R O Y A L   P R E C I S I O N

Electronic Computer

LGP - 30

<u>PROGRAMMING MANUAL</u>

Royal McBee Corporation

Port Chester, New York

April 1957

CONTENTS

WHAT IS PROGRAMMING?

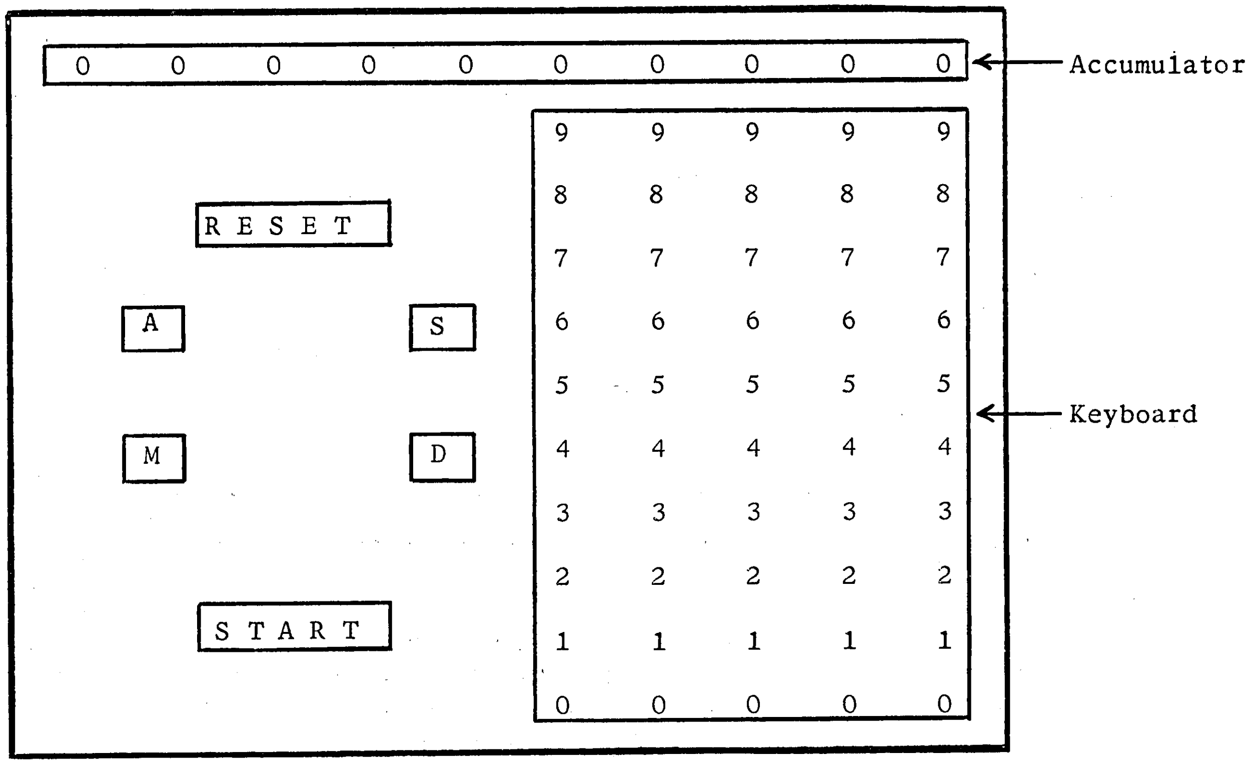
Programming is planning how to solve a problem. No matter what method is used -- pencil and paper, slide rule, adding machine, or computer -- problem solving requires programming. Of course, how one programs depends on the device one uses in problem solving. Programming the Royal Precision LGP-30 is basically simple. Understanding certain problems requires special knowledge, however programming for the LGP-30 does not. Hence this manual is meant for beginners as well as those with experience in stored program computers and describes completely the fundamentals of programming for the LGP-30.

Experienced programmers may find a reading of the summary of orders at the back of the manual sufficient.

A Program for a Desk Calculator.

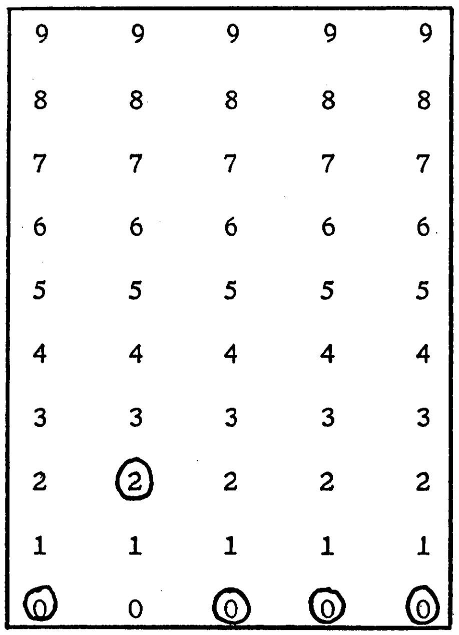
The purpose of a computer is to solve problems. More specifically, however, computers are used to calculate numerical answers to numerical problems. By "calculate" is meant, the use of addition, subtraction, multiplication, and division. To illustrate programming, it is sufficient to illustrate with any machine that calculates; for instance, a desk calculator.

Let us invent the description of a desk calculator with which to illustrate programming.



Desk Calculator

The accumulator register holds the results of calculation at each stage. Depressing the start button after the reset button is depressed puts zeros at all ten of the accumulator digit positions. The first step in adding two numbers, such as perhaps 2 and 4, is to reset the accumulator if any numbers remain in the accumulator from previous calculation and then to put the number 2 in the keyboard. Note that we could depress a 2 in any one of the five positions on the keyboard, and zeros in the others. In this case suppose we depress the number 2 in the column second from the left.



Keyboard with 2

The depressed keys are shown circled. Next, we depress the add button labeled A. Next, when the start button is depressed the number appears in the accumulator.

0	0	0	0	0	0	2	0	0	0

Accumulator with 2

Now to add the number 4, we set 4 in the keyboard. Note that we must put the 4 in at the same position as we put the 2 in, namely, second from the left. The next steps are to depress the add button A, depress the start button, and finally write the result, which is 0000006000.

This procedure would have been good for adding 200 and 400 or .02 and .04 or 2 and 4 time <u>any</u> power of 10. We simply say we have scaled them differently by powers of ten. It is found useful to use a shorthand terminology for such scaling. This shorthand terminology we shall call scale factor and designate by the letter "f". In the above case, we have added 2 and 4 at a scale facter f = -3 since the numbers are shifted 3 places to the left with respect to the decimal point which is located at the right hand end of the accumulator.

If we multiply, scale factors add. For instance, if we multiply 0000002000 on the accumulator by 04000 on the keyboard, the result in the accumulator will be 0008000000 in the accumulator. This operation can be expressed as 2 at f = -3 times 4 at f = -3 equals 8 at f = -6. Now suppose we multiply again by the number 04000 on the keyboard. Note what occurs. The result in the accumulator is 2000000000. In other words, we have lost the 3 of the 32 at f = -9 which should result from the multiply. This is because we have added together scale factors which result in a number too large for the accumulator to hold. We say that the accumulator has <u>overflowed</u>. The smallest value of f at which we can hold 32 is f = -8.

However, if we had $\overset{?}{\text{added}}$ 2 at f = 0, and 4 at f = 0, and had multiplied that result by 4 at f = 0, the result would have been 0000000032 at f = 0 in the accumulator. In general, then, we want to keep our numbers at as high a scale factor as possible in order to avoid the possibility of overflow. The highest scale factor possible for the numbers 2 and 4 is f = 0. Note that the numbers .02 and .04 could have been held at a scale factor f = +2.

Now let us try solving a somewhat less simple problem on our desk calculator, the evaluation of an algebraic expression $a_0x^4 + a_1x^3 + a_2x^2 + a_3x + a_4$.

Suppose we are given values for x and for each of the five coefficients a_0 through a_4. We could then plan our program to get values for x^2, x^3, and x^4, multiply by the coefficients as indicated in order to get the terms of the expression, and finally add the terms to get the answer. We write A for add and M for multiply. If we follow such a plan, our program looks like this:

1	Reset
2	A x
3	M x
4	Write value of x^2
5	M x
6	Write value of x^3
7	M x
8	Write value of x^4
9	M a_0
10	Write value of a_0x^4
11	Reset
12	A x^3
13	M a_1
14	Write value of a_1x^3
15	Reset
16	A x^2
17	M a_2
18	Write value of a_2x^2

calculation notes for <u>storing</u> information

Given:

a_0

a_1

a_2

a_3

a_4

Calculated:

x^2

x^3

x^4

a_0x^4

a_1x^3

a_2x^2

a_3x

final result $= a_0x^4 + a_1x^3 + a_2x^2 + a_3x + a_4$

```
19      Reset
20      A $x$
21      M $a_3$
22      A $a_4$
23      A $a_2x^2$
24      A $a_1x^3$
25      A $a_0x^4$
26      Write value of final result
```

With a little reorganization we could have greatly simplified our program. For instance, if we had looked at the expression in factored form $(((a_0x + a_1)x + a_2)x + a_3)x + a_4$ the program would have looked like this:

```
 1   Reset
 2   A $a_0$
 3   M $x$
 4   A $a_1$
 5   M $x$
 6   A $a_2$
 7   M $x$
 8   A $a_3$
 9   M $x$
10   A $a_4$
11   Write value of final result
```

As we will discuss in detail later, we can program this problem on the LGP-30 with an even simpler set of steps (B, M, A, M, A, M, A, M, A, H) because the LGP-30 provides B which combines reset and add.

The steps in organizing a program such as this deserves emphasis. They are as follows:

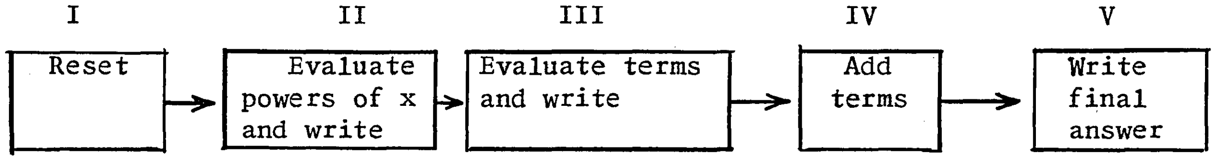
(1). Find a mathematical statement of the problem.

(2). Find the best numerical expression for machine calculation. In the case given we had, to begin with, a numerical expression for machine calculation, but it was not the best.

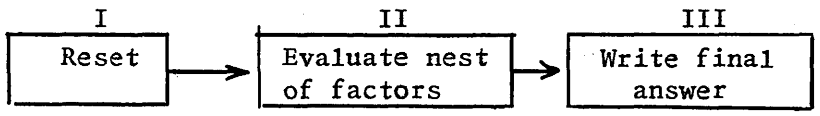
This improvement of a program, by changing the method of expressing it mathematically, is the very simplest illustration of an art known as numerical analysis. The results of our analysis have saved time required for writing intermediate results, saved paper, and program steps.

(3). Write a flow diagram.

In the first case a flow diagram looks as follows:



For the second case a flow diagram looks like this:



(4). Code the problem. The coding of a problem is the actual writing of the instruction required for each program step in the solution. In the case of the desk calculator, an instruction consists of an order part such as "add" and a numerical part such as a_0.

(5). Check the method of programming. Checking can be accomplished by solving the problem with a set of simple values chosen for the variable and the constants. Another method of checking is by programming the problem in two different ways. In this case, we have two methods, which provide checks for each other. Try checking with pencil and paper by using the following values:

$x = \frac{1}{2}$, $a_0 = 32$, $a_1 = 24$, $a_2 = 16$, $a_3 = 10$, and $a_4 = 1$

For simplicity assume all of the numbers are at a scale factor $f = 0$.

It is usually the case that the largest part of programming is that represented by the fourth stage, coding. Hence, the word programming is sometimes used to mean only the fourth stage of this sequence.

What orders did we use in constructing our program? Of course, there were add and multiply. But there are also others. For instance, we used reset and write as orders. These orders are not so much numerical as logical and result more from the nature of the calculator than from the demands of the problem. It will be seen that since the computer to be described in this manual is different from and more powerful than the desk calculator, other orders are useful which have no counterparts in the case of the desk calculator.

What Type of Computer is the LGP-30?

The Royal Precision Electronic Computer LGP-30 is a general purpose electronic digital computer. The phrase "general purpose" is intended to mean that the computer can solve to any required order of accuracy any mathematical problem expressable in numerical or logical terms. However, for any given computer there are always some problems beyond its practical reach because of the length of time required for their solution. By "electronic" is meant simply that the device uses vacuum tubes and germanium diodes. One way of classifying computers is by the terms analog and digital.

An analog computer measures while a digital computer counts. The prototype of the analog machine is the slide rule or the automobile speedometer. A digital computer works on the principle of the abacus or of the desk calculator.

There are other phrases, too, which help to classify the LGP-30.

The LGP-30 is (1) a fixed point machine. In the desk calculator which we described, the decimal point is always understood to be at the right hand end of the keyboard and the accumulator. So also in the LGP-30 there is a fixed location for the decimal point. However, the LGP-30 is (2) a fractional machine. That is, the point is understood to be at the left hand end of the accumulator, rather than at the right hand end. Hence, all numbers must be scaled so that representation in the machine is in the range between -1 and +1.

However, scaling can always be accomplished by a program and need never be the concern of the operator or programmer. Furthermore, the LGP-30 is (3) internally binary. Instead of a digit from 0 to 9 in each position of the accumulator, it is only possible for either a 0 or 1 to be in each digit position of the accumulator. One common device, which is both fractional and binary, is the ruler. Suppose, for instance, we were to measure 13/16th of an inch. We would note that we had a $\frac{1}{2}$ an inch, a $\frac{1}{4}$ of an inch, and a 1/16 of an inch. A simple representation of this process of counting halves, quarters, eighths, etc., is as follows:

1/2's	1/4's	1/8's	1/16's	1/32's	etc.
.1	1	0	1	0	0 ...

or

.1101	=	13/16
BINARY		DECIMAL FRACTION

Just as scaling can be handled by a program so also can conversion from binary to decimal so that the LGP-30 can be used as easily as any decimal computer.

The LGP-30 is (4) a stored program computer. It is easy now to see that a problem requires not only the numbers to be operated on but also a set of instructions describing the sequence of operations; that is, a program. In the case of a stored program computer, not only can the numbers be stored, but <u>the instructions can also be stored</u>.

Functional Components of the LGP-30.

There are four basic functional groupings of the components of the Royal Precision Electronic Computer LGP-30 which are necessary for problem solving. These functional groupings are: The accumulator, the memory, the input-output system, and the control system. Each of these have analogs in the case of the desk calculator. For instance, the memory of the LGP-30 is equivalent in the desk calculator case to the paper required for storing initial data, intermediate results, and final results. An essential difference from the desk calculator, however, is that in the case of the LGP-30 the program itself is also stored in the memory.

The accumulator of the LGP-30 is entirely similar in function to the accumulator in the desk calculator.

In the case of the desk calculator, getting the results from our system is simply a matter of reading what is in the accumulator, or what we have written on paper. In the case of a stored program computer, however, the numbers to be operated on and the results are stored in the memory. In the LGP-30 it is the input-output system which enables us to get numbers in and results out.

The control function is manual in the case of the desk calculator but is automatic in a stored program computer. For instance, control in the desk calculator requires pressing the start button to execute each instruction, but in the stored program computer the control function provides for executing the program either one step at a time by pressing the start button for each instruction or all at once by pressing the start button just once for the entire program.

Why a Stored Program Computer?

Since the desk calculator is a useful computer, the question arises, "What is the need for a stored program computer?" The answer lies in the speed of computation. For instance, the stored program computer LGP-30 can execute over 400 additions per second, whereas the desk calculator can only execute approximately one addition per second, not counting the time required to enter the numbers into the keyboard. A table at the back of the manual shows the important physical and operational specifications of the Royal Precision electronic computer LGP-30.

Besides a comparison of the LGP-30 with desk calculators, a comparison with other general purpose stored program computers is of interest. The LGP-30 occupies less floor space, requires less power, has a simpler list of instruction types, has fewer components, and costs less than any other general purpose stored program computer now in use. The LGP-30 has as large a memory, requires as few operators, is as fast, and is as easy to program as any other general purpose stored program computer now in its price range.

We will cover in order:

Memory and recirculating registers
Building a program
Number representation
Input-output and control

MEMORY AND RECIRCULATING REGISTERS

Memory Drum. The heart of the memory section of the Royal Precision electronic computer LGP-30 is the magnetic drum shown below. The drum is a metal cylinder 6.5 inches in diameter and 7 inches long. It is coated with material which can be magnetized, and it rotates at approximately 3700 revolutions per minute.

Track. Read-record heads, which are for magnetizing and for detecting magnetization on the drum, are mounted in a frame around the drum. Reading and recording on the drum are done in a manner somewhat different from that used to record on tape in tape recorders. The read-record heads are spaced along the axis of the drum so that each one can record and read spots in a circle around the drum as the drum rotates. There are 64 such circles and they are called tracks.

The use of 64 tracks and 64 read-record heads means that any given portion of the drum is available to a read-record head at least 64 times faster than if the memory consisted of a tape governed by one read-record head.

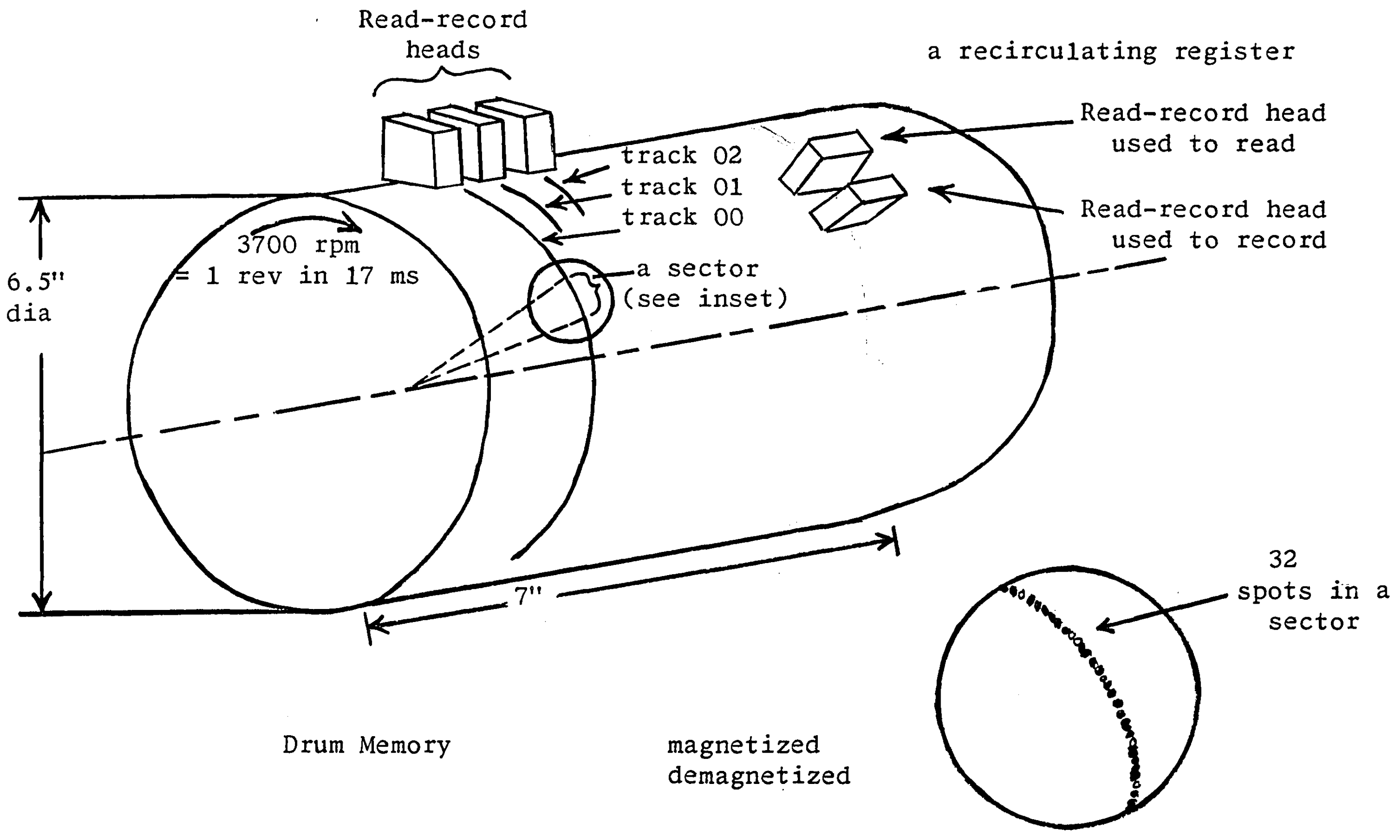
Sector. In each track there are 64 groups of spots. Each group occupies a sector. Each sector consists of 31 spots each of which can be magnetized or de-magnetized and a 32nd spot, called the spacer, which is always unaffected by recording and is never examined by reading.

Locations and Addresses. There are 64 tracks and 64 sectors per track in the memory. Hence every location in memory can be identified by a track and sector number and there are 64 x 64 = 4096 such locations. Location numbering is generally by track and sector, tracks being numbered 00 through 63, sectors 00 through 63, and locations 0000 through 6363. Note, however, that although there are 4096 locations, a location number such as 2089 is impossible with this numbering system since sectors number only through 63. A location number is called an address.

Access Time. Each sector of a given track is accessible for reading or recording by the head associated with the track of that location once per revolution. Since the drum rotates once every 17 milliseconds, each location is accessible once every 17 milliseconds.

Recirculating Registers. In addition to the 64 tracks of memory on the drum, there are three tracks each of which contains a recirculating register. Each of these recirculating registers is one sector in length. As the recirculating register passes under a read head, a record head continuously records the information read, back into the drum at a distance of one sector length from the read head. The advantage of the recirculating register is a reduction of sector access time to the equivalent of one sector length or about .26 milliseconds, whereas, a location in memory has an access time equivalent to one track length or 17 milliseconds.

The three recirculating registers are the accumulator, the instruction register, and the counter register. The function of the accumulator should be clear from the discussion of the desk calculator. The functions of the instruction register and the counter register are covered later.



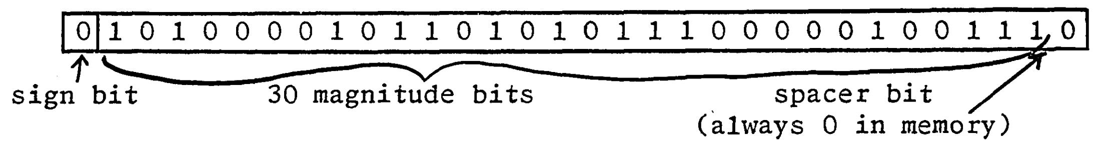
Bit. Each of the 31 spots in each sector on the drum can be in either of two states: not magnetized or magnetized. These two states may be interpreted as corresponding to 0 and 1 which are the two digits of the binary number system just as 0 through 9 are the digits of the decimal system.

The usefulness of a drum type computer depends on this correspondence between magnetized spots and binary digits. It can be seen, then, that information can be stored on the drum in terms of binary numbers. There is a binary digit, either 0 or 1, corresponding to the state of each spot on the drum. The phrase "binary digit" is generally contracted to the word "bit".

The term "bit" is sometimes used to mean the spot on the drum as well as the binary digit which the spot represents. Also the word bit is sometimes applied as meaning the digit 1 as opposed to the digit 0. Although the LGP-30 operates in the binary number system, a program to convert from decimal to binary can be stored on the drum so that the LGP-30 is as convenient as any decimal computer.

Number Words. The information stored in a memory location in terms of magnetized or de-magnetized spots is called a word. A word can either be a number or, since this is a stored program computer, an instruction. We will discuss here what a word looks like once it is in the memory and defer until later how to get words in and out of the LGP-30.

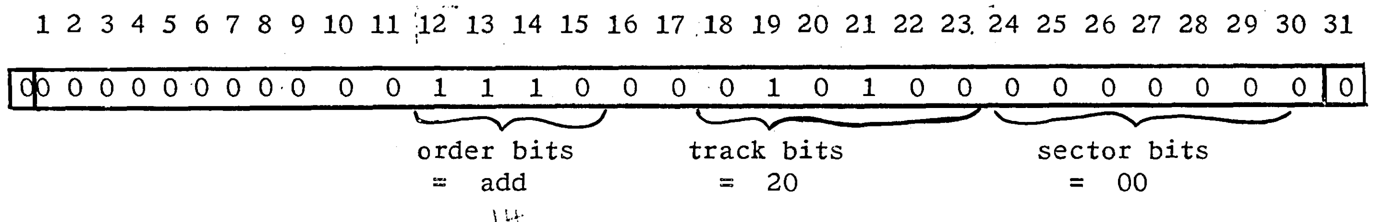
If a word is a number, it consists of 30 bits to represent magnitude, one bit related to sign representation, and a spacer bit.



If a number is positive, the sign bit will be 0. If the number is negative the sign bit will be 1.

In the desk calculator example, the accumulator is ten decimal digits long, and hence can be used to represent numbers up to 10^{10} in magnitude. Since a word in the LGP-30 is thirty binary bits long, a number up to a magnitude of 2^{30} can be represented in each word. Note that 100 in the number system base 2 is 2^2 just as 100 in the number system base 10 is 10^2. Hence, 1 followed by 30 0's in the binary system is equivalent to 2^{30}. Since 2^{30} is approximately equal to 10^9 in the decimal system, magnitudes of nine significant decimal digits can be represented in the LGP-30. A discussion of binary arithmetic, representation, and scaling is covered later.

Instruction Words. An instruction word consists of two parts, an order part and an address part. For instance, the order part of the instruction might represent the operation add. The address part of the instruction represents, by track and sector, the location in memory of some number. For instance, add 2000 is interpreted as meaning add the number located in track 20, sector 00 to the contents of the accumulator and store the result in the accumulator. Note a major difference from the instructions discussed earlier where add 2000 meant add the number 2000 not a number in location 2000.



instruction word = a 2000

As indicated, bit positions 12 through 15 in a word are used to represent the order. Since 4 bits are used to represent the order, there are 2^4 or 16 possible orders in the LGP-30. For instance, 1 1 1 0 in binary is interpreted as "add" if located in bit positions 12 through 15 of an instruction word. Bit positions 18 through 23 are used to represent track and bit positions 24 through 29 are used to represent sector.

Since there are six bit positions allowed for track, and six bit positions allowed for sector, 2^6 or 64 tracks and 2^6 or 64 sectors can be designated by the address part of an instruction word.

The computer has no way of knowing by examining the word inself, whether a word is intended as a number or an instruction. Under most circumstances it is the numbers which are operated on and the instructions which are followed as instruction. The LGP-30 has the ability to operate on instructions as numbers. This ability is not only valuable but a feature which makes for the great power of stored program computers. However, it is always to be avoided to use numbers as instructions. Since an instruction word is indistinguishable from a number word, some means must be provided to avoid using numbers as instructions. How this is accomplished is described under "BUILDING A PROGRAM". Also, how we convert from a 2000 to the binary representation shown is discussed later.

Summary. In summary, the essential element with which we deal in the computer is the word, and it may be either a number or an instruction. The next section discusses the orders of the LGP-30 and the effect they have on number representations in the computer.

BUILDING A PROGRAM

Instruction Execution. We have learned that an instruction consists of two parts, an order part such as the letter a to designate "add" and an address part such as 2000 to designate a memory location.

For instance, the instruction a 2000 means add the number in location 2000 to the number in the accumulator and put the result in the accumulator. Let us see what the LGP-30 does to execute this instruction.

There are four phases to instruction execution. First let us suppose that a 2000 is in location 1000 and that the LGP-30 is ready to execute the instruction in location 1000. The computer is ready to execute the instruction in location 1000 when 1000 is in the counter register. Hence, we are assuming that 1000 is in the counter register.

The first phase of execution is "search". As the drum rotates,the LGP-30 searches for the address given by the counter register, in this case 1000.

The second phase is "transfer" As location 1000 passes under the read-record head of track 10, the LGP-30 places a 2000 in the instruction register, but at the same time leaves a 2000 undistrubed in location 1000.

The third phase is "search for operand". The LGP-30 searches for location 2000.

The fourth phase is "completion". The LGP-30 picks up the operand as location 2000 passes under the read-record head of track 20 and completes the operation according to the order part of the instruction. In this case, it adds the operand to the contents of the accumulator and places the result in the accumulator.

Before execution of an instruction is complete, the LGP-30 adds 1 to the contents of the counter register. Therefore, when execution of the instructions in 1000 is complete, the LGP-30 begins execution of the instruction in 1001. Hence, the LGP-30 executes instructions in sequence according to memory location. When the last instruction in a track is executed (for instance, the instruction in 2063), the instruction in sector 00 of the next track is executed because 2100 appears in the counter register.

Now we see how the LGP-30 executes a sequence of instructions. How it begins executing a sequence and how it stops is covered in the following discussion of the orders. Later we will discuss what we have to do to store instructions in memory in the first place and how a 2000 is converted into binary representation. For now we are going to discuss what orders and instructions do and how we can use them to build a program.

What orders do we need to evaluate the expression $(((a_0x + a_1)x + a_2)x + a_3) x + a_4$ which we introduced earlier? We said that four LGP-30 orders designated by b, m, a,and h(called bring, multiply, add, and hold) were sufficient. To make from these orders the instructions necessary to evaluate the given expression, we need the address of some memory locations.

Hence let us assume that x, a_0, a_1, a_2, a_3, and a_4 are in locations 2000 through 2005 respectively.

Bring From Memory. Now we can write an instruction such as b 2001. This means reset the accumulator to zero and add the number in 2001 (in this case a_0) to the contents of the accumulator. Note the improvement over the desk calculator which required two orders, reset and add, instead of the one order bring of the LGP-30.

M Multiply. The instruction m 2000 means multiply the contents of the accumulator by the number in location 2000 and put the most significant half of the resulting product in the accumulator (in this case, a_0x). In general, the multiplication of two numbers results in a product which has as many digits as the sum of the number of digits in the multiplier and the multiplicand. For instance, if we multiply .20 by .10, we get .0200. Hence, if we interpret these operands as $x = 2$ at a scale factor $f = 1$ and $a_0 = 1$ at a scale factor $f = 1$ and if we retain only the significant half of the product, the result is $a_0x = 2$ at a scale factor $f \doteq 2$. Therefore by proper scaling, significance may be retained with the M multiply in keeping only the most significant half of the product. More about scaling later.

Add. The instruction a 2002 means add the number in location 2002 to the contents of the accumulator and keep the result in the accumulator (in this case, $a_0x + a_1$).

Hold & Store. The instruction h 2006 means store the contents of the accumulator in location 2006 and retain it also in the accumulator.

Now supposing we locate our program beginning in location 1000, we are equipped to write the following program notes.

Location	Instruction or Number	Operand	Result
1000	b 2001	a_0	a_0
1001	m 2000	x	a_0x
1002	a 2002	a_1	$a_0x + a_1$
1003	m 2000	x	$(a_0x + a_1)x$
1004	a 2003	a_2	$(a_0x + a_1)x + a_2$
1005	m 2000	x	$((a_0x + a_1)x + a_2)x$
1006	a 2004	a_3	$((a_0x + a_1)x + a_2)x + a_3$

Location	Instruction or Number	Operand	Result
1007	m 2000	x	$(((a_0x + a_1)x + a_2)x + a_3)x$
1008	a 2005	a_4	$(((a_0x + a_1)x + a_2)x + a_3)x + a_4$
1009	h 2006	Answer	Store answer in 2006
2000	x		
2001	a_0		
2002	a_1		
2003	a_2		
2004	a_3		
2005	a_4		
2006	(answer)		

We have now provided instructions which perform all of the functions performed by the desk calculator in the solution of this problem: reset, add, multiply, and write. We have written the answer into a memory location, however, and do not know what it is. Later we will discuss how to program so that our answer is printed out on the typewriter.

Stop. In the desk calculator, computation requires pressing the start button to execute each instruction. In the case of the LGP-30, however, pressing the start button may initiate the entire sequence of instructions. Hence, we need some method of instructing the computer to stop computation when we have accomplished what we desire.

The LGP-30 provides a stop order designated by the letter z. Up to this point every instruction has implied an address. Actually, however, computation does not stop to any memory location; it simply stops. Hence, a stop instruction usually has the same effect regardless of the numbers in the address portion of the instruction. It is customary to write a stop instruction as z 0000.

Looping. The repetition of the group of two instructions, multiply and add, suggests a generalization of the program we have written. Perhaps we could in some way use the same multiply and add instructions repeatedly instead of writing a sequence of several pairs of multiply and add instructions. We require two additional types of functions to accomplish this:

(1). an instruction to transfer back to the multiply and add instructions and

(2). a means of modifying the address in the add instruction.

Since we have located the coefficients a_0 through a_4 in sequential memory locations, it will be useful to add 1 to the address portion of the add instruction before transferring back to the multiply and add instructions. Let us consider program notes for the same expression as above which look as follows:

Location	Instruction or Number	Operand	Notes or Result
1000	b 2000	working storage	Initially zero
1001	m 2004	x	
1002	a(2005)	a_n	Initially a_0
1003	h 2000		*
1004	b 1002	a( )	}
1005	a 2001	1	} modify add
1006	h 1002	a( )	} instruction
1007	u 1000		
2000	working storage		Initially zero
2001	1 at a p of 29, or dec. 4		
2002	not used		
2003	not used		
2004	x		
2005	a_0		
2006	a_1		
2007	a_2		
2008	a_3		
2009	a_4		

no "stop"

* Intermediate and final answers into memory location 2000.

Note that after executing the add instruction for the first time we write into memory our first intermediate result, a_0. The following three instructions bring, add, and hold change the add instruction from a 2005 to a 2006. Hence, we note that address modification has not required any new type of order.

✓ Unconditional transfer. However, transfer back to the multiply instruction does require a new order. For this the LGP-30 provides an unconditional transfer order designated by the letter u. An instruction such as u 1000 means execute next the instruction in location 1000. Each repetition of the sequence preceding the u instruction is called an iteration. It should be noted that a u instruction does not affect the contents of the accumulator. So far the only other instruction of this nature is the stop instruction.

Now we have an answer to the question "How do we start and stop a sequence of instruction?" A u instruction transfers computation to the beginning of a sequence and a z instruction stops computation at the end of a sequence. Later we will describe how to use the typewriter and control console to execute the proper u instruction to begin executing a program once it is stored in memory.

We also have an answer to the question "How do we avoid executing numbers as instructions?" Since instructions are executed in sequence and since we have means of starting and stopping any sequence, we can avoid any sequence of memory locations where numbers are stored.

✓ Working storage. Besides transfer and address modification, there are two other ideas in our new program worth noting. First, the initial bring instruction when first executed does not bring a_0 but brings the contents of 2000 which is zero. The reason is this: after executing the multiply and add pair, we must store the result each time because we need to make use of the accumulator in modifying the address of the add instruction. Having stored the intermediate result each time we have to bring it again after we transfer to the beginning. Since we want to avoid bringing a_0 after the first time and need to bring the intermediate result, we provide a special memory location for storing intermediate results and find another method for bringing a_0 initially. Since the add instruction initially has the address of a_0 and the contents of working storage is initially zero, the effect of the first execution of the add instruction is to bring a_0.

A second idea we need to mention is that the 1 we add to the instruction in location 1002 must increase the sector by one each time. Hence, according to the diagram of an instruction word which appeared earlier, the number in location 2001 must have a one in position 29 but otherwise have all zeroes.

Locations 2002 and 2003 we did not use so that they would be available for certain modifications we will later make to the program.

✓ The Counter. You probably notice several things wrong with the program as it stands. The principal fault is that there is no way of controlling when it stops. After the instruction in 1002 becomes a 2009, it becomes a 2010 and so on indefinitely. Numbers in locations following 2009 are added into the result, which we do not want. Two additional orders are helpful in controlling the number of iterations executed. These orders are "subtract" and "test".

Subtract. The instruction s 2003 means subtract the number in location 2003 from the contents of the accumulator and keep the result in the accumulator.

✓ Test. The instruction t 1000 means that if the number in the accumulator is negative, transfer to location 1000 and if the number in the accumulator is zero or positive, execute the instruction following t 1000.

Let us consider the following sequence of instructions:

Location	Instruction or Number	Operand	Results or Notes
1007	b 2002	counter	
1008	a 2001	1	augment counter
1009	h 2002	counter	store augmented counter
1010	s 2003	5	flag
1011	t 1000		tests number of iterations
1012	z 0000		stops program
2001	1 at 29 or dec. "4"		
2002	working storage		counter initially zero
2003	5		flag

The first iteration brings zero, adds one, holds one, subtracts 5 to leave -4 in the accumulator, and transfers.. The second iteration leaves -3 in the accumulator before testing and so on until the last iteration leaves zero and the result of the test instruction is to go to the stop instruction in 1012. In this case, the sequence of instructions not shown (from 1000 to 1007) is executed 5 times before testing out.

Since the preceding program required modification of the add instruction in 1002 by one for each iteration, we might use the add instruction itself as the counter. Then our program notes look as follows:

Location	Instruction or Number	Operand	Result or Notes
1004	b 1002	a( ) as counter	
1005	a 2001	1 at 29	augment counter
1006	h 1002	a( )	store augmented counter
1007	s 2003	a 2010	flag
1008	t 1000		tests number of iterations
1009	z 0000		stops program
2001	1 at 29		
2002	working storage		counter initially a 2005
2003	a 2010		flag

The question usually arises, "Given a certain initial value of counter and augmenting by 1 sector each time, how do we determine what to have for a flag?" Note that the add instruction in 1002 is augmented after execution and before testing. Hence, it is augmented after execution the last time and the flag should have an address larger by one than the address of the last add instruction to be executed.

This last sequence for controlling iterations permits us to execute the preceding multiply-add pair exactly 5 times, which is the number of times required since we must add in the five coefficients a_0 through a_4.

✓ Initializing. There are still faults with the program. If we execute the computation a second time, we obtain a different answer. For one thing, when computation begins the second time, the contents of the working storage location 2000 contains the final answer and not zero. For another, the counter and initial add instruction is a 2010 not a 2005. We must provide instructions, to precede those we have written, which set correct initial values in both of these locations. If we store the instruction a 2005 in 2002 and a zero in 2010, the following four instructions are sufficient to properly initialize our program.

```
b 2002
h 1002
b 2010
h 2000
```

Clear and Store. The LGP-30 provides a clear order which can make this initialization even simpler. The instruction c 1002 means replace the contents of memory location 1002 with the contents of the accumulator and replace the contents of the accumulator with zero. Now the initializing instructions can be reduced to:

```
b 2002
c 1002
h 2000
```

The use of the clear instruction reduces the number of instructions by one and eliminates the need for storing a zero because the use of a clear instead of a hold to store the initial add instruction leaves a zero accumulator.

Subroutines. A subroutine is a program which computes a frequently needed function such as the square root or the printing out of a number in the accumulator. The word "routine" simply means "program" and the prefix "sub-" simply reflects the fact that the evaluation of a function such as the square root is often needed as a subordinate part of a larger program. Subroutines are frequently retained in known locations in memory so that they may easily be used when needed.

In the case of our present program we have need of a print-out subroutine so that the final result stored in location 2000 can be printed out. Suppose we locate the printout subroutine in locations 3000 through 3050. We can easily transfer to the subroutine by executing a u 3000 instruction. The question arises, however, "How can we easily return for more computation to the sequence of locations where we have stored the program of which the subroutine is a part?"

Return address. The answer is provided by the return address order. If the instruction r 3050 is located in 1013, it stores the address 1015 in location 3050.

If the instruction <u>u 0000</u> is located in 3050, it becomes <u>u 1015</u>. Hence, if we have

1012	b 2000
1013	r 3050
1014	u 3000
1015	z 0000

we bring the answer to the accumulator, set up a return transfer instruction, transfer to the beginning of the printout subroutine in 3000, execute the printout routine, transfer back to 1015, and stop. Refer to the LGP-30 Subroutine Manual to see how various subroutines are programmed.

<u>Final Program.</u> Now we can write a program to evaluate $(((a_0x + a_1)\ x + a_2)x + a_3)\ x + a_4$ which incorporates all the devices we have learned to this point.

	Location	Instruction or Number	Operand	Result or Notes
set	1000	b 2002	a 2005	initial add instruction
	1001	c 1005		
	1002	h 2000	zero	initializing working storage
compute	1003	b 2000	working storage	
	1004	m 2004	x	
	1005	a(2005)	a_n	
	1006	h 2000	working storage	intermediate and final results
modify	1007	b 1005	a(2005 + n)	
	1008	a 2001	1 at 29	
	1009	h 1005	a(2005 + n+1)	
test	1010	s 2003	a 2010	flag or terminating constant
	1011	t 1003		
print and stop	1012	b 2000	final result	
	1013	r 3050		
	1014	u 3000	print routine	
	1015	z 0000		stop
	2000	working storage		
	2001	1 at 29		
	2002	a 2005		
	2003	a 2010 $\approx K_T$		
	2004	x		
	2005	a_0		
	2006	a_1		
	2007	a_2		
	2008	a_3		
	2009	a_4		

Aside from the four instructions required for printout, this program is two steps longer than the original program consisting of b, m, a, m, a, m, a, m, a, h instructions and requires the use of three additional storage locations.

Nevertheless, we have gained something, for we now have a program capable of evaluating the more general expression

$$(\ .\ \ .\ \ .(a_0x + a_1)x + \ .\ \ .\ \ .a_{n-1})x + a_n.$$

All we need to supply each time are a value for x and the coefficients a_0 through a_n and a value for the flag equal to a(2006 + n). Note that, although more storage space is required for more coefficients, the number of instructions in our program is constant regardless of the size of n. That we can achieve such a program shows the great power of the LGP-30 and of stored program computers generally.

Although we have been able to build a respectable program, there are several orders provided by the LGP-30 which we have not used.

Store address. Suppose we have a problem in which we wish to square all the numbers stored in track 20 and store each square in the location formerly occupied by the number. We might write a program to appear as follows:

Location	Instruction	Operand	Notes
1000	b(2000)	x	
1001	m(2000)	x	x^2
1002	h(2000)	x^2	
1003	b 1000	b(2000+n)	
1004	a 1015	1 at 29	
1005	h 1000	b(2001+n)	
1006	b 1001	m(2000+n)	
1007	a 1015	1 at 29	
1008	h 1001	m(2001+n)	
1009	b 1002	h(2000+n)	
1010	a 1015	1 at 29	
1011	h 1002	h(2001+n)	
1012	s 1016	h 2100	flag
1013	t 1000		loop
1014	z 0000		stop
1015	1 at 29		augmenter
1016	h 2100		flag

This program can be greatly simplified by the use of the store address order. The instruction y 1000 means replace the contents of the address portion of the word in location 1000 with the contents of the address portion of the word in the accumulator. The contents of the accumulator is unaffected. Hence we can rewrite the program as follows:

Location	Instruction or number	Operand	Result or notes
1000	b(2000)	x	
1001	m(2000)	x	x^2
1002	h(2000)	x^2	
1003	b 1000	b(2000 + n)	
1004	a 1011	1 at 29	

Location	Instruction or Number	Operand	Result or Notes
1005	h 1000		Counter & augmented instruction
1006	y 1001		
1007	y 1002		
1008	s 1012	b 2100	flag
1009	t 1000		
1010	z 0000		stop
1011	1 at 29		
1012	b 2100	flag	

We have been able to shorten the program by 4 steps because each y instruction takes the place of the b, a, h sequence. A y instruction can put the same address into instructions which have different orders.

Extract. More than one kind of data may be stored in a given word. For instance, a calendar date consists of three types of data: month, day and year. A word with these three types of data might look this way.

$$\boxed{0}\boxed{\underbrace{1\;1\;0\;0}_{\text{month}}\;\underbrace{0\;0\;1\;1\;1}_{\text{day}}\;\underbrace{1\;1\;1\;1\;0\;0\;1\;0\;1}_{\text{year}}\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0}$$

Sometimes it is desirable to deal with only one of the three pieces of data. The extract order makes it possible to separate different data stored in one word. The instruction e 2000 means: put zeroes in the word in the accumulator wherever there are zeroes in the word in location 2000 but otherwise leave the word in the accumulator unchanged. For instance, if the above data word is in the accumulator and in location 2000 is the word

$$\boxed{1}\boxed{1\;1\;1\;1\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0}$$

then the result in the accumulator is the following word which contains only the month part of the date.

$$\boxed{0}\boxed{\underbrace{1\;1\;0\;0}_{\text{month}}\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0}$$

The word in location 2000 is called the extract mask. It is possible with another extract instruction and mask to retain the day and not the month or the year, or to retain any part of any given word in the accumulator. The extract order achieves its result by multiplying bits in corresponding positions of the extract mask and the accumulator. That is, if there is a one in position 29 of both words, there is a one in position 29 of the result; otherwise, there is a zero.

N multiply. The instruction n 2000 means multiply the contents of the accumulator by the contents of location 2000 and retain the least significant half of the product in the accumulator. The N multiply and the divide order which follows are both discussed further in connection with binary representation.

Divide. The instruction *d 2000* means divide the contents of the accumulator by the contents of location 2000 and retain the rounded quotient in the accumulator.

Print and Input. Both the print and input orders are discussed under input output and control.

Summary. There are 16 orders available for constructing instructions in the LGP-30. An instruction contains an order part such as the letter *a* for add and an address part such as 2000. It must be emphasized that 2000 in the instruction *a 2000* is the *location* of a number stored in memory and *not* the number itself. Further properties of the m, n, d, a, and s orders are discussed in connection with binary representation, but a summary of the properties of each order can be found all in one place at the back of the manual.

NUMBER REPRESENTATION

Binary Numbers. Just as a number system can be developed using the ten digits 0 through 9, so also a number system can be developed using only the two digits 0 and 1. This number system is called the binary system. In counting with the decimal system, when the digits 0 through 9 have been used in the low order position, a one is placed to the left of the low order position and counting continues with 10, 11, 12, etc. In the counting with the binary system we also first use all the digits in the low order position and then place a one to the left of the low order position so that counting goes 0, 1, 10, 11, etc. The binary numbers equivalent to the decimal digits are as follows:

Decimal digit	Binary number
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

Number representation can be looked on as an efficient method of counting. That is, we need just the ten digits, not as many digits as the magnitude of number we wish to represent, because the digit to the left of the low order position represents the number of 10's. For instance, the number 1,234 is

$$(\underline{1} \times 10^3) + (\underline{2} \times 10^2) + (\underline{3} \times 10^1) + (\underline{4} \times 10^0)$$

similarly in the binary system the number 1101 means

$$(\underline{1} \times 2^3) + (\underline{1} \times 2^2) + (\underline{0} \times 2^1) + (\underline{1} \times 2^0)$$

(where the digits are decimal).

Sometimes it is useful to use a subscript to identify what number system a given expression is in, especially where different number systems use some of the same symbols for digits. For instance, $1101_2 = 13_{10}$; that is, 1101 in binary is equal to 13 in decimal not one thousand one hundred one. It is worthwhile pointing out that

$$10_{10} = 10^1 \qquad 10_2 = 2_{10}$$

$$100_{10} = 10^2 \qquad 100_2 = 4_{10}$$

$$1000_{10} = 10^3 \qquad 1000_2 = 8_{10}$$

Binary Arithmetic. Arithmetic in the binary system is similar to arithmetic in the decimal system.

a. The rules for addition are

1. 0 + 0 = 0
2. 1 + 0 = 1
3. 1 + 1 = 10 (0 with 1 carried)

As an example the sum of the two numbers 1011001 and 1001010 is

```
Carries 1 0 1 1 0 0 0
          1 0 1 1 0 0 1
          1 0 0 1 0 1 0
        ---------------
        1 0 1 0 0 0 1 1
```

b. The rules for subtraction are:

1. 0 - 0 = 0
2. 1 - 1 = 0
3. 1 - 0 = 1
4. 0 - 1 = 1 (with one borrowed)

As an example the difference of the two numbers 11001011 and 1010110 is

```
Borrows       1 1 0 1
            1 1 0 0 1 0 1 1
            - 1 0 1 0 1 1 0
            ---------------
Difference0 1 1 1 0 1 0 1
```

c. The multiplication table for binary digits is

1. 0 x 0 = 0
2. 1 x 0 = 0
3. 1 x 1 = 1

The rules for multiplication and division in longhand are exactly the same as the rules in the decimal system. For example the multiplication of 1.011 by 0.110 is

```
         1. 0 1 1
         0. 1 1 0
         --------
         0  0 0 0
       1 0  1 1
     1 0 1  1
   0 0 0 0
  ---------------
  1. 0 0 0  0 1 0
```

To facilitate the handling of the binary point in binary arithmetic, most computers are constructed so that the binary point is fixed either to the left of the most significant digit or to the right of the least significant digit.

The LGP-30 is designed to operate on numbers with the binary point left of the most significant digit. Numbers held in the computer are then represented as fractional quantities with the range of magnitude from +1 to -1.

Number Conversion and Scaling. Let us consider how we might represent the decimal number 19 in the LGP-30. First of all, let us convert 19 into binary. To do this we first find the largest number representable by a power of 2 which is equal to or smaller than 19. The number in this case is 2^4 or 16, or in binary, 10000. Conversion is easier to visualize if we represent the binary number as

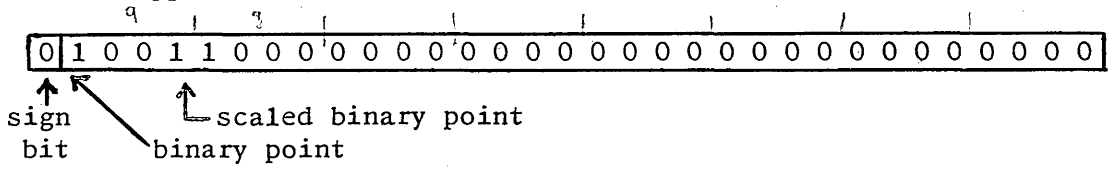
$$(\underline{1} \times 2^4) + (\underline{0} \times 2^3) + (\underline{0} \times 2^2) + (\underline{0} \times 2^1) + (\underline{0} \times 2^0)$$

With respect to the remainder 19 - 16 = 3, 2^1 is the largest power of 2 equal to or less than 3. Now we can write a binary representation of 18 based on the fact that it is the sum of numbers which are integral powers of 2. That is, $18 = 2^4 + 2^1 = 16 + 2$, which in binary is 10010 or

$$(\underline{1} \times 2^4) + (\underline{0} \times 2^3) + (\underline{0} \times 2^2) + (\underline{1} \times 2^1) + (\underline{0} \times 2^0)$$

By this time it should be easy to see that 19 in decimal is equivalent to 10011 in binary.

Now that we have converted the number we chose into binary, we need to represent it as a fraction since the LGP-30 is a fractional machine. We can do this by shifting the number far enough right with respect to the binary point. A shift of 5 places gives .10011 for the representation. Note that 5 is the smallest number of places we could shift right and still have a fraction. We keep track of shifts by the scale factor q. In this case, we say we have 19 at q = 5. In a memory location the number word would appear as



q = 5

If the number we wish to represent is small enough, it is possible to represent it at a negative q. For instance, 1/8 decimal is equivalent to 0.001 in binary and 0.1 in binary is 1/8 at q = -2. In this case we moved the number two places to the left with respect to the binary point.

In spite of the fact we have given some attention to converting a number here, the LGP-30 can be programmed to do all such conversion so that the operator may use it as a decimal computer.

M multiplication. Suppose we consider multiplying 2 and 3. In binary 2 is 10 and 3 is 11. Each can be represented in the LGP-30 at a binary scale factor q = 2. Hence 2 at q = 2 is .10 and 3 at q = 2 is .11.

When we multiply, scale factors add. The result, then, is 6 at q = 4, which in binary is .0110. The appearance of the operands and the result of an M multiply in the LGP-30 are as follows:
In the accumulator

```
|0|1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
```

In memory

```
|0|1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
```

Result in the accumulator

```
|0|0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
```

Addition. If we add 2 and 3 each at q = 2, the result is 5 at q = 2 which in binary is 1.01. Since this number is not fractional, it cannot be held in the LGP-30. Hence, although a q = 2 was sufficient for multiplication of 2 and 3, 3 is the minimum q for adding 2 and 3. Numbers can be added only if they are at the same q.

Overflow. When addition results in a number too large for the LGP-30, we say computation overflows. The result is that the machine stops computing at the add order. Overflow can occur as the result of substraction when numbers of opposite signs are the operands. Overflow due to division can also occur. Multiplication can never result in overflow since the multiplication of fractions can never result in a number as large as 1.

Truncation. An M multiply, however, has another characteristic worth noting. Suppose we multiply 3 at q = 30 by 2 at q 2. The result should be 6 at q = 32.

In the accumulator

```
|0|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0|
```

In memory

```
|0|1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
```

Result in the accumulator

```
|0|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0|
```

The result is 4 at q = 32 instead of 6 at q = 32. In other words, part of the result has been lost because the word length is 30 bits. This type of error is called truncation error and can be minimized by carrying operands for multiplication at as small a q as possible.

In general then, it is desirable to carry numbers at as high a q as possible to avoid overflow and at as small a q as possible to avoid truncation errors.

Division. Suppose we divide 3 at q = 2 by 2 at q = 2. The result is 1.5 at q = 0 since the q of a quotient is the q of the dividend minus the q of the divisor. This result in binary is 1.1, which is too large for the machine to hold. In this case, too, we have an overflow and the LGP-30 stops computation. However, if we divide 3 at q = 3 by 2 at q = 2 the result is 1.5 at q = 1.

In the accumulator

0	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

In memory

0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

The result in the accumulator

0	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

In case the result of division includes a remainder, the quotient is rounded to the nearest bit in the thirtieth place.

Negative number representation. In the LGP-30, negative numbers are represented by their complements. A complement is formed by changing all the ones to zeroes and all the zeroes to ones and then adding one in the thirtieth position. The complement of 6 at q = 4 is as follows:

6 at q = 4

0	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

-6 at q = 4 ?

? ?

1	1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Subtraction is performed by adding the complement of the number to be subtracted.

Scaling Example. Let us reexamine the program for

$$(((a_0x + a_1)x + a_2)x + a_3)x + a_4$$

to see what happens in the accumulator when the program is executed. Suppose again that $x = \frac{1}{2}$, $a_0 = 32$, $a_1 = 24$, $a_2 = 16$, $a_3 = 10$, and $a_4 = 1$. Suppose we convert the numbers representing x and the coefficinets into binary.

$a_0 = 100000.$
$a_1 = 011000.$
× $a_2 = 001000.$ ÷ 010000 $\qquad x = 0.1$
$a_3 = 001010.$
$a_4 = 000001.$

Next, let us scale these numbers. The minimum q for a_0 is 6. Since all the coefficients play similar roles in the program, let us keep them all a q = 6. The variable x requires no scaling since it is already a binary fraction. Now let us examine what happens in the program.

$$\begin{aligned} a_0 &= .100000 = 32 \text{ at } q = 6 \\ a_0x &= .010000 = 16 \text{ at } q = 6 \\ a_0x + a_1 &= .101000 = 40 \text{ at } q = 6 \\ &\;\;\vdots \qquad\quad \vdots \qquad\qquad \vdots \\ a_0x^4 + a_1x^3 + a_2x^2 + a_3x &= .001110 = 14 \text{ at } q = 6 \\ a_0x^4 + a_1x^3 + a_2x^2 + a_3x + a_4 &= .001111 = 15 \text{ at } q = 6 \end{aligned}$$

Suppose, however, in the preceding problem, matters are complicated by having x = 1 instead of x = $\frac{1}{2}$. Then the minimum q at which we can keep x is 1. In this case, the first steps result in

$$\begin{aligned} a_0 &= .100000 = 32 \text{ at } q = 6 \\ a_0x &= .010000 = 32 \text{ at } q = 7 \end{aligned}$$

Note that we now can no longer add a_1 as we did before because a_1 is at q = 6 and a_0x is at q = 7 and we are permitted to add terms only if they are at the same q. One solution to this problem is to enter the coefficients at different q's. However, this requires a good deal of effort on the part of the operator. The best solution is to enter the coefficients in some unconverted and unscaled decimal form and have a program to convert and scale. Subroutines can be written called floating point routines which can take care of this problem, so that the LGP-30 can be used as a floating point machine.

Shifting. One of the essential requirements of such a program is that it scale numbers by shifting. There are three orders in the LGP-30 which can be used for shifting, M multiply, N multiply, and divide.

If we M multiply by 1 at q = 1 we shift right by 1. If we M multiply by 1 at q = 2, we shift right by 2. And so on. In the case of divide, if we divide by 1 at q = 1, we shift left by 1. If we divide by 1 at q = 2, we shift left by 2. And so on. How we shift with an N multiply is discussed in the next paragraph.

N multiplication. Suppose, now we are interested in the result of a multiply as a magnitude and not simply as a shift. If we multiply 1 at q = 20 in the accumulator by 1 at q = 20 in memory, the result is 1 at q = 40. The result from M multiplication is lost, since 1 at q = 40 is not in the most significant half of the product. However, we have an order which can preserve the least significant half of a product as well as the most significant half. The order which the LGP-30 provides for this function is the N multiply. The instruction n 2000 means: multiply the number in the accumulator by the number in location 2000 and retain the 31st through the 61st bits of the product in the sign position and the 30 magnitude positions of the accumulator. Note that the sign position in this case represents magnitude and not sign. The sign of the product is found as a result of the M multiply. If one of the two operands is negative, the result relates to the sign of the product to the extent that it is complement in form. Why the thirtieth magnitude position of the result is sometimes significant is shown in the discussion of input and output.

The N multiply can be used for shifting left. For instance, to N multiply by a 1 at q = 30 shifts left 1. To N multiply by a 1 at 29 shifts left by 2. And so on.

Hexadecimal Digits. A shorthand is useful to indicate each word, since it is somewhat space consuming and time consuming to write out 32 o's and 1's. In order to develop such a shorthand the word is marked off into groups of four bits each to total 32 bits consisting of the sign bit, the 30 magnitude bits, and the spacer bit. There are 16 possible combinations for any group of four bits. Hence, each combination of four bits can be represented by one of a group of 16 characters. The following table shows single character representations for each of the 16 possible combinations and their decimal equivalents.

Decimal and Hexadecimal
Equivalents of Binary Numbers

Binary	Hexadecimal		Decimal
0 0 0 0	0		0
0 0 0 1	1		1
0 0 1 0	2		2
0 0 1 1	3		3
0 1 0 0	4		4
0 1 0 1	5		5
0 1 1 0	6		6
0 1 1 1	7		7
1 0 0 0	8		8
1 0 0 1	9	D-1000	9
1 0 1 0	f	b	10
1 0 1 1	g	c	11
1 1 0 0	j	d	12
1 1 0 1	k	e	13
1 1 1 0	q	f	14
1 1 1 1	w	g	15

This method of single character representation is, in fact, the number system of base 16 called the hexadecimal system. Just as there are the two digits 0 and 1 in the number system base 2 and the ten digits 0 through 9 in the number system base 10, so also there can be digits 0 through W in the number system base 16.

For Example, we converted the number 19 in decimal into 10011 in binary and scaled to .10011. Hence, the binary and hexadecimal equivalents of 19 at a q - 5 as they appear in a word are

```
|0|1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
   4       j       0       0       0       0       0       0
```

Scaling for Range. Suppose we have a problem which requires computing ALQ^2 for various values of A, L, and Q but where each variable is confined to a range of values as follows:

$$12 \times 10^{-15} \leq A \leq 523{,}000 \times 10^{-15}$$
$$0 \leq L \leq 9{,}999$$
$$0 \leq Q \leq 125{,}000$$

and we want our result in the range

$$0 \leq ALQ^2 \leq 12$$

but we want our accuracy to supply the third decimal place from 0.000 to 12.000.

First let us determine the minimum q for values of L. Since 9,999 is the maximum value for L, it determines the minimum q. We can find the minimum q by referring to the table of powers of 2. From this table we see that $2^{13} < L_{max} < 2^{14}$. Hence q = 14 is high enough for the minimum q. We can generalize the procedure for finding the minimum q. If there is a number x such that

$$2^{n-1} \leq x < 2^n$$

then $q = n$ is sufficiently large for q_{min}, but if $x = 2^n$, $q = n$ is not large enough and $q_{min} = n + 1$ is required.

We follow this procedure to determine that

for A, $q_{min} = -30$
for L, $q_{min} = 14$
for Q, $q_{min} = 17$

If we are to provide for the whole range of values for our variables, we must be sure that our program can store the values of these variables at no smaller q's than these.

TABLE OF POWERS OF 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	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
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125

Next, by following the rule that multiplication implies the addition of q's, we determine that

for AL, $q_{min} = -16$
for ALQ, $q_{min} = 1$
for ALQ^2, $q_{min} = 18$

Since word length is 30 bits, there are 30 - 18 = 12 bits used for expressing the fractional part of the answer. Since $2^{-12} < .001_{10}$, the result is at a q sufficinet to provide the accuracy required. Note that we might have specified conditions which could not have been met by single precision operation. Floating point subroutines are available which take care of all such scaling problems as these.

Binary Representation of Orders. The order part of an instruction is contained in bit positions 12 through 15 in a word. Each of the 16 order letters is represented by a unique pattern of zeroes and one's located in these four bit positions. The following table gives the binary equivalents of the orders.

Binary equivalents of order letters

z	0 0 0 0
b	0 0 0 1
y	0 0 1 0
r	0 0 1 1
i	0 1 0 0
d	0 1 0 1
n	0 1 1 0
m	0 1 1 1
p	1 0 0 0
e	1 0 0 1
u	1 0 1 0
t	1 0 1 1
h	1 1 0 0
c	1 1 0 1
a	1 1 1 0
s	1 1 1 1

INPUT OUTPUT AND CONTROL

Print. Before discussing how to get information into the LGP-30, we must discuss the print order. A print instruction executes the typewriter keyboard function indicated by the 6 track bit. A print instruction affects only the typewriter and has no effect on any memory location, the accumulator, or the counter register.

For example, p 2000 has 010100 in the track bits which is the code for a back space on the typewriter. The execution of p 2000, then, results in the typewriter's back spacing. The following table gives a complete list of typewriter keyboard codes.

LGP-30 Input Output

Keyboard Code

Numerical

	123456
)0	000010
L1	000110
*2	001010
"3	001110
△4	010010
%5	010110
$6	011010
¶7	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

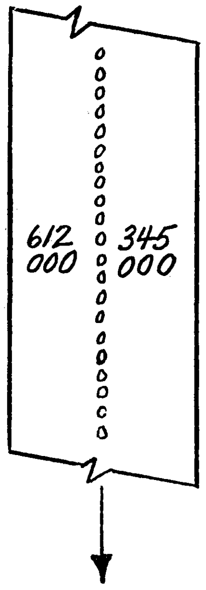
	123456	
Lower Case	000100	4
Upper Case	001000	8
Color Shift	001100	10
Car Ret	010000	16
Back Space	010100	20
Tab	011000	24
Cond Stop(')	100000	32
Start Read	000000	0
Space	000011	3
Delete	111111	63

Signs

= +	001011	11
_ -	000111	7

Balance of Keyboard

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



The print order provides the means for executing any typewriter function.

The principal method of entering information is by means of paper tape punched with holes representing the keyboard codes. Hence, in order to enter data, we must have the means of starting the tape reader. If we are to program the entry of data, we can begin by executing a p 0000 instruction which starts the tape reader. Although the print instruction is the means we have of creating output, the unique instruction p 0000 is related to input.

Input. The instruction p 0000 must always be followed, but not necessarily immediately, by the instruction i 0000, the input instruction. This instruction transfers the characters read on the tape into the accumulator. No address portion other than 0000 is ever associated with the input order, and the instruction p 0000 always precedes the instruction i 0000.

One of the typewriter codes which can appear on the punched paper tape is the stop code. Its function is to stop the tape reader and to send a start signal to the computer. After the codes for characters necessary to fill a given word are read from the tape and transferred into the accumulator, a stop code is used. When the computer receives a start signal, it executes the instruction which follows the i 0000 instruction in memory. Often this is a clear or hold instruction so that the word which has been filled into the accumulator can be stored in some memory location.

Note that the stop code on the tape is reciprocal in function to the pair of instructions p 0000 and i 0000 in the computer. The pair stops computation and starts the reader, whereas, the stop code stops the reader and starts computation.

Be sure not to confuse a stop code with a stop order or stop instruction. A stop instruction can be located in the computer to stop computation but a stop code is always on tape and stops the tape reader.

Two types of input are provided, 4 bit and 6 bit. Four bit input fills into the accumulator only the first four of the 6 bits representing each character. Hence, although 000101 represents b and 000110 represents 1, both have the same effect on the accumulator when read from tape (0001). When characters are read from tape, they are typed at the same time so that although the codes for 1 and b have the same effect on the computer using four bit input, the typed result is different. The 6 bit input switch on the computer control console selects 6 bit input when depressed. For numerical work, the use of 4 bit input is more common.

Since the accumulator is 32 bit positions long, including sign bit and spacer bit, 8 characters are sufficient to fill the accumulator on 4 bit, input. Hence, a stop code must appear on tape at least every eight characters. The accumulator is filled four bits at a time from the right hand end. That is, when the first character is read, it goes into the last four bit positions of the accumulator. When the next character is read, it occupies the last four bit positions of the accumulator and pushes the first character read into the next to last four bits. And so on. If a ninth character, is read, the first character read is lost.

Bootstrap. Consider, now the following program.

Location	Instruction or Number	Operand	Results or Notes
0000	p 0000		start tape reader
0001	i 0000		bring in a word
0002	c(2000)	input word	
0003	b 0002	c( )	
0004	a 0007	1 at 29	modify c( )
0005	y 0002	c( )	
0006	u 0000		return to input
0007	1 at 29		

This program represents the simplest type of input routine, one which simply brings in words and stores them without conversion. It is called a bootstrap routine. Note that the initial clear instruction has an address 2000. Actually this initial address could be anything from 0008 through 6363 depending on the number of words to be filled. Hence, we must prepare our input routine to include the initial address into which we wish to fill the words on tape. Actually, we could modify the routine so that it would use the first word on the tape as the start fill address.

Note also, that although we loop through the instructions repeatedly, there is no need to have a counter. This is because after the last word has filled and computation returns to the input instruction, there is no further stop code on the tape to stop the tape reader and to send a start signal to the computer. Hence, the reader continues even after the end of the tape has passed out of the reader and we can stop the reader at our convenience simply by depressing the STOP READ switch on the typewriter.

Now the question arises, "How do we get the bootstrap routine into memory?" Our discussion of input so far assumes that there is already an input routine in the computer. Before discussing the process of entering the bootstrap, there are three things to consider.

Turning Power On. First, let us consider turning the power on. Turning on the typewriter is simple. Just flip the toggle switch to ON. The typewriter can be turned on even if the computer is not on, but the computer must be plugged in. To turn on the computer:

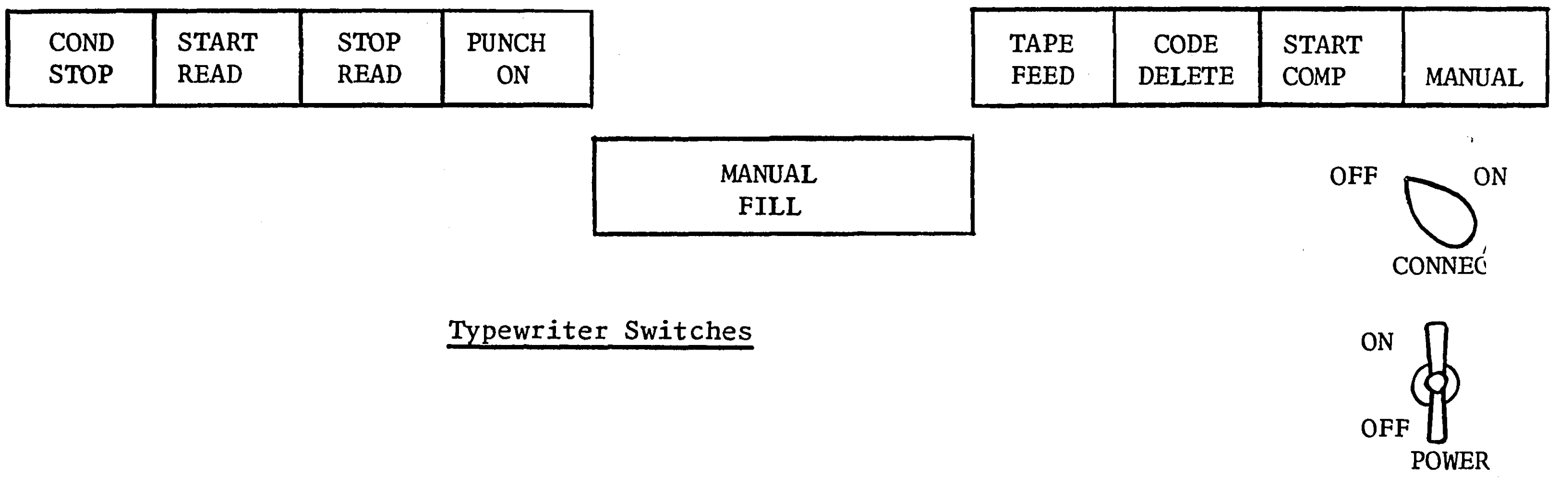
(1) Depress MANUAL INPUT switch. Note that the NORMAL, ONE OPERATION, and MANUAL INPUT switches are interlocked. The MANUAL INPUT switch can only be depressed after the ONE OPERATION switch is depressed.

(2) Depress the OPERATE switch.

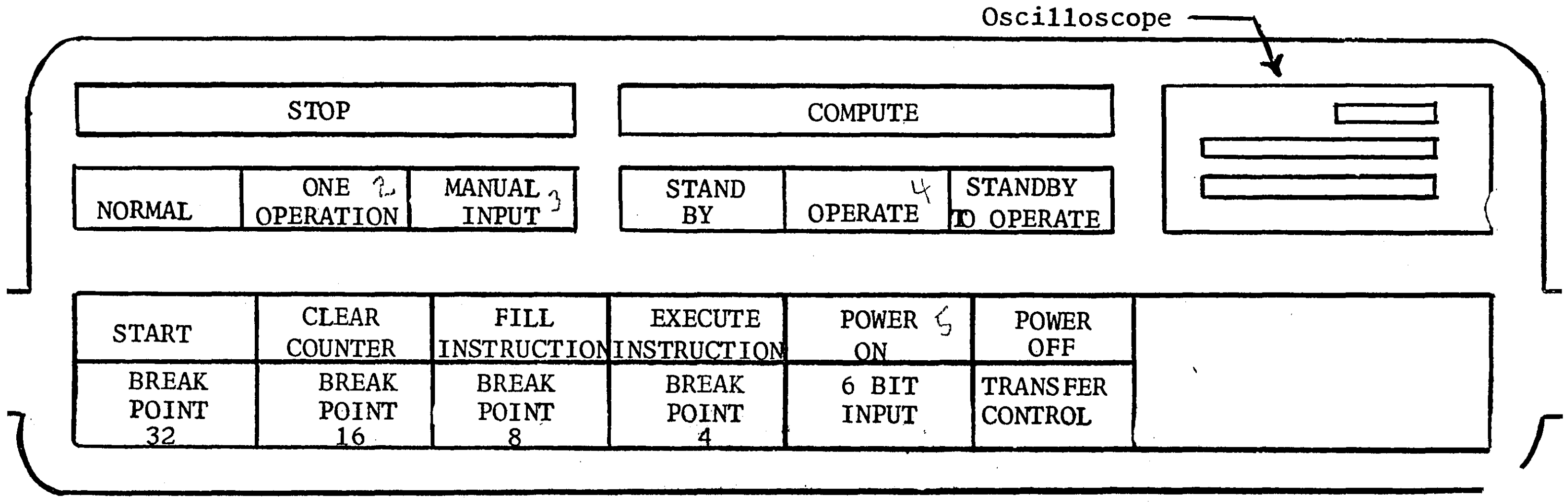
(3) Depress the POWER ON switch.

For fifty seconds the STANDBY light is on to indicate that tubes are at half filament power. For fifty more seconds the STANDBY TO OPERATE light is on indicating that the tubes are at full filament power and that the drum motor is energized. When the operate light comes on, the D.C. voltages are applied and the computer is ready for use.

Control Switches. Second, let us look at some of the other switches on the typewriter and on the control console of the computer. The CONNECT switch on the typewriter when turned off prevents start signals from passing from the typewriter to the computer and start-read signals from going from the computer to the typewriter. When the MANUAL INPUT switch on the computer is depressed, keyboard characters typed on or read by the typewriter fill into the accumulator even though the CONNECT switch may be off. With the MANUAL INPUT switch depressed, recording in the memory is impossible, and therefore pressing the START switch by mistake can do no harm if an error has been made in typing. The FILL INSTRUCTION switch transfers the contents of the accumulator to the instruction register. When the ONE OPERATION switch is depressed, pressing the START switch executes one instruction at a time. The instruction executed is the one contained in the location given by the counter register. However, if the EXECUTE INSTRUCTION switch is depressed when the ONE OPERATION switch is down, it is the instruction in the instruction register which is executed.

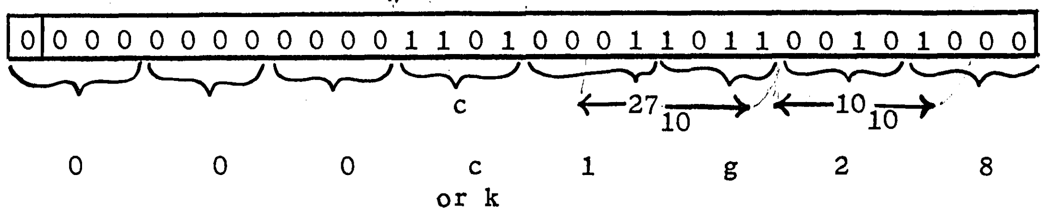


Typewriter Switches



Control Console Switches

Instruction Representation. Finally, let us consider the hexadecimal representation of instructions. Consider, for instance, the binary representation of the instruction c 2710.



Note that there is no problem with the order bits. If we had filled c 2000, the address portion of the word in memory would have been wrong but the order part would have been correct.

The track bits appear different because the two highest order track bits belong to one hexadecimal character on input and the lowest four to another.

The sector bits appear different for two reasons. First, like the track bits they are divided between two hexadecimal characters. Second, the lowest order sector bit corresponds to a hexadecimal 4 not a 1 in the last hexadecimal position of the word.

A quick method of conversion is to divide the track number in decimal by 16 to get the first hexadecimal character and then express the decimal remainder as a hexadecimal digit for the second hexadecimal character. For sector, divide the decimal number for sector by 4 and express the result as a hexadecimal digit. Then multiply the remainder by 4 and express the result as a hexadecimal digit.

```
 27/16              =   1    (+ remainder)
 27-(1x16) = 11     =   g
 10/4               =   2    (+ remainder)
(10 - 2x4) x 4      =   8
```

Filling the Bootstrap. Now suppose we want to put the instruction p 0000 into location 0000.

(1) Depress MANUAL INPUT switch on computer control console
(2) Turn CONNECT switch off
(3) Turn on the typewriter and the computer
(4) Type c0000 on typewriter keyboard
(5) Depress FILL INSTRUCTION switch
(6) Type 000p0000 on typewriter keyboard
(7) Depress ONE OPERATION switch
(8) Depress EXECUTE INSTRUCTION switch

The instruction p0000 has now been stored in location 0000.

Suppose we wish to fill i 0000 into location 0001. The procedure is the same except that we have 000i0000 and the clear instruction is c0004. Instead of typing each instruction we could have put on tape

c0000'000p0000'c0004'000i0000' . . . etc.

The apostrophes stand for stop codes. Then our instructions would be the same except that instruction (4) and (6) would be "Depress START READ switch on the typewriter." and to fill the entire bootstrap we would add instruction (9)"Return again to step (3) until all the words of the bootstrap routine are filled into memory."

It is important that the CONDITIONAL STOP switch on the typewriter is not depressed. If it is, one pressing of the START READ switch sends the entire tape through the reader without stopping at the end of each word.

Note that we can express the number which is added to modify the address of the clear instruction as 00000004_{16} or $z0001_{10}$ since the code for the z order is 0000_2. The tape for the whole bootstrap is c0000'000p0000' c0004'000i0000'c0008'000cxxxx'c000j'000b0008'c0010'000a001j'c0014'000y0008' c0018'000u0000'c001j'000z0004'. Carriage return codes may be interspersed among instructions on the tape in order to limit line length for typing. When used, carriage return codes should follow the stop orders.

Executing the Bootstrap. Now let us execute the bootstrap.

(1) Turn CONNECT switch on
(2) Depress CLEAR COUNTER switch. This action puts zero in the counter register so that the next instruction to be executed is the instruction in location 0000. Since we usually place a program input routine such as the bootstrap starting in location 0000, clearing the counter is the quickest way to get to the program input routine. Either the MANUAL INPUT switch or the ONE OPERATION switch must be down, not the NORMAL switch.
(3) Depress the NORMAL switch
(4) Depress the Start switch
(5) When the last word on the tape has been read, depress the STOP READ switch on the typewriter.

Program Input Routine. The bootstrap input routine described has several shortcomings. For one thing, it requires manually filling the initial address of the clear instruction, called the start fill address. For another, both numbers and instructions must be put in hexadecimal. However, a simple bootstrap such as this can be used to load another program capable of filling instructions expressed in decimal form, of distinguishing between instructions and data words, and of accepting a word on tape as a start fill instruction. Such a program input routine is described in the Subroutine Manual for the LGP-30.

Executing a Program. If we wish to execute a program located somewhere other than in 0000, we cannot use the CLEAR COUNTER switch to reach it. To transfer manually to any location in memory

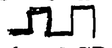
(1) Turn CONNECT switch off
(2) Depress MANUAL INPUT switch on computer control console
(3) Type, for instance, u2000 in hexadecimal
(4) Depress FILL INSTRUCTION switch
(5) Depress ONE OPERATION switch
(6) Depress EXECUTE INSTRUCTION switch

At this point 2000 appears in the counter register. Next, if we depress the NORMAL switch and the START switch, the program beginning in 2000 is executed.

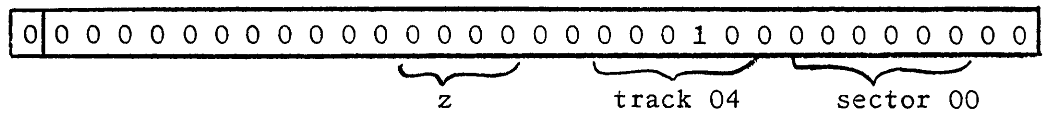
Calling for Data. The pair p 0000 and i 0000 need not appear only at the beginning of a program input routine. They can appear in the middle of a program to bring in more data. We could have added them to the program discussed earlier so that values for x and the coefficients a_0 through a_n could have been brought in from tape. When these instructions are included in a program, be sure that the CONNECT switch is on. If the MANUAL switch on the typewriter is depressed when the p 0000 and i 0000 pair are executed, the reader does not start but the MANUAL FILL light on the typewriter comes on. When this happens data can be entered from the keyboard. After the desired characters have been typed, depressing the START COMPUTER switch on the typewriter continues execution of the program.

Filling the Spacer. Typing a character such as 9 for input fills the last four bits of the accumulator so that 1001 appears in the 28th, 29th, 30th, and spacer positions of the accumulator. Hence, although the spacer for a word in a memory location is always zero, it may be a one in the accumulator. In order not to lose this bit by storing immediately following accumulator fill, it is often worthwhile to execute an N multiply by a 1 at 30 in order to shift the input word by one bit.

Using the typewriter. The typewriter handbook covers those functions of the typewriter which do not relate to the computer such as punching a tape. However, it is worth noting that a good safety precaution in punching tape or otherwise using the typewriter by itself is to turn the CONNECT switch off to prevent the interchange of start signals between the computer and the typewriter and to depress the MANUAL INPUT switch on the computer so that recording in memory is prevented.

Reading the Scope. An oscilloscope on the LGP-30 makes it possible to read the contents of the accumulator, the instruction register, and the counter register in binary representation. Wherever there is a square wave on the scope there is a one stored; otherwise, there is a zero stored. For instance,  represents 0101 in binary or 5 in decimal. In ONE OPERATION the LGP-30 executes a program one at a time as the START switch is depressed. The middle line on the scope shows a binary representation of the instruction which has been executed as the result of the last depression of the START switch. The counter register shown on the top line, gives the address of the next instruction to be executed, and the accumulator, shown on the bottom line, shows the results. In MANUAL INPUT, when the START switch is depressed successively, the contents of successive locations appears in the instruction register. The counter register, in this case also, gives the address of the location following the one shown in the instruction register. No change in the appearance of the accumulator occurs in this mode of operation since the contents of locations are simply being observed and instructions are not being executed. In ONE OPERATION the same instructions appear successively in the instruction register as in MANUAL INPUT except in the case of unconditional transfer instructions and test instructions which may result in a transfer.

Break Points. If computation is to stop, the address portion of the stop instruction is usually of no significance. However, the stop order has a special characteristic. If there is a one in the third bit from the right hand end of the track portion of a stop instruction,

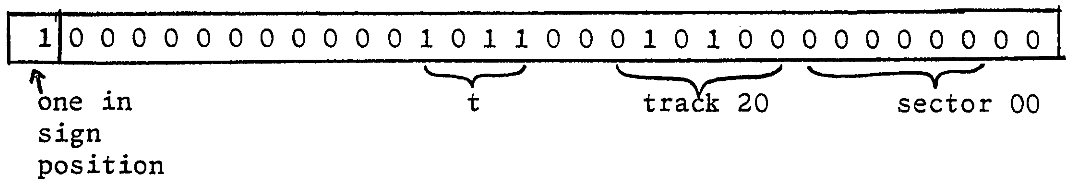


computation does not stop if break point switch 4 on the console of the computer is depressed. There is a similar relationship between the stop order and each of the other four break point switches as follows:

Track bits	Break Point Switch
000100	4
001000	8
010000	16
100000	32

Break points may be combined. For instance, if a track number in a stop instruction is 010100, computation fails to stop if either break point switch 4 or 16 is depressed.

Transfer Control. The test order has one special characteristic. If it is stored in memory with a one in the sign position



the instruction acts like a test instruction unless the "transfer control" switch on the console of the computer is depressed. If the "transfer control" switch is depressed, the test instruction acts like an unconditional transfer instruction, regardless of whether a one or a zero is in the sign bit of the word in the accumulator.

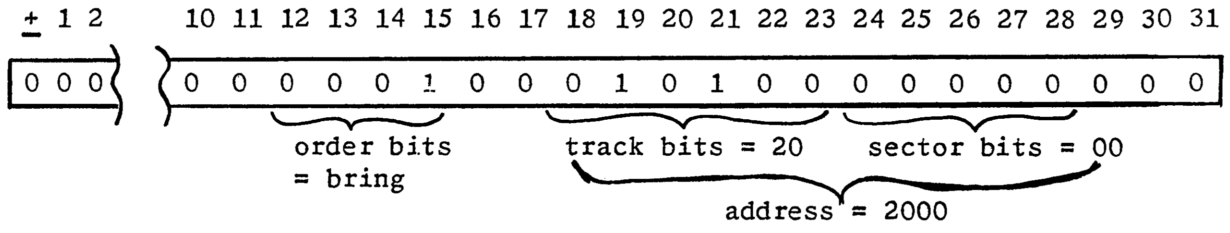
Printing Out. When a print instruction other than p 0000 is executed, a signal goes to the typewriter to execute a typewriter character or function such as the letter "a" or a carriage return. However, unlike the case when there is an input instruction following the print instruction, computation does not stop. After the typewriter has executed the character or function it sends a start signal to the computer. Hence, it is desirable to have computation in the computer stop before the completion of character execution at the typewriter. Therefore, a stop instruction should follow a print instruction. Since the execution of a typewriter character requires about 6 drum revolutions computation requiring up to 6 drum revolutions may occur between a print instruction and the following stop instruction.

Turning Power Off. To turn power off, first depress the MANUAL INPUT switch and then depress the POWER OFF switch. In MANUAL INPUT mode of operation, recording in the memory is impossible so that transients occurring when the LGP-30 is turned off cannot affect the contents of memory.

It is possible to depress the STANDBY switch so that only half tube filament power remains on. When the machine is not to be used for some time period during a shift of operation, the LGP-30 can be maintained in STANDBY mode of operation rather than in OPERATE mode so that tube life can be extended. In switching to STANDBY as in turning power on or off be sure that the MANUAL INPUT switch is depressed first.

SUMMARY OF ORDERS

An instruction consists of an order part such as the letter b for bring and an address part such as the number 2000 to designate a memory location. All instructions have a similar appearance in an LGP-30 word. The order bits occupy positions 12 through 15 of the word and the address bits occupy positions 18 through 29 of the word. For instance, the instruction b 2000 appears as



The following pages describe the orders of the LGP-30 and give examples of instructions using the orders for which the address part is usually 2000. For all of these cases the appearance of the instruction in a word is as given above except for the order bits.

Bring from Memory:

Letter designation:

b which is equivalent to 0001 binary, 1 decimal, 1 hexadecimal
Example of a bring order as used in an instruction:

b 2000

Meaning:

Replace the contents of the accumulator with the contents of memory location 2000. The contents of memory location 2000 is unaffected. This order is equivalent to a reset and add order in some other computers.

Hold and Store:

Letter designation:

h which is equivalent to 1100 binary, 12 decimal, j hexadecimal
Example of a hold order as used in an instruction:

h 2000

Meaning:

Replace the contents of memory location 2000 with the contents of the accumulator. The contents of the accumulator is unaffected. This order is equivalent to a write order in some other computers.

Clear and Store:

Letter designation:

c which is equivalent to 1101 binary, 13 decimal, k hexadecimal
Example of a clear order as used in an instruction:

c 2000

Meaning:

Replace the contents of memory location 2000 with the contents of the accumulator and replace the contents of the accumulator with zero. This order is equivalent to a write and reset order in some other computers.

- -

Store Address:

Letter designation:

y which is equivalent to 0010 binary, 2 decimal, 2 hexadecimal
Example of a store address order as used in an instruction:

y 2000

Meaning:

Replace the contents of the address portion of the word in memory location 2000 with the contents of the address portion of the word in the accumulator. The contents of the accumulator is unaffected.

Notes:

If the word in the accumulator before execution of the instruction y 2000 is an a 3000 and the word in memory location 2000 is a b 5000 the result in memory location 2000 after execution is b 3000.

- -

Unconditional transfer:

Letter designation:

u which is equivalent to 1010 binary, 10 decimal, f hexadecimal
Example of an unconditional transfer order as used in an instruction:

u 2000

Meaning:

Replace the number in the counter register with the contents of the address portion of the unconditional transfer instruction.

Notes:

Normally instructions are executed in sequence according to memory location. For instance, if b 3000 is located in 1000 and h 4000 is located in 1001, the hold instruction is executed immediately after the bring instruction. This is governed by the fact that during the execution of the instruction located in 1000, one is added to the 1000 that already exists in the counter register. After execution of the instruction in 1000, the counter register gives the address of the next instruction to be executed; in this case, the instruction in 1001.

However, if instead of b 3000 the instruction u 2000 had been located in 1000, 2000 would be in the counter register after execution of the u instruction, and the instruction in 2000 would follow the instruction in 1000 instead of the instruction in 1001. Hence, computation is transferred to location 2000.

Most instructions affect or use the contents of the accumulator but unconditional transfer and return address instructions affect the counter register.

- -

Return address:

Letter designation:

r which is equivalent to 0011 binary, 3 decimal, 3 hexadecimal
Example of a return address order as used in an instruction:

r 2000

Meaning:

Add one to the contents of the counter register and replace the address portion of memory location 2000 with the contents of the counter register.

Notes:

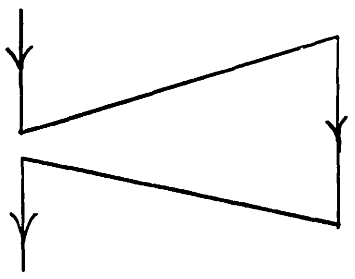
If the word in memory location 2000 is u 0000 before execution of the return address instruction and if the return address instruction is located in 1000, the result after execution is u 1002 in memory location 2000.

This type of instruction is almost always used before a u instruction. The purpose of this pair of instructions is to execute a block of instructions not in normal sequence.

For instance:

Location	Instruction
1000	b 3000
1001	h 3001
1002	r 2000
1003	u 1900
1004	h 3002
1005	c 3003
.	.
.	.
.	.

Path of Computation



Location	Instruction
1900	c 4000
1901	b 4001
.	.
.	.
.	.
1963	c 4009
2000	u 1004

This program segment shows that the r and u instructions allow computation to transfer from the 1000, 1001, 1002, 1003 sequence into the 1900 through 2000 sequence and then to transfer back to the 1004, 1005, etc. sequence.

- -

Test:

Letter designation:

t which is equivalent to 1011 binary, 11 decimal, g hexadecimal
Example of a test order as used in an instruction:

t 2000

Meaning:

If a one is in the sign bit of the word in the accumulator, the test instruction has the effect of an unconditional transfer. If a zero is in the sign bit of the word in the accumulator, the next following instruction in normal sequence is executed.

Note A:

Since a zero in the accumulator has a zero in the sign bit, a test instruction goes on to the next instruction in normal sequence if the word in the accumulator is zero or positive and transfers if the word in the accumulator is negative. For instance, if the word in the accumulator is zero and t 2000 is located in 1000, the next instruction to be executed is the one located in 1001.

Note B:

The test order has one special characteristic. If it is stored in memory with a one in the sign position,

$$\boxed{1\,|\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,\underbrace{1\,0\,1\,1}_{t}\,0\,0\,\underbrace{0\,1\,0\,1\,0\,0}_{\text{track 20}}\,\underbrace{0\,0\,0\,0\,0\,0}_{\text{sector 00}}\,0\,0}$$

one in sign position

the instruction acts like a test instruction unless the "transfer control" switch on the console of the computer is depressed. If the "transfer control" switch is depressed, the test instruction acts like an unconditional transfer instruction, regardless of whether a one or a zero is in the sign bit of the word in the accumulator.

- -

Stop:

Letter designation:

z which is equivalent to 0000 binary, 0 decimal, 0 hexadecimal
Example of a stop order as used in an instruction:

z 2000

Meaning:

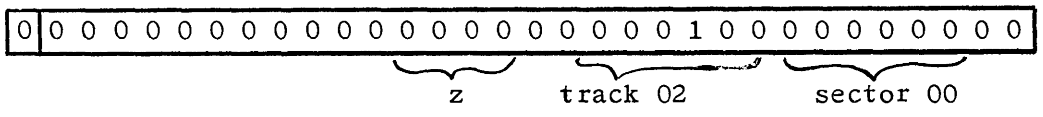
Stop computation.

Note A:

When the computer begins computation in normal operation, each instruction is executed in sequence according to memory location as fast as the computer can execute them. A stop instruction is the method used to prevent the computer from going on to some sequence of instructions other than those required for solving the problem at hand.

Note B:

If computation is to stop, the address portion of the stop instruction is usually of no significance. However, the stop order has a special characteristic. If there is a one in the third bit from the right hand end of the track portion of a stop instruction,



computation does not stop if break point switch 4 on the console of the computer is depressed. There is a similar relationship between the stop order and each of the other four break point switches as follows:

Track bits	Break Point Switch
000100	4
001000	8
010000	16
100000	32

Break points may be combined. For instance, if a track number in a stop instruction is 010100, computation fails to stop if either break point switch 4 or 16 is depressed.

- -

Print:

Letter designation:

p which is equivalent to 1000 binary, 8 decimal, 8 hexadecimal

Example of a print order as used in an instruction:

p 2000

Meaning:

Execute the typewriter keyboard function indicated by the 6 track bits. The print order has no effect on the contents of any memory location, the accumulator, or the counter register.

Notes:

For example, p 2000 has 010100 in the track bits which is the code for a back space on the typewriter. The execution of p 2000, then, results in the typewriter's back spacing. A table in the section INPUT OUTPUT AND CONTROL gives a complete list of typewriter keyboard codes. The print order provides the means for the execution of any typewriter function by the computer.

Special note must be made of the instruction p 0000; that is, the execution of typewriter keyboard code 000000. This instruction starts the tape reader as necessary for bringing from tape words for storage in the memory. The instruction p 0000 is always followed in memory but not necessarily immediately by i 0000, the input instruction. After a p 0000 instruction starts the tape reader, an i 0000 instruction transfers into the last 4 bit positions of the accumulator the first 4 bits of the typewriter code for the first character read on the tape. When the second character is read, the bits representing the first character are shifted into the next to last four bit positions of the accumulator and the first four bits of the typewriter code of the second character on tape are placed in the last four bit positions of the accumulator. This process continues indefinitely until a stop code (100000) appears on the tape. The stop code stops the tape reader and sends a start signal to the computer so that the instruction following i 0000 in memory is executed. Often this next instruction is a hold or clear instruction so that the characters read into the accumulator can be stored in some memory location.

It is never desirable to have more than eight characters to be filled into memory preceding a stop code on tape. If more characters are on tape, the four bits in the accumulator representing the first of the nine characters is shifted out of the accumulator and lost when the ninth character on the tape is read.

The p 0000 i 0000 combination is reciprocal in function to the stop code on tape; that is, it stops computation and starts the tape reader, whereas the stop code on tape stops the tape reader and starts computation. This combination is also required for manual input as well as for tape input. When the MANUAL switch on the typewriter is depressed, the execution of p 0000 and i 0000 instructions stops computation but turns on the MANUAL FILL light on the typewriter instead of starting the tape reader. When the MANUAL FILL light is on, characters typed on the keyboard enter the accumulator as they enter when read from tape. When the characters desired have been typed, depressing the START COMPUTER switch on the typewriter sends a start signal to the computer.

When a print instruction other than p 0000 is executed, a signal goes to the typewriter to execute a typewriter character of function such as the letter "a" or a carriage return. However, unlike the case when there is an input instruction following the print instruction, computation does not stop. After the typewriter has executed the character or function, it sends a start signal to the computer. Hence, it is desirable to have computation in the computer stop before the completion of character execution at the typewriter. Therefore, a stop instruction should follow a print instruction. Sinçe the execution of a typewriter character requires about 6 drum revolutions, computation requiring up to 6 drum revolutions may occur between a print instruction and the following stop instruction.

- -

Input:

Letter designation:

i which is equivalent to 0100 binary, 4 decimal, 4 hexadecimal
Example of an input order as used in an instruction:

i 0000

Notes:

The address portion of an instruction constructed from this order is always 0000. It is always preceded by the instruction p 0000. Refer to the notes under print order for an explanation of its meaning.

Add:

Letter designation:

a which is equivalent to 1110 binary, 14 decimal, q hexadecimal
Example of an add order as used in an instruction:

a 2000

Meaning:

Add the contents of memory location 2000 to the contents of the accumulator and place the result in the accumulator. The contents of memory location 2000 is unaffected.

Notes:

Suppose we consider adding a 2 in memory location 2000 to a 3 in the accumulator each scaled to a q = 3. A scale factor q = 3 is equivalent to considering the number to be shifted 3 places to the right with respect to the binary point. The appearance of the words before execution is as follows:

In location 2000:

0	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

In the accumulator:

0	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

The result in the accumulator after execution is:

0	1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Note that although the numbers 2 and 3 could have been expressed at a scale factor q = 2, the result could not. In such a case we say that the accumulator overflows. The machine is designed to stop computation if such a situation arises. To prevent overflow it is necessary to carry the numbers to be added at a high enough q. It is possible, of course, to shift so far to the right as to lose significant digits. For example, a 3 at q = 31 appears in memory only as a 2 at 31 since the word is 30 bits long.

Hence, the criteria which determine the scale factor at which to perform additions are (1) to carry the number far enough left to prevent loss of significant digits and (2) to carry it far enough right to prevent overflow.

Note, too, that two numbers must be added at the same scale factor q. The result of adding 2 at 3 and 3 at 4, for instance, would be

```
|0|0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
```

which is not 5 at <u>any</u> scale factor and hence has no significance in relation to the process of adding 2 and 3.

- -

<u>Subtract:</u>

<u>Letter designation:</u>

<u>s</u> which is equivalent to 1111 binary, 15 decimal, w hexadecimal
<u>Example</u> of a <u>subtract</u> order as used in an instruction:

s 2000

<u>Meaning:</u>

Subtract the contents of memory location 2000 from the contents of the accumulator and place the result in the accumulator. The contents of memory location 2000 is unaffected.

<u>Notes:</u>

If the number in memory location is of the same sign as the number in the accumulator, the result of subtraction is a number smaller in absolute value than the larger of the two numbers. Hence, in such a case, overflow as described under <u>add</u> cannot occur. However, if the numbers are of opposite sign, overflow <u>can</u> occur and the numbers must be scaled as described in the notes under <u>add</u> in this summary.

Negative numbers are represented in the LGP-30 by a complement formed as follows. Change all the ones to zeroes and all the zeroes to ones and add a one in the thirtieth bit position. The process of subtraction is accomplished by complementing the minuend and adding the complement to the subtrahend. For instance, suppose we subtract 2 from 5 each held at a scale factor q = 3.

In memory:

```
|0|0 1 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
```

In the accumulator:

```
|0|1 0 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
```

Complement of minuend:

1	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Result in accumulator:

0	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

The result, of course, is 3 at a scale factor q = 3.

- -

M Multiply:

Letter designation:

m which is equivalent to 0111 binary, 7 decimal, 7 hexadecimal
Example of an M multiply order as used in an instruction:

m 2000

Meaning:

Multiply the number in the accumulator by the number in memory location 2000 and place the most significant thirty bits of the product in the accumulator. The contents of memory location 2000 is unaffected.

Notes:

In general, the multiplication of two numbers results in a product which has as many digits as the sum of the number of digits in the multiplier and the multiplicand. For instance, the product of 9, a one digit number, and 12, a two digit number, is 108, a three digit number. In the LGP-30, a word in memory has 30 binary digits of magnitude. A word in the accumulator may have as many as 31 binary digits of magnitude, if the spacer bit is filled on input. Hence, a multiplication in the LGP-30 can result in a product with 61 magnitude bits. The result of an M multiply order is the sign bit and the 30 most significant bits of the 61 bit product.

Multiplication requires the addition of scale factors. For instance (0.2) (.03) = (.006) can be interpreted in terms of scale factors as 2 (at a scale factor of 1) times 3 (at a scale factor of 2) equals 6 (at a scale factor of 3).

As an example of an M multiply, suppose we multiply 3 at a scale factor q = 3 in the accumulator by 2 at a scale factor q = 4 in memory location 2000. The result in the accumulator is 6 at a scale factor of q = 7.

In the accumulator:

0	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

In memory:

```
|0|0 0 1 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
```

Result:

```
|0|0 0 0 0 1 1 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
```

As noted under add in this appendix, numbers to be added must be at the same scale factor. Hence, it is sometimes necessary to shift one of the numbers to be added so that it is at the same scale factor as the other. For instance, in the example above, the number in memory can as well be interpreted as 1 at a scale factor q = 3 as 2 at a scale factor q = 4. The result then can be interpreted as 3 at a scale factor q = 6. Hence we can shift a number to the right using an M multiply by an amount equal to the scale factor at which we carry the 1 in the multiplier. In this case we shifted 3 from a scale factor q = 3 to a scale factor q = 6.

Note that an M multiply can result only in a shift to the right because all numbers in the LGP-30 must be scaled to less than one since the binary point is at the lefthand end of the accumulator. Hence, an M multiply is sometimes referred to as a fractional multiply.

Let us consider another example. Suppose we multiply 3.25 at a scale factor q = 15 in the accumulator by 2 at a scale factor q = 15 in memory location 2000. The result in the accumulator is 6.5 at a scale factor q = 30.

Result:

```
|0|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0|
```

Note that the result when an M multiply is used, is 6 at a scale factor q = 30 instead of 6.5. The reason is that it requires 31 bits to express 0.5 at a scale factor q = 30 and only the most significant 30 bits of the 61 bit product are retained by an M multiply. Note that the last bit in the above representation of the result is the spacer bit which is always zero as a result of an M multiply. This example also serves to point out the fact that an M multiply results in a truncated, not a rounded product.

Another type of multiply, the N multiply, is described in the section that follows.

N multiply:

Letter designation:

n which is equivalent to 0110 binary, 6 decimal, 6 hexadecimal
Example of an N multiply order as used in an instruction:

n 2000

Meaning:

Multiply the number in the accumulator by the number in memory location 2000 and place the least significant thirty-one magnitude bits of the product in the sign bit and thirty magnitude bits of the accumulator. The contents of memory location 2000 is unaffected.

Notes:

As discussed in the section under M multiply, multiplication in the LGP-30 can result in a product with 61 magnitude bits. The result of an N multiply is the least significant 31 bits of the 61 bit product. Note that there is no truncation since the result of the N multiply includes the bit of least significance of the entire 61 bit product.

The sign position as well as the thirty magnitude bit positions of the accumulator must be used to hold the thirty-one bits resulting from an N multiply. Since the sign position is required to represent a magnitude bit, there is no room in the accumulator for holding the sign bit of the product resulting from an N multiply. To N multiply by a 1 at q = 30 shifts the multiplicand left by 1 place. To N multiply by a 1 at q = 29 shifts left 2 places, and so on. Therefore, an N multiply by a 1 at q = 1 shifts left 30 places. This provides a left shift symmetrical with the right shift of the M multiply with which a 1 at q = 1 shifts right 1 and a 1 at q = 30 shifts right 30.

As pointed out in the discussion of the M multiply, scale factors add.

As an example, suppose we N multiply 3 at q = 31 in the accumulator by 2 at q = 30 in memory. Note that this implies that there is a one in the spacer of the accumulator.

In the accumulator:

```
|0|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1|
```

In memory:

```
|0|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0|
```

Result:

```
|0|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0|
```

The result, of course, is 6 at q = 61. Note that the result of M multiplying these numbers would be a zero accumulator.

The N multiply provides a left shift just as the M multiply provides a right shift. The above example can be interpreted as the multiplication of 3 at q = 31 in the accumulator by 1 at 29 in memory.

The result, then, is 3 at q = 29 in the accumulator. In other words we have shifted the 3 two places.

In an example discussed under M multiply, part of the product was lost, namely 0.5. The result 6.0 occurred instead of 6.5. If an N multiply is executed with the same operands, the result in the accumulator is

```
| 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
```

which is, in this case, 0.5 at q = 31. Hence, the one in the sign position represents, in this case, magnitude and not sign. However, if one of the operands is negative, the sign of the product is represented to the extent that the result of the N multiply is complement in form although the sign bit is not included.

The shifting right by an M multiply has led to calling the M multiply the fractional multiply. Since the N multiply shifts left, it is sometimes referred to as the integral multiply.

Note that there can never be overflow with either type of multiply since multiplication always results in the addition of q's and hence can never result in a number that is as large as 1 at q = 0.

- -

Divide:

Letter designation:

d which is equivalent to 0101 binary, 5 decimal, 5 hexadecimal
Example of a divide order as used in an instruction:

d 2000

Meaning:

Divide the number in the accumulator by the number in memory location 2000 and place the quotient rounded to thirty bits in the accumulator. The contents of memory location 2000 is unaffected.

Notes:

Suppose we divide 3 at q = 2 by 2 at q = 2. The result is 1.5 at q = 0 since the q of a quotient is the q of the dividend minus the q of the divisor. This result in binary is 1.1, which is too large for the machine to hold. In this case, too, we have an overflow and the LGP-30 stops computation. However, if we divide 3 at q = 3 by 2 at q = 2 the result is 1.5 at q = 1.

In the accumulator:

```
| 0 | 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
```

In memory:

$$\boxed{0}\boxed{1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}$$

The result in the accumulator:

$$\boxed{0}\boxed{1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}$$

In case the result of division includes a remainder, the quotient is rounded to the nearest bit in the thirtieth place.

- -

<u>Extract:</u>

<u>Letter designation:</u>

<u>e</u> which is equivalent to 1001 binary, 9 decimal, 9 hexadecimal
<u>Example</u> of an <u>extract</u> order as used in an instruction:

e 2000

<u>Meaning:</u>

Place zeroes in the word in the accumulator wherever there are zeroes in location 2000 but otherwise leave the word in the accumulator unchanged. The contents of location 2000 is unaffected.

<u>Notes:</u>

More than one kind of data may be stored in a given word. For instance, a calendar date consists of three types of data: month, day and year. A word with these three types of data might look this way.

$$\boxed{0}\boxed{\underbrace{1\ 1\ 0\ 0}_{\text{month}}\ \underbrace{0\ 0\ 1\ 1\ 1}_{\text{day}}\ \underbrace{1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1}_{\text{year}}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}$$

Sometimes it is desirable to deal with only one of the three pieces of data. The extract order makes it possible to separate different data stored in one word. The instruction e 2000 means; put zeroes in the word in the accumulator wherever there are zeroes in the word in location 2000 but otherwise leave the word in the accumulator unchanged. For instance, if the above data word is in the accumulator and in location 2000 is the word

$$\boxed{1}\boxed{1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}$$

then the result in the accumulator is the following word which contains only the month part of the date.

$$\boxed{0}\boxed{\underbrace{1\ 1\ 0\ 0}_{\text{month}}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}$$

The word in location 2000 is called the extract mask. It is possible with another extract instruction and mask to retain the day and not the month or the year, or to retain any part of any given word in the accumulator. The extract order achieves its result by multiplying bits in corresponding positions of the extract mask and the accumulator. That is, if there is a one in position 29 of both words, there is a one in position 29 of the result; otherwise, there is a zero.

TABLE OF

LGP-30 SPECIFICATIONS

Type:	General purpose, electronic, digital, single address, fixed binary point, fractional, stored program
Number Base:	2 (binary)
Word Length:	9 decimal digits plus sign (30 binary bits plus sign bit and spacer bit)
Mode of Operation:	Serial
Memory:	Magnetic drum, 4096 words, 3 one word recirculating registers.
Clock Frequency:	120 KC
Access Time:	2 ms. minimum, 17 ms. maximum
Transfer Time:	1 ms. minimum, 17 ms. maximum
Addition Time:	.26 ms. excluding access time
Multiplication or Division Time:	17 ms. excluding access time
Input-Output:	Paper tape or electric typewriter
Size:	Depth - 26", Length - 44", Height - 33"
Weight Uncrated:	740 lbs
Cooling System:	Internal forced air blower
Heat Dissipation:	5000 B.T.U. /hr.
Power Requirement:	115-volt, 60-cycle, single phase, 13 ampere alternating current
Number of Tubes:	113
Number of Diodes:	1350

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

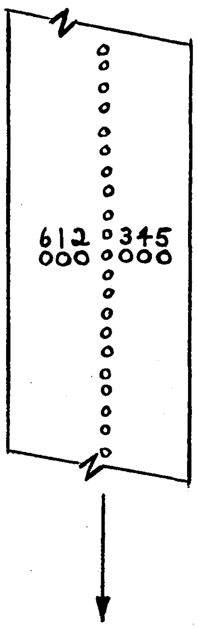
	123456
Lower Case	000100
Upper Case	001000
Color Shift	001100
Car Ret	010000
Back Space	010100
Tab	011000
Cond Stop (')	100000
Start Read	000000
Space	000011
Delete	111111

Signs

= +	001011
_ -	000111

Balance of Keyboard

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

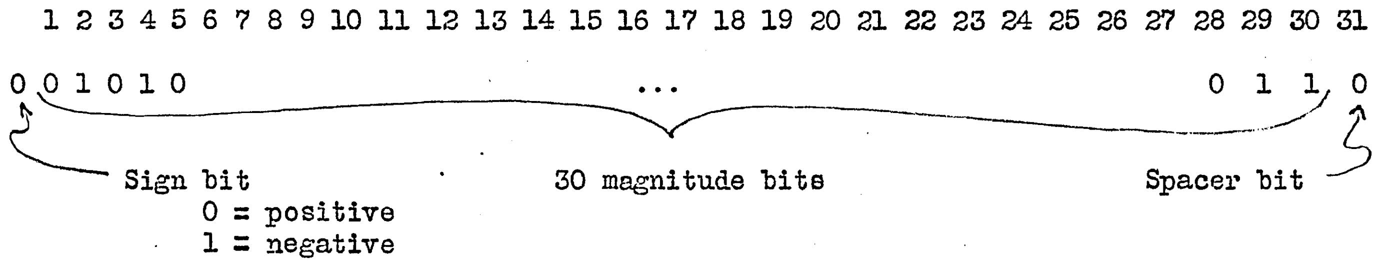


Specifications Summary for Royal Precision LGP-30

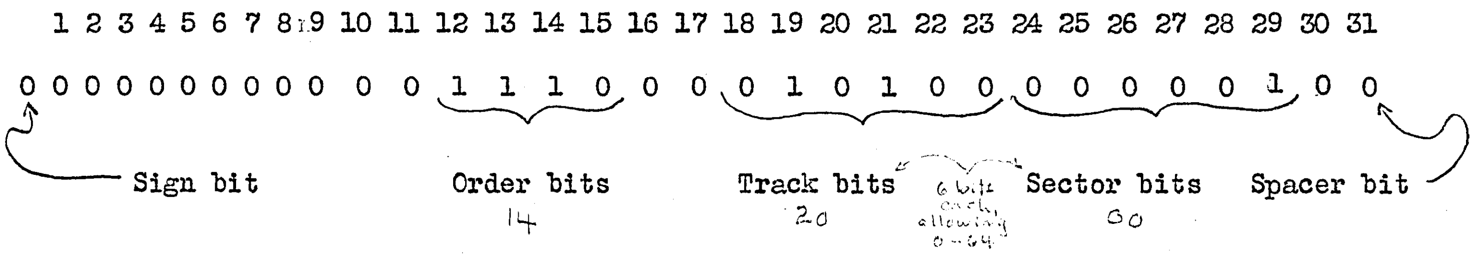
The LGP-30 is a general-purpose electronic digital computer; fixed point, fractional, internally binary, stored program.

Memory -- magnetic drum, 6.5"x7", 3700 rpm, or 1 rev. per 17 milliseconds.
64 tracks, with 64 sectors each, for 4096 memory locations.
Address specifies track and sector, from 0000 to 6363. Thus 2089 impossible.
-- three additional tracks, each with a recirculating register having access time of 0.26 milliseconds; accumulator, instruction register, and counter register.
-- each memory sector contains 31 bit positions and a blank

Word Structure:
Numbers,



Instructions,



Note: LGP-30 can record numbers from 0 to 2^{30}, roughly 10^9; or 1,000,000,000

4 bits allow 16 combinations for 16 instructions.
6 bits allow 64 combinations for 64 tracks, and 64 seftors.