the drum in the following relationship: (in addition to showing the memory relationships the following table shows the program numbers which also appear in the complete LGP-30 list, printed periodically in the LGP-30 Newsletter).

Program No. & Title	Start Fill	Set mod.	Number of Tracks	Number of Words
H1-24.0 - F.P. Interpretive	L	L	10.0	640
J2-11.3 - Data Input 4	L1000	L1000	6.0	384
J3-12.3 - Data Output 4				
B1-14.1 - Sine-Cosine 2	L1600	L	2.5	160
B1-16.2 - Arctangent 2	L1832	L	1.5	96
B3-18.1 - Log 3	L2000	L	1.0	64
B3-17.1 - Exponential 2	L2100	L	2.0	128
		Totals	23.0	1472

Time:

The time required for executing the more conventional instructions range from 133 to 566 milliseconds, while the input floating point data require 666 milliseconds and the print instruction uses 1.86 seconds.

FLOAT & UNFLOAT 1 (Program L3-25.0 R)

This routine performs two functions for operations:

1. It converts a fixed point binary number to standard floating point form as defined in Program H1-24.0. This conversion is defined as "floating" a number.
2. It converts a floating point number as used by Program H1-24.0 to fixed point form. This second type of conversion process is designated as "unfloating" a number.

Storage: Three Tracks.
Time: Float: (150 - 16n) ms, Unfloat: 133 ms. (n designates the number of leading zeros.)

MEMORY SEARCH FOR ADDRESS 1 (Program K3-26.0)

This routine searches the drum for any given instruction (operation and address). If and when the instruction is found, its location is printed in decimal. This search procedure is primarily useful in finding programming errors which cause the computer to:

1. Transfer to an unwanted location, or
2. Store information in an unwanted location.

This memory search does not alter any location searched (locations 0000-6263).

Storage:

One (1) track.

Time:

Four minutes

MATRIX INVERSION 1, Self scaling (Program D1-127.0)

This matrix inversion routine contains the following features:

1. Decimal prescaling of matrix elements.
2. Automatic scaling to adjust for "growth error."
3. Dynamic scaling.
4. Fixed point computation effective at any positive integral "q".
5. Matrices are repositionable in memory.

The method of this matrix inversion routine concerns itself with the prevention of overflow that ordinarily occurs when the maximum sizes of generated numbers are incorrectly estimated. The routine performs tests during the course of the inversion and employs variable scale factors to avoid such overflow. Scaling in this routine is of three types: Decimal prescaling, automatic scaling of "q" when necessary, and dynamic scaling by a scale factor $2^{\emptyset}$. All computation is done in fixed point and is effective at any positive integral "q". The input consists of the exponent (dynamic scale factor) along with the decimally prescaled matrix elements, while the output gives the row permuted identity matrix, the correctly ordered scaled matrix inverse, and the row permuted inverse. The program description for this routine is complete with a clarifying matrix inversion example and "A Program for Operating the MATRIX INVERSION ROUTINE".

Storage:

13-1/4 tracks.

Time:

A "3 x 3" matrix requires approximately one minute for inversion provided there is little scaling. A "6 x 6" requires approximately 5-1/2 minutes.

SEARCH FOR PIVOT ELEMENT (Program D1-27.1)

These two routines function as subsidiaries for other routines in the matrix series; namely, the Matrix Inversion Routine, and the Matrix Multiplication Routine. These routines are utilized in the Inversion routine by conventional "calling sequences" linkages, but in the Multiplication routine the essential elements each appear explicitly in the coding sheets without the use of linkages. The purpose of the "Search for Pivot Element" is the determination of the largest matrix element stored in a given sequence of memory locations, while the purpose of the "Store Data in Two Places" is, as its name implies, the transference of a given set of data from one memory interval to another.

Storage:

One track for each of the two.

MATRIX MULTIPLY 1 (Program D1-227.2)

The matrices multiplied by this routine must satisfy the requirement that the number of columns in the first equals the number of rows in the second. The routine has two characteristics that are the same as two corresponding ones in the Inversion routine; namely:

1. Matrix elements are fractionalized by decimal prescaling.
2. The dynamic scale factor $2^{\emptyset}$ is used, and the exponent "Ø" is entered and printed out at the conclusion in a manner similar to the Inversion routine's method.

Programs D1-27.1 and D1-27.3 are explicitly included as integral parts of this program.

Storage:

Eleven tracks for instructions and constants; track 63 used for temporary storage of matrix.

MATRIX NORMALIZE 1 (Program D1-27.3)

This routine carries out the operations resulting in the presence of a non-zero number in the highest ordered digit of matrix elements. Like the Inversion and Multiplication routines, this program uses the exponent "Ø" and enters it and prints it out in a similar manner.

Storage:

Two tracks of instructions and constants. (One sector, temporary storage).

MATRIX ADD-SUBTRACT 1 (Program D1-327.4)

This routine adds or subtracts two matrices if the number of rows and columns of the two matrices are respectively equal. The characteristics listed under "1" and "2" for D1-227.2 above are also possessed by this routine. The "q" of the matrix elements is unit for both this routine and the Multiplication routine; both for input and output. Since prescaling is used, the final result $2^{\emptyset}$ (C_{ixj}) is multiplied by the post scaling factor 10λ.

INTEGRATION OF DIFFERENTIAL EQUATIONS (Program G2-28.0)

The purpose of this routine is the integration of n first-order equations by S. Gill's modification of the fourth order Runge-Kutta method, with two changes to enable greater accuracy than that provided by certain equations of Gill's paper. The routine itself occupies 200 sectors for storage plus eleven sectors on track 63.

Time:

Roughly (4/3n - 1/5 sec/variable - 4 times the time required to evaluate the derivatives; this is based on 15.2 msec/revolution).

ROYAL McBEE CORPORATION

LGP-30 Input-Output
Keyboard Code

Numerical

	123456
)0	000010
L1	000110
*2	001010
"3	001110
Δ4	010010
%5	010110
$6	011010
π7	011110
Σ8	100010
(9	100110
Ff	101010
Gg	101110
Jj	110010
Kk	110110
Qq	111010
Ww	111110

Commands

	123456
Zz	000001
Bb	000101
Yy	001001
Rr	001101
Ii	010001
Dd	010101
Nn	011001
Mm	011101
Pp	100001
Ee	100101
Uu	101001
Tt	101101
Hh	110001
Cc	110101
Aa	111001
Ss	111101

Controls

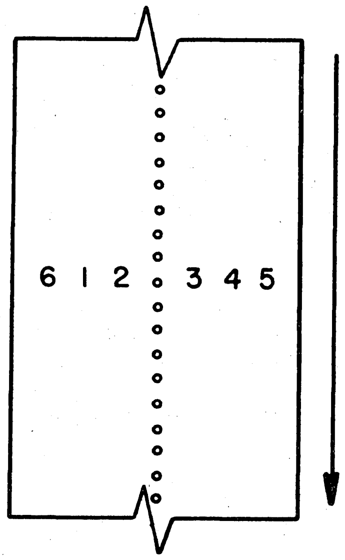
	123456
Lower Case	000100
Upper Case	001000
Color Shift	001100
Car. return	010000
Back space	010100
Tab	011000
Cond. Stop	100000
Start Read	000000
Space	000011
Delete	111111

Signs

= +	001011
_ -	000111

Balance of Keyboard

	123456
: ;	001111
?/	010011
].	010111
[,	011011
Vv	011111
Oo	100011
Xx	100111

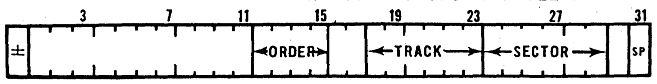


Symbol	Command	Binary	Hex	Dec
Z	Stop	0000	0	0
B	Bring	0001	1	1
Y	Store Add.	0010	2	2
R	Return Add.	0011	3	3
I	Input	0100	4	4
D	Divide	0101	5	5
N	N Multiply	0110	6	6
M	Multiply	0111	7	7
P	Print	1000	8	8
E	Extract	1001	9	9
U	Transfer	1010	F	10
T	Test	1011	G	11
H	Hold	1100	J	12
C	Clear	1101	K	13
A	Add	1110	Q	14
S	Subtract	1111	W	15

Keyboard Code					
)0	02	02	Zz	01	01
Ll	06	06	Bb	05	05
*2	0f	10	Yy	09	09
"3	0q	14	Rr	0k	13
△4	12	18	Ii	11	17
%5	16	22	Dd	15	21
$6	1f	26	Nn	19	25
π7	1q	30	Mm	1k	29
Σ8	22	34	Pp	21	33
(9	26	38	Ee	25	37
Ff	2f	42	Uu	29	41
Gg	2q	46	Tt	2k	45
Jj	32	50	Hh	31	49
Kk	36	54	Cc	35	53
Qq	3f	58	Aa	39	57
Ww	3q	62	Ss	3k	61
Sp.	03	03	LC	04	04
–	07	07	UC	08	08
=+	0g	11	CS	0j	12
:;	0w	15	CR	10	16
?/	13	19	BS	14	20
].	17	23	Tab	18	24
[,	1g	27	Del.	3w	63
Vv	1w	31		20	32
Oo	23	35			
Xx	27	39			

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
32 768	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

00	0
q4	57
j8	50
fj	43
90	36
74	29
58	22
3j	15
20	8
04	1
q8	58
jj	51
g0	44
94	37
78	30
5j	23
40	16
24	9
08	2
qj	59
k0	52
g4	45
98	38
7j	31
60	24
44	17
28	10
0j	3
w0	60
k4	53
g8	46
9j	39
80	32
64	25
48	18
2j	11
10	4
w4	61
k8	54
gj	47
f0	40
84	33
68	26
4j	19
30	12
14	5
w8	62
kj	55
j0	48
f4	41
88	34
6j	27
50	20
34	13
18	6
wj	63
q0	56
j4	49
f8	42
8j	35
70	28
54	21
38	14
1j	7



COMMERCIAL COMPUTER DIVISION

GP GENERAL PRECISION

INFORMATION SYSTEMS GROUP

SC 0001

PRINTED IN U.S.A.