#10

LGP-30

FLOATING POINT INTERPRETIVE SYSTEM (24.2)

PROGRAMMING NOTES

Prepared by:

School of Electrical Engineering

Purdue University

November 1959

1.0 INTRODUCTION TO DIGITAL COMPUTERS

1.1 The Major Portions of a Computer

A digital computer is, basically, a device which is capable of processing numbers toward some desired result. An adding machine is, therefore, a digital computer, but only in a restricted sense of the term. The adding machine has three basic parts: the input device (keyboard), the arithmetic unit (internal mechanical elements), and the output device (printed tape or display registers). It is controlled by a human operator who has a list of the operations to be performed and another list of the numbers on which to perform these operations.

The first step toward improvement of the adding machine to make a more powerful device might be to add a memory unit to it so that it could remember some of the numbers with which it was working. Then the intermediate results could be stored without the need for printing or recopying. The desk calculator does this in a small way, for it is capable of holding intermediate results in the various dial registers, and some desk calculators can even store a number from the keyboard for use in several subsequent calculations. The sequence of operations to be performed, however, is still controlled entirely by the human operator.

The second step toward improvement of the adding machine would then appear to be that of adding to the device some means of holding a list of the operations to be performed (this list is often called a program), so that when the machine was finished with one operation it could go on to the next without waiting for the human operator to decide what to do next. The first machines to do this on a large scale were the card-programmed calculators. These devices had the ability to process numbers, store data and intermediate results, provide output as required, and accept their operating sequence (program) from a deck of punched cards.

One trouble with the deck of punched cards, though, is that it is impossible for the machine to change the sequence by itself. This must be done by the operator by altering the order of the cards. Also, cards, like operators, cannot work at the electronic speeds now possible. If some way of allowing the machine to change its sequence of operations by itself can be found, the machine can be made more versatile. This decision to change its own sequence might be based on some condition like the change of the sign of a result from positive to negative. Since the change of sequence does not require the handling of cards now, the change can be made much more rapidly.

This has been done with the type of digital computer which we will be studying from here on. This computer might better be described as a "general-purpose, automatically-sequenced, stored-program, electronic digital computer." The definitions of these modifiers might be:

general purpose - can do almost any numerical or logical problem of reasonable size;

automatically sequenced - performs operations one after the other without outside intervention;

stored program - stores the sequence of operations to be performed inside the machine before the operation of the program is started.

electronic - uses electron devices such as tubes or transistors; and
digital - processes discrete numbers rather than continuous quantities.

1.2 Internal Functioning

We have described the type of digital computer which we are talking about with the various modifiers given above. From here on we will, when we say "digital computer," we mean a "general-purpose, automatically ... etc...computer" and will not bother to write all of the modifiers. First we will describe the instructions which tell the machine what to do. Then we will consider the functions of the four major parts of the computer which the programmer is concerned with. These parts are the arithmetic unit, the memory, the input, and the output. (There is a fifth major part, the control unit which keeps the other four running properly, but the programmer is not concerned with this part.)

1.21 Instructions - Instructions are the orders which tell the computer what we want it to do with the numbers which we have given it. Instructions are stored inside the machine before the operation of the problem is started. Then these instructions are considered by the machine one by one and the data are processed accordingly.

It is important to realize at this point, though, that instructions are numbers. They look exactly like a piece of data which we may also store in the machine. The memory can hold only numbers, some of which may be data-type numbers and some of which may be instruction-type numbers. Since they look the same, we must be careful to keep instructions and data separate.

Instructions have two major portions. The first portion is a code, called the operation code, which tells the computer which operation is to be performed next, like "add." The second portion is the address portion. In the address portion are the numbers of the locations in memory where the data to be used in this operation can be found. The number of addresses which are placed in one complete instruction varies from computer to computer. We will be considering here only those machines which have one single address following the operation code. We will see how this is used later.

1.22 Arithmetic Unit - The arithmetic unit is a device which can do arithmetic operations on command. It can add, subtract, multiply, and divide, and it can also test the sign of a number. It cannot do anything like decoding or translating instructions, and if the programmer accidently sends an instruction to the arithmetic unit, it will consider this instruction as a number and operate on it, perhaps producing unplanned results. The arithmetic unit contains a device known as the accumulator. The accumulator is a register much like the lower dial of a desk calculator. It holds the number which is currently being processed. Numbers may be added to the number in the accumulator, subtracted from the number in it, and so on. The number in the accumulator may be tested to see if it is positive or negative, or the accumulator may be set to zero to clear it.

1.23 Memory - The memory is the portion of the computer where all numbers, which may be either data or instructions are stored. The memory is broken up into cells, often called registers or storage locations. A system of

of addresses is established which assigns a unique address to each memory cell. The cell address is a number. These address numbers are the computer's way of keeping track of the information stored in a particular cell. To refer to a particular piece of data, we instruct the computer to call at the address of that data. The memory cell which we have addressed is then connected so that we can get the number from the cell or place another number into it. Taking a number from a cell, usually called "reading," does not change the number which is stored in the cell, just as reading a book does not alter the printed word on the page. Placing a new number in a cell does change the number in the cell, since we cannot write two numbers in the same space.

1.24 Input and Output - Input and output devices are provided for getting data and programs in and out so that the computer can communicate with the outside world. There are two distinct types of inputs, however, which must be handled by the input device. The first is the program and some associated constants. The program must be loaded before it can be used. The second type of input is the input of data; this comes under the control of the user's program and reads into the machine whatever he desires. Output is generally entirely under the control of the programmer, but the loading of the program is not. We will speak more about this in the next section.

1.3 Computer Operation

The operation of the computer can effectively be divided into two very distinct parts, the loading phase and the operating phase. During the loading phase the program is placed in the machine; during the operating phase the program is followed step by step to produce the desired result (or at least a result).

Since instructions are stored in the memory, it is necessary to get these instructions into the memory before the program can be operated. This loading phase only loads the computer with instructions, which have been prepared on some medium which can produce electrical signals. Common program input media are electric typewriters, punched paper tape and cards, and magnetic tape. It is important to realize that during the loading phase no operation of any of the user's programmed instructions is carried out. The instructions are merely placed in the memory.

It is during the operating phase that the computer carries out the instructions which the programmer has written. These instructions may do all sorts of things, including the input of data and the output of results. Note that the data are read into the machine under the control of the user's program. Hence the data must not be placed in a position to be brought into the machine until the entire program has been loaded and set into operation.

2.0 BASIC PROGRAMMING OF THE LGP-30 FLOATING INTERPRETIVE SYSTEM

2.1 Introduction

The Floating Point Interpretive System (24.2) for the LGP-30 computer is designed to make the coding of problems for the LGP-30 as simple as possible and to assist the programmer in carrying out many of the things which are often confusing or difficult to do. The system is quite flexible and yet is very easy to program and operate.

The LGP-30 itself is a small, fairly inexpensive digital computer. It works in binary, and the numbers have a fixed point at the left end of the number. It uses an electric typewriter (Flexowriter) and a paper tape reader and punch for input and output. The Floating Point Interpretive System is a program written for the LGP-30 to enable it to understand an instruction code which is not its basic language. The program works in decimal, and the decimal point is not fixed in any one particular location in the number. The program interprets the instructions which it is given to carry on operations on these decimal numbers.

2.2 Available Functions

The Floating Point Interpretive System (24.2) provides for:

1. Program input
2. Data input
3. Data output
4. Basic orders for
   a. Floating point arithmetic
   b. Sin, cos, arctan, square root, a^b, e^x, 10^x, $\log_e$, $\log_{10}$
   c. Logical decisions
   d. Printing
   e. Input
   f. Index registers
5. Decimal memory printout
6. Trace routine
7. Alphameric output
8. Hexadecimal output

2.3 Memory

There are 1408 storage locations available to the programmer for the storage of instructions and data. These locations are divided into 22 tracks of 64 locations each. These locations are often called sectors. Each location is addressed by specifying a track number (from 40 through 61) and a sector number (from 00 through 63). Since instructions in this system require only one memory location each, they are stored and executed in sequential order. Numbers require two locations each, but they are always addressed by the number of the first of the two locations. Examples of consecutive locations are 4000, 4001, 4002, . . . 4061, 4062, 4063, 4100, 4101, 4102, . . .

2.4 Numbers

Floating point means that the numbers are not represented in the usual machine code which has the point fixed in a certain location but rather in a code which gives numbers in the form m x 10^n where m is a decimal integer and n is also a decimal integer giving the power of 10 by which m must be multiplied to produce the original number. Although the range would appear to be nearly unlimited, when the problems of input and output are considered, the system must be limited in this machine to powers of 10 within the range from -99 to +99.

These floating point numbers are stored in two consecutive memory locations. The first location contains the m part and the second contains the n part. However, the programmer need address the entire number only by giving the address of the first location of the pair. But all numbers, regardless of whether parts of them are zeros, take two locations.

The format for numbers on input is xxxxxxxx 'sees' where xxxxxxxx is a decimal integer with up to eight digits (leading zeros need not be written), the first s is the sign of the integer, ee is the decimal integer giving the power of 10 by which the first integer must be multiplied to get the orginal number, and the second s is the sign of the exponent. The apostrophe (') is a special signal to the computer and must be placed as indicated. This mark is called a "stop code" or "conditional stop" and must be used.

The format for numbers which are printed on output is the same, except that a decimal point replaces the first ', the signs are omitted if they are +, and the second ' is also omitted.

Examples of floating point format:

Number	Input Format	Output Format
2.0	2'+00+'	20000000. 07-
16.9	169'+01-'	16900000. 06-
-0.031	31'-03-'	31000000. -09-
15600.0	156'+02+'	15600000. 03-

For greater precision, numbers with less than eight digits may have zeros placed after them to fill all eight digits and have the exponent adjusted accordingly.

2.5 Instructions

Instructions are of the form Øttss where Ø is the operation code and ttss is the track and sector number of the operand involved (or a special code in some cases). When we consider the instructions themselves, it is handy to have some abbreviations for the quatities being processed. We will use xxxx to refer to a general four-digit track and sector number. We will use the symbol c() to mean "the contents of the location specified by the number in the parentheses." For example, c (xxxx) means the number in location xxxx, and c(a) means the number in the accumulator. On the next page are the basic instructions for the Floating Point Interpretive System (24.2). Other available instructions are discussed in chapter 3.

2.51 Arithmetic Instructions -

axxxx Form c(a) + c(xxxx) and leave in accumulator.

sxxxx Form c(a) - c(xxxx) and leave in accumulator.

mxxxx Form c(a) times c(xxxx) and leave in accumulator.

dxxxx Form c(a)/c(xxxx) and leave in accumulator.

rxxxx Form c(xxxx)/c(a) and leave in accumulator.

2.52 Sign Instructions -

b0000 Make c(a) have a positive sign.

t0000 make c(a) have a negative sign.

y0000 Change the sign of c(a).

2.53 Storage Instructions -

bxxxx Bring c(xxxx) into the accumulator without changing c(xxxx).

nxxxx Bring -c(xxxx) into the accumulator without changing c(xxxx).

hxxxx Store c(a) in location xxxx without changing c(a).

2.54 Functions -

r0000 Form the square root of c(a) and leave in accumulator.

s0000 Form sin c(a) and leave in accumulator.

c0000 Form cos c(a) and leave in accumulator.

a0000 Form arctan c (a) and leave in accumulator.

exxxx Raise c(a) to the power c(xxxx) and leave in accumulator.

h0000 Raise e to the power c(a) and leave in accumulator.

h0010 Raise 10 to the power c(a) and leave in accumulator.

n0000 Form the natural log of c(a) and leave in accumulator.

n0010 Form the base 10 log of c(a) and leave in accumulator.

2.55 Sequence-Changing Instructions -

uxxxx Take the next instruction from xxxx instead of from the location following the location of this uxxxx instruction.

txxxx If c(a) is negative, consider this as a uxxxx instruction. If c(a)

is positive or zero ignore this instruction.

z0000 Halt.

2.56 Input and Output Instructions -

ixxxx Receive data from tape and begin placing it in location xxxx and those following until the symbol f' is reached on the data tape. Then take the next instruction in sequence. Floating point numbers require two locations each.

pxxxx Print c(xxxx) as a floating point number on the typewriter with no control of the typewriter carriage such as tabs or carriage returns.

p0000 Print c(a) as a floating point number on the type writer with no control of the typewriter carriage such as tabs or carriage returns.

d0000 Perform a tab on the typewriter.

m0000 Perform a carriage return on the typewriter.

2.6 Programming Examples

We will assume here in these programming examples that the programs all start with their first instruction in location 4000. It is necessary in all programs using this interpretive system with the LGP-30 to have as the first two instructions in the program the following:

IN THE FIRST TWO LOCATIONS

r6300'u0400'

2.61 Example 1 - Assume that a number a is in location 5000, a number b is in location 5002, a number c is in location 5004, and a number d is in location 5006. We desire to form x = a + b + c + d and place the result in location 5008.

in location	4000	r6300'	special instruction
	4001	u0400'	special instruction
	4002	b5000'	bring a into the accumulator
	4003	a5002'	a + b
	4004	a5004'	a + b + c
	4005	a5006'	a + b + c + d = x
	4006	h5008'	store x in 5008
	4007	?0000'	stop the computation

2.62 Example 2 - We desire to read two numbers from tape and subtract the second from the first. If the result is positive, print the result. If the result is negative, change the sign of the result and print it preceded by a tab. After printing the desired number perform a carriage return and then read another pair of numbers and repeat the problem.

in location	4000	r6300'	special instruction
	4001	u0400'	special instruction
	4002	i4100'	read the first into 4100 the second into 4102
	4003	b4100'	bring the first into the accumulator
	4004	s4102'	first - second
	4005	t4008'	if +, continue; if -, go to 4008
	4006	p0000'	print positive result
	4007	u4011'	go to 4011 for a carriage return
	4008	b0000'	make the sign positive
	4009	d0000'	perform a tab

4010	p0000'	Print number
4011	m0000'	perform a carriage return
4012	u4002'	go back to read more numbers

The data tape in this example will have two numbers in floating point format followed by f'. Then there would be two more numbers, and so on. The computer is stopped by shutting off the reader so that no more numbers can be read in. Paragraph 2.71 shows an example of the program tape and the data tape for this problem.

2.7 Program Preparation - WRITE INSTRUCTION XXXX' hXXXX'

2.71 On the Flexowriter -

a-Press the Manual Input button on the computer down.
b-Put all switches in the top row of the Flexowriter up.
c-Turn the Connect switch on the Flexowriter off.
d-Turn the Flexowriter power on.
e-Press the Punch On switch on the Flexowriter down.
f-Feed about 6 inches of blank tape using the Tape Feed switch.
g-Type a carriage return (CAR RET key).
h-Type ;0004000'/0000000' followed by a carriage return. (This is a command to the computer to start loading your program into location 4000 when you actually begin the loading phase.)
i-Type the instructions of your program as written, each followed by '.
j-Place a carriage return on the tape after every ~~16th~~ 8TH instruction (hence before 00, 16, 32, and 48).
k-To correct an error, back up the punch to the wrong character by turning the knob on the punch away from you. One click will back up the punch to the last character punched, etc. Press the Code Delete button enough times to delete (punch out all holes) from that point on. Then type the correct characters.
l-At the end of the program, type a carriage return, followed by .0004000' (a command to the computer to tell it where you want it to start operating your program), followed by a carriage return.
m-Feed some blank tape.
n-If you want data on the same tape, type a carriage return followed by the data. Don't forget the necessary f' codes.
o-When done making tape, feed about 6 inches of blank tape.
p-Tear off the tape against the tear bar.
q-Raise the Punch On switch.
r-Proofread your tape as shown in paragraph 2.73

The following is the tape for example 2 with two sets of data following:

```
;0004000'/0000000'
r6300'u0400'i4100'b4100's4102't4008'p0000'u4011'b0000'd0000'p0000'm0000'
u4002'
.0004000'
12'+01-'68'+01-'f'16'-00+'12'+00+'f'
```

2.72 On the Typewriter - (Not Available at Wayne State University)

a - Turn on the power

b - Press tape Feed on the keyboard and the silver key sticking out of the front of the cabinet above the keyboard to feed about 6 inches of blank tape. If the tape does not feed properly, pull gently on the tape as it feeds to get the feed holes in step.

c - Follow steps g through p of paragraph 2.71.

d - Turn off the Teletypewriter.

e - Proofread your tape on the Flexowriter as shown in paragraph 2.73.

2.73 To proofread a tape on the Flexowriter -

a - Press the Manual input button on the computer down.

b - Put all switches in the top row of the Flexowriter up.

c - Turn the Connect switch on the Flexowriter off.

d - Turn the Flexowriter power on.

e - Load the tape into the reader with the printing on the tape aligned the same way that it is in the punch. The tape hold-down on the sprocket wheel is opened by pulling the tab forward. The tape slides from the left side under the reader head, under the open gate, and under the small flat hook at the rear on the right. Be sure that the pins on the feed sprocket are through the feed holes on the tape before closing the tape hold-down. Let the rest of the tape fall through the space between the reader and the punch.

f - Press the Cond Stop switch on the Flexowriter down.

g - Press Start Read.

h - At the end of the tape, press Stop Read.

i - Raise the Cond Stop switch.

j - Advance the platen by hand and tear off the typed copy.

k - Remove the tape from the reader by opening the gate and sliding it out.

l - If necessary, reproduce and correct the tape as shown in paragraph 2.74.

2.74 To reproduce and correct a tape on the Flexowriter -

a - Follow the steps for proofreading a tape, except that the punch must be on and you should feed about 6 inches of blank tape at the beginning and end of the tape.

b - While the typewriter is typing the word just before the incorrect word, raise the Cond Stop switch. The tape will stop on the stop code just before the incorrect word.

c-Type the correct word or words. Then advance the tape in the reader to skip the incorrect characters by counting the number of wrong characters to be skipped and then advancing the tape just that many holes. Do not count deletes (all six holes punched) as lines skipped. If a word has been omitted from the original tape, it is not necessary to skip any words on the tape.

d-Run a proof copy when the corrections have been made.

2.8 Program Operation

2.81 To turn the computer on (if it is not on) -

a-Press the Manual Input switch on the computer down.

b-Press the Operate switch on the computer down.

c-Press the Power On button on the computer

d-Wait for the green light under the Operate switch.

2.82 To load and run a program -

a-If the computer is not on, see paragraph 2.81.

b-Make sure that the Manual Input switch on the computer is down. and on Flexo

c-Put all switches in the top row on the Flexowriter up, except Man Input

d-Turn the Flexowriter power on.

e-Turn the Connect switch on the Flexowriter on.

f-Load the tape into the reader (see paragraph 2.73e). PR

g-On the computer, press One Operation, Clear Counter, Normal, Start. Comp. on Flex
g-1: lift up Man Input on Flex

h-When the tape reaches .0004000 it will stop.,

i-If the data tape is separate, load ~~it~~ into the reader. depress "Man Input" on Flexo DATA TAPE
i-1: lift "Man Input" Lever on Flexo

j-Press the Start Comp ~~button~~/on the Flexowriter to start the program. LEVER

k-If it is necessary to stop the tape, use the Man Input button.

2.83 If the program fails to operate all the way through -

a-Press the Transfer Control button on the computer down.

b-Press the Start button on the computer.

c-The location of the instruction either on which the computer stopped or the one following will be printed, along with the instruction in that location and the number in the accumulator at the time of printing. This will give you information on where the computer was in your program at the time that it ran into trouble.

d-Stop the printing by pressing the One Operation button on the computer.

e-Raise the Transfer Control switch by pressing it again.

2.84 To leave the computer -

a-Press the Manual Input switch on the computer down.

b-Turn the Connect switch on the Flexowriter off.

c-Turn the Flexowriter power off.

d-Place all switches in the top row on the Flexowriter up.

e-Write your time in the log.

2.85 To turn the computer off -

The computer is never turned off if it will be used within the following two hours. It is also not turned off before 5 pm on week days. It is never put into Stand By Operation. To turn the computer off:

a-Press the Manual Input switch on the computer down.

b-Press the Power Off button.

c-Turn off the Flexowriter

3.0 ADVANCED PROGRAMMING OF THE LGP-30 FLOATING POINT INTERPRETIVE SYSTEM

3.1 Introduction

The Floating Point Interpretive System (24.2) has been provided with many additional features which were not discussed in the previous sections because they are not of interest to the beginning programmer. It is not long, however, before the programmer begins to realize that he cannot do things that he wants to do with the system as it has been presented so far. One of the first things which he has trouble doing is running programs which require access in some sequential order to different locations in memory; operations with determinants or matrices are examples of this type of problem. The input and output for the systems are more flexible than has been described, and it is possible to print alphabetic characters as well as fixed and floating point numbers. Aids to the programmer are available which will assist him in operating his program and in determining what happened when it fails. These include tracing, address searching, and memory printouts.

Some users of the Floating Point Interpretive System become concerned with the length of time each instruction takes, so a table of operating times is included here. Since the programmer will find that these times are rather long, suggestions for use of the much faster basic machine language are made where they may be of use in speeding up the operation of programs. For the user who would like to combine operations in basic (machine-language) fixed point and in floating point, the layout of the memory and the arrangement of the various types of numbers and registers is given.

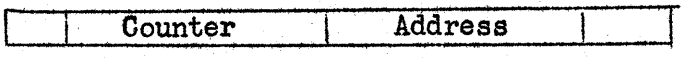
3.2 Index Registers

3.21 Looping and Address Modification -

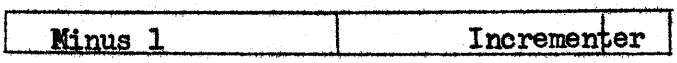
So far, looping (repeating a section of coding by transferring from the end to the beginning of it) has been accomplished by simply writing a conditional or unconditional transfer at the end of the loop. There is a much easier method (although somewhat slower) using the index registers. It is possible to write a single command which will count the number of passages through a loop and transfer out of the loop when the desired number has been reached. These same index registers may be used for modifying addresses of instructions as stored in memory. This makes it possible to write loops in which some instructions refer to a different place in memory each time the loop is traversed.

3.22 Parts of the Index Register -

There are eight index registers available in this system, numbered from 1 to 8. They are all the same and may be used interchangeably. Each index register has four parts, shown in the diagram below.



q=3 q=11 q=29



q=11 q=29

(The values of q are given for those working in basic machine language). The counter is the section of the index register which keeps track of the number of passages through a loop. It can be set to any desired value. The minus one is used with this counter to reduce the counter value by one during each passage. The programmer cannot change this minus one. The address is the number which may be added automatically to the address of any instruction to get a new address for the instruction. This addition takes place only when the instruction itself is called for during the operation of the program, and takes place just before the instruction is carried out. The instruction as it stands in memory is unchanged. The incrementer is the value which is added to the address value in the index register each time the loop is traversed. All three parts of the register must be set by the program before it can be used even if only one part is to be used.

3.23 Instructions for Using index Registers -

There are four basic orders for loading, incrementing, and texting the index registers. In describing these commands, the following abbreviations will be used:

n The number of the index register from 1 to 8.

a_n The address stored in the nth index register.

i_n The incrementer in the nth index register.

c_n The counter value in the nth index register.

All values set into the index registers must be written in the same track and sector notation which we use for regular addresses. Although this is normal for addresses, this must also be done for the incrementer and the counter. For example, if a counter value of 100 is desired (to traverse a loop 100 times, for example), the number to be loaded into the counter is 0136, meaning 1 full track of 64 locations plus 36 more. The instructions for using the counters are as follows:

ncxxxx Set the counter value of the nth index register (i_n) to xxxx. The maximum value in track-and-sector notation is 3163.

nixxxx Set the incrementer of the nth index register (i_n) to xxxx.

nexxxx Set the address value of the nth index register (a_n) to xxxx.

nzxxxx Increase a_n by the amount i_n, decrease c_n by one, and test c_n. If c_n is not zero, take the next instruction from xxxx. If c_n is zero, take the next instruction in sequence.

The incrementer may be used as a decrementer (to reduce the address value each time) by setting it to the desired number of tracks and sectors subtracted from track 63 sector 64. For example

	Using index register 2	
Amount	Increment	Decrement
2 locations	2i0002	2i6362
60 locations	2i0060	2i6304
70 locations (1 track, 6 sectors)		2i6258

3.24 Instructions Modified by Index Registers -

Most of the 24.2 instructions may have their address modified by simply placing the number of the index register desired in front of the instruction letter. These may be defined as follows:

n$\emptyset$xxxx Perform the indicated operation ($\emptyset$) on location xxxx + a_n.

n$\emptyset$0000 Perform the indicated operations ($\emptyset$) on location a_n.

The operation ($\emptyset$) may be any of the following instructions:

a	Add	b	Bring
s	Subtract	n	Negative bring
m	Multiply	h	Hold
d	Divide	P	Print
r	Reciprocal Divide		

There are several other instructions which are different from their unindexed counterparts:

nyxxxx Place a_n in the address portion of the instruction in xxxx.

ny0000 Illegal - causes machine to stop.

nuxxxx Set a_n to the location of the nuxxxx instruction and transfer to xxxx.

nu0000 Illegal - causes machine to stop.

ntxxxx Take the next instruction from xxxx + a_n. Notice that this is an unconditional transfer.

nt0000 Take the next instruction from location a_n.

There are several differences to be noted when using indexed instructions.

1) The address xxxx and the address 0000 do not make different instructions like they did without index registers. For example axxxx means add, a0000 means $\tan^{-1}$, but naxxxx and na0000 both mean add.

2) The conditional transfer is nzxxxx when using index registers, and the unconditional transfer is done with the nt command.

3) The input command is not indexable.

4) None of the algebraic functions may be indexed.

3.25 Example Using Index Registers -

Suppose that there are 100 numbers starting in location 5000 and we wish to get the sum of these and place the sum in location 4962. A possible program is

in location 4000	r6300'	Special Instruction
4001	u0400'	Special Instruction
4002	3c0136'	Set counter value to 136
4003	3e0000'	Set address value to 0
4004	3i0002'	Set incrementer to 2
4005	b4962'	bring the sum so far
4006	3a5000'	add next number
4007	h4962'	store new sum
4008	3z4005'	test counter - if not done, loop
4009	z0000'	done - stop

Note: It has been assumed here that location 4962 was set to 0 before the program was operated. This must be done by the programmer.

3.26 Basic Language Approaches to Indexing - The index register commands of the 24.2 system are very slow. The programmer can, with only a little extra work, learn the basic machine language to perform the indexing operations in fixed point. These operations in general consist of adding to a fixed location chosen as a counter and adding to the addresses of instructions each time through a loop. The effect of this is to speed up the operation of the indexing portions of the program by a factor of more than 50. The basic machine language will not be discussed here, by the serious programmer who has a long problem to do will find it well worth his time to learn this portion of basic machine language.

3.2 Input and Output

3.21 Coding Sheet Data Entry - The programmer often has several constants which he wishes to use in his program but which he rather would not have to enter as data using the data input command (i). The following command permits him to enter special constants at the same time that he loads his program:

cxxxx Convert the specially-coded fixed point number in location xxxx into floating point and place it in the floating point accumulator.

Notice that this is the equivalent of the bring instruction except that it applies only to specially-coded numbers.

The format for the coding sheet is xxxxxses' where xxxx' is a five digit decimal integer, the first s is the sign of the integer, e is the decimal integer giving the power of 10 by which the first integer must be multiplied to get the original number, and the second s is the sign of the exponent. The stop code appears only after the entire number. Examples of these numbers are:

Number	Format
53.216	53216+3-'
-5468000	54680-2+'

The number may be placed anywhere on the coding sheet where it is convenient to place constants. However, it must be entered as a hexadecimal word. This simply means that the number (Or a group of them) must be preceded by the hexadecimal word code ,00000nn' where nn is the number of these specially-coded constants to follow. For example, the two numbers given in the example above would be written on the program tape as follows:

,0000002'53216+3-'54680-2+'

Note that the hexadecimal word code requires no space in memory and that each of the specially-coded words requires only one space each. The only floating point command which can be used to refer to these constants is the c command. If other commands must also use them, the constants should be converted by using the c command to get them into the floating point accumulator and then holding them into an area where there is space for regular floating point numbers which take two locations each.

3.22 Programs for General Locations - Programs written so far have been written for a particular place in memory. This is not necessary and often not desirable since the programmer may want to rearrange the memory at a later time. If the program is written as a subroutine to be used by many other programmers, the program should be flexible enough to be locatable anywhere in the memory.

The programmer may do this with any program by making use of the address modification feature of the Input Program. The programmer can write a program as if it were to be placed in locations from 0000 on. He can then cause a constant to be added to the address of every instruction in his program as it is loaded into the machine. This in effect modifies the program so that it now appears to be written for some other area. The rules for doing this are as follows:

1) Write the program as if it were working in locations starting at 0000.

2) Place an x before any instruction which is not to have its address modified. (For example, r6300 is to refer to 6300 no matter where the program is located; therefore it would be written xr6300.)

3) When loading the program into, say, location 4000, precede the program with ;0004000'/0004000' and then type the program. The number after the / is called the set modifier' and is added to the addresses of all instructions not preceded by x. (It does not affect anything except instructions.)

3.23 More on Data Input - The data input command ixxxx is somewhat restricted in that data must go into a particular place in memory assigned by the program. Another input command is available which allows the assignment of the starting location of the data to be made on the data tape:

i0000 Receive data from tape and begin placing them in memory starting with location given as the first four digits on the tape. Continue in consecutive pairs of locations until either g' or f' is reached. If g' is encountered, continue loading, but start a new sequence of consecutive pairs of locations starting with the one given as the next four digits on the tape. If f' is encountered, stop loading and take the next instruction in sequence in the program.

For example, to load the numbers 33.1 and 142 into locations 5000 and 5002 and the number 0.0067 into location 5216, the program would contain the command i0000 and the data tape would be:

5000 '331' +01- '142'+00+'g'5216'67'+04-'f'

Accuracy of the data input may be improved somewhat by representing numbers with as large an integer as possible. For example, the best way to represent 4.0673 is to write 40673000'+07-' rather than 40673'+04'. Numbers are always printed out in the larger form.

Typing time for data can be shortened by omitting leading zeros. This is usually done when typing the integer part of the number, but it can also be done with the exponent if the programmer remembers that the + sign may be considered as a leading zero. Hence +06-' can be written 06-' or even as 6-'. If the exponent is zero, it may be omitted entirely, but the stop code following it must be printed. If the sign of the integer is negative, however, the entire exponent must be printed.

It is sometimes convenient not to have to take the next instruction in sequence following an ixxxx or i0000 command but to transfer to some other location depending on the data set which was fed in. This may be done by writing, on the data tape in place of f', a transfer command uxxxx'. This will cause the program to take the next instruction from xxxx after reading in the set of data.

3.24 Compatible Output - The regular print commands pxxxx and p0000 print in a non-compatible format; i. e., in a format which is not like the input format. Therefore if output data is to be punched on tape the format is incorrect for use as input by another program. The following print commands print in a format exactly like the data input format (see section 2.4):

800pxxxx Print c(xxxx) in compatible format.

800p0000 Print c(a) in compatible format.

If the print command is used with the x to prevent input address modification or with the index register number n or with both, the forms are 80xpxxxx, 80npxxxx, and 8xnpxxxx. As with the non-compatible print commands, no page format control is exercised over the typewriter such as tabs or carriage returns.

3.25 Fixed Point Output - Data may be printed in a fixed-point format with no exponent and with the decimal point properly located:

zCnnn Print c(a) as a fixed point number with no more than nnn digits after the decimal point. Print only enough zeros to locate the decimal point.

zxxxx Print c(xxxx) as a fixed point number with eight significant figures and only as many zeros as are necessary to locate the decimal point.

Examples of the use of this print command are

c(a)	Command	Output
12345678'+06-'	z0000	none-machine stops.
12345678'+06-'	z0003	12.345
12345678'+06-'	z0006	12.345678
12345678'+06-'	z0022	12.345678
12345678'+07+'	z0000	123456780000000.
12345678'+20-'	z0018	.00000000000123456

The zxxxx command always produces eight significant digits.

The number nnn is counted by tracks and sectors. For example, if 63 digits are desired, write z0063, but if 64 digits are desired, write z0100 (i.e., one track of 64 sectors plus no additional sectors).

3.26 Alphameric Output - Alphabetic characters, digits, symbols, and typewriter control commands may be printed out using the alphameric print commend:

u0000 Print the codes in the following locations in the program as alphameric characters until the code 00 is reached. Then take the next word following this as an instruction.

Each location after the u0000 command must contain four alphameric codes as listed below. The last location must have at least one 00 code and may be filled with other 00's to obtain four codes. These codes are considered as hexadecimal words and must be preceded by the hexadecimal word code ,00000nn' where nn is the number of locations occupied by these alphameric codes. For example, a program which is to print The End would be u0000', 0000003'20t410h4'e4zj20e4'10n4d400' and the typewriter would execute upper case, T, lower case, h, e, space, upper case, E, lower case, n, d, and then control would be returned to the instruction right after the last code word. Note that the typewriter must be placed in upper and lower case when desired.

Character desired		
Upper case	Lower case	Code
A	a	a4
B	b	b4
C	c	c4
D	d	d4
E	e	e4
F	f	f8
G	g	g8
H	h	h4
I	i	i4
J	j	j8
K	k	k8
L	l	l8
M	m	m4
N	n	n4
O	o	oj
P	p	p4
Q	q	q8
R	r	r4
S	s	s4
T	t	t4
U	u	u4
V	v	vj
W	w	w8
X	x	xj
Y	y	y4
Z	z	z4

Character desired		
Upper case	Lower case	Code
)	0	08
L	1	18
*	2	28
"	3	38
Δ	4	48
%	5	58
$	6	68
π	7	78
Σ	8	88
(	9	98
:	;	;j
?	/	/j
⌋	.	.j
⌊	,	,j
=	+	+j
—	-	-j
	Space	zj
	Delete	wj
	Lower case	10
	Upper case	20
	Color shift	30
	Carriage return	40
	Backspace	50
	Tab	60
	Stop code (')	80

3.4 Miscellaneous Features

3.41 Trace Routine - A trace routine is provided to assist the programmer in checking out his program when he is testing it. The trace routine will provide a complete record of what is happening in the machine by printing the location of the instruction just performed, the instruction itself, and the contents of the accumulator after the instruction has been executed.

To initiate tracing, depress Transfer Control. The trace routine will begin with the instruction just executed and will continue from that point on. The Transfer Control may be pressed while the program is operating or when it is stopped (if it is stopped, start it again by pressing start). If the program stopped because of an illegal operation, it will print the instruction on which it stopped (or in some cases the one following). To stop tracing, raise the Transfer Control button by pressing it again.

The number in the accumulator must be a proper floating point number before the trace routine will print it. Improper numbers can be present during initial input instructions and as a result of machine operations in basic language. The trace routine will stop after printing the instruction if an illegal number is present. Merely press the Start button to make it continue.

3.42 Decimal Memory Printout - The decimal memory printout is another aid to the programmer for it enables him to get a picture of what is in a section of memory. This is valuable to check on the correctness of input operations and to see what the program has stored in a region if the program should fail for some reason. The decimal memory printout will print as follows:

1) Instructions - Complete instruction including index register number (if any), but with the address unmodified by the index register.

2) Floating Point Numbers - The number with its exponent.

3) Anything else - Printed as hexadecimal numbers if the number is not recognized as an instruction or a floating point number. Some numbers which the programmer wrote as hexadecimal words may be recognized as instructions or floating point numbers. The digits of the hexadecimal number will always be written in the standard hexadecimal code (see the basic language programming manuals).

To operate the decimal memory printout:

1. Depress Man in on the Flexowriter.

2. Press One Operation, Clear Counter, Normal, Start.

3. When the light on the Flexowriter comes on, type .0003300' and press the Start Comp button twice.

4. When the light comes on again, type, as a single eight-digit number, the decimal value of the initial and final locations of the block of memory to be printed. (For example, to have the section from 4000 through 4015 printed, type 40004015'.)

5. Press Start Comp.

6. When the printing is finished, step 4 may be repeated.

7. The contents of the index registers may be printed by giving the range as zero. (Nothing need by typed; just press the Start Comp button for Step 4). The index registers are printed in order, 1-8, giving counter value, address value, and incrementer.

3.43 Search for Address - The address search makes it possible for the programmer to find out if any instruction in the region from 4000 through 6163 has a certain address in it. The program will look through this entire region for a particular address. If any instruction is found which refers to that address, the program will print the location of the instruction followed by the instruction letter. It will then continue searching the rest of the region. (Note that it is possible for hexadecimal numbers to look like instructions now and then.)

Operation is as follows:

1. Depress Man In on the Flexowriter.

2. Press One Operation, Clear Counter, Normal, Start.

3. When the light on the Flexowriter comes on, type .0003700' and press the start Comp button twice.

4. When the light comes on again, type the four decimal digits of the address to be searched for.

5. Press Start Comp.

6. The routine will print locations if any are found and will finally stop. If it is desired to continue with a new address, depress Transfer Control and press Start. Then repeat step 4. If Transfer Control is not depressed, the routine will reduce the address to be searched for by one and will continue when the Start button is pressed.

3.44 Hexadecimal Punch - The hexadecimal punch is used to punch in hexadecimal form on tape a program already stored in the machine. This may be used when the program has been designed for use many times in the future, since it provides a slightly faster input and has a check sum on the end of the tape to check the accuracy of input. The hexadecimal punch routine is generally not in the computer because it must be placed in the same area that the fixed point print routine and the address search routine occupy. However, it is the same as the LGP-30 Subroutine J4-13.2M1 which is on file in the computer room and which may easily be used when a hexadecimal program tape is desired. The prospective user should consult the subroutine write-up file.

If the floating point hexadecimal punch is in the computer, the operation is the same as for the decimal memory printout except that the starting location is .0003600'.

3.5 Test Program for 24.2 - There are times when it is desirable to check the entire 24.2 program to see if everything is written correctly on the drum. Operations which make it necessary to reload the program or parts of it, power line transients which may have changed part of the program, and trouble with the program itself make this text necessary. A test program, tape H1- 24.2C, is available to test the entire program. It operates by checking four tracks at a time. The program occupies most of tracks 60, 61, and 62, but may be relocated if necessary.

The operation is as follows:

1) Place the H1-24.2C tape in the reader.

2) Place all switches in the top row of the Flexowriter up.

3) Turn on the Flexowriter power and connect switches .

4) Press One Operation, Clear Counter, Normal, Start.

5) The tape will read in most of the way and stop after printing .0006000'. Press Start Comp.

6) Do not remove the tape from the reader until the entire testing operation is finished.

7) The numbers of the groups in error will be printed. These numbers represent the section of the 24.2 program tape which must be read in again to replace the faulty area. You will have to get help to reload 24.2

8) If no errors are found, the program will print 'none' and stop.

3.6 Comments on Speed

3.61 Interpretation

The Floating Point Interpretive System is a program which takes the programmer's instructions and decodes them under program control to cause the desired operation. In other words, the instructions are not being interpreted electronically as the basic machine language instructions are. This decoding operation takes time, since it takes a certain amount of time to decode machine instructions electronically and it takes many machine instructions to decode one floating point instruction. Therefore, interpretive systems are inherently slow. For this loss of speed, however, the programmer often gains in ease of programming. He is essentially taking less of his time to program but is taking more of the machine's time. There is a certain break-even point where reducing the programmer's time still more will raise the cost of the entire operation by taking too much machine time. It is difficult to say where this point is because the various costs are not known, but the programmer with any large job to do on the computer should consider carefully the amount of machine time that may be required to enable him to reduce his programming time.

3.62 Use of Basic machine Language

The basic machine language is similar to the floating point language except that it deals with fixed point numbers and works entirely in binary. Input and output routines are available to convert between decimal and binary so that the programmer does not have to do this. There are three important facts to be considered when deciding whether to use floating point or basic language:

1. The slowest basic machine operation speed is 50 operations per second compared with the fastest interpretive system operation speed of 2 operations per second.

2. Logical operations such as counting, looping, and printing are much more easily done in basic machine language. Moreover, some logical operations are nearly impossible to do in 24.2.

3. The programmer may have to spend some extra time and effort learning the basic language to be able to use it effectively, and there are many more idiosyncracies in basic language that there are in 24.2.

3.63 Mixed Usage of Languages

Some programs may be effectively written by using floating point for the numerical operations and basic language for the logical operations such as looping and counting. The effect depends on the percentage of these two kinds of operations in a given program, but at least some machine time is saved. The floating point interpretive system may be entered at any point in a program merely by writing r6300'u0400'. Instructions which follow will be operating as interpreted instructions. To get back out of the floating point operation, the following instruction is used:

e0000 Take the instructions following this as a basic machine language instructions.

3.7 Space Required for 24.2

The 24.2 Floating Point Interpretive System occupies a block of 40 tracks within the machine. These tracks are electrically interlocked so that it is impossible to write anything onto them and destroy the floating point program. The program is broken up into four tapes, called H1-24.2A, H1-24.2B1, H1-24.2B2, and H1-24.2B3. (Tape H1-24.2A is sectioned into groups for ease of reloading.) Either H1-24.2B2 or H1-24.2B3 is in the memory at any one time, not both. The memory is divided as follows:

H1-24.2A	Program Input Routine	0000-0363
	Floating Point Interpretive Routine	0400-1763
	Floating Point Data Input Routine	1800-2163
	Floating Point Data Output Routine	2200-2563
	Logarithm	2600-2663
	Exponential	2700-2839
	Arctangent	2840-2963
	Sine-Cosine	3000-3163
H1-24.2B1	Alphameric Output Routine	3200-3263
	Decimal Memory Printout Routine	3300-3463
	Floating Point Trace Routine	3500-3563
H1-24.2B2	Addition to Decimal Memory Printout and Address Search Routine	3600-3763
	Unfloat and Print Routine	3800-3963
H1-24.2B3	Hexadecimal Punch Routine	3600-3963

In addition to the space above, the 24.2 system requires tracks 63 and 62 for temporary storage. The only locations which may be of interest to the programmer who may combine floating point and basic language are those of the floating point accumulator. The fraction is usually in 6258; the exponent is 6205. See section 3.8 for a further description of the numbers.

3.8 Binary Representation of Floating Point Numbers

Although numbers are entered into the machine in decimal, they must of course be converted to binary to be stored and used. The number is stored in the form $x(2^Y)$ where x is a binary fraction and y is the binary exponent. The binary fraction is stored in the first of two locations with a q of 0; the exponent is stored in the second location with a q of 29. (See the basic language manuals for a discussion of q.) The fraction is always shifted so that the first bit on the left is a one. This normalization is done whenever a number is brought into the floating point accumulator. For example, the number 3.75 would be stored as follows:

$$3.75 = 0.9375\ (2^2)$$

Fractional Part

0.111100000000000000000000000000

Sign of fraction 0.9375 @ q = 0

Exponent

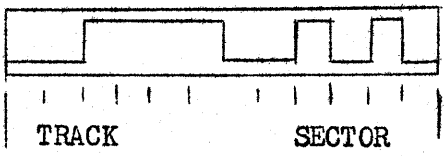
0.000000000000000000000000000100

Sign of exponent 2 @ q = 29

3.9 Programmed and Error Stops

There are various operations in floating point which are not legal for some reason or another. For example, attempting to take the square root of a negative number is not permitted. Each of these illegal operations leads to a machine stop. By reading the counter of the machine, the programmer can determine what illegal operation he attempted to perform.

The program counter is the top window of the oscilloscope on the console of the computer. It may be read as follows:



1. Read only the horizonal lines: if at top, one; if at bottom, zero.
2. Read from left to right, reading track and sector separately.
3. Assign values to the ones according to the values given in the diagram.
4. If necessary, simply read off the sequence as a series of 12 binary digits (0's and 1's).

For example, the pattern shown above would be read as 001111001010 in binary and as

$$1(2^3) + 1(2^2) + 1(2^1) + 1(2^0) = 8 + 4 + 2 + 1 = 15$$

$$1(2^3) + 1(2^1) = 8 + 2 = 10$$

so the result is 1510.