Royal Precision Electronic Computer

LGP-30

SUBROUTINE MANUAL

Royal McBee Corporation

LGP - 30 SUBROUTINE MANUAL

This subroutine manual was compiled by the Royal McBee Computing Section to facilitate the coding of problems for the LGP-30. The programming was done by the Royal McBee and Librascope Computing Sections This subroutine manual, in its present form, is considered to be complete.

The subroutines provide for a variety of operations whose major types are as follows:

a. Machine Input and Output.
b. Evaluation of Elementary Functions.
c. Program Check-Out.
d. Complex Operations.
e. Floating Point Operations.

For each subroutine the calling sequences, running time, and storage requirements are given. Also where applicable, the range of variables, scaling, and accuracy are given.

It is hoped that as LGP-30 users develop other useful subroutines, they will submit them to the Royal McBee Computing Section. Synopses of these subroutines together with those developed by the Royal McBee Computing Section itself will appear regularly in the LGP-30 Newsletter. The Newsletter is distributed monthly to all LGP-30 users. This arrangement will facilitate prompt distribution of routines and will prevent unnecessary duplication.

Royal McBee Corporation
1560 North LaBrea Avenue
Hollywood 28, California

Copy No. 383

- CONTENTS -

DEFINITIONS

1. A routine is a logical subdivision of a program, complete in itself, and serving a specific function in the problem. There is no fixed length to any routine, and each routine occupies only as much storage as is actually needed.

2. A subroutine consists of a set of instructions to perform a standard task which is of a sufficiently general nature to be used in a number of different programs. Examples are subroutines to input and output data, compute square roots, arctangents, etc. This Subroutine Manual is a compilation of the specifications of the subroutines, completely describing the function and use of each.

3. A calling sequence is a set of instructions used for transferring from the main routine to a particular subroutine. It may also include information needed by the subroutine, such as constants and the locations of certain quantities. The calling sequence for each of the subroutines is given in the Subroutine Manual.

4. Minimum Time Programs

There are occasions when it is necessary to write programs which will be executed in as little time as possible. These minimum time programs are referred to as "optimum" programs. Since the subroutines contained in the Manual are to be used over and over again, they have been optimized. (The process of optimizing requires placing the sector of the operand at α + (7k+1) where $2 \leq k \leq 6$ for most instructions). The programmer should bear in mind that 10,000 executions of all nonoptimum instructions would take less than 3 minutes longer than 10,000 executions of optimum instructions. If the programmer spends 15 - 30 minutes on each routine trying to save machine time by optimizing, this time may never be made up in the actual running of the problem.

5. A scale factor of a scaled number in memory is defined as the power of 2 by which this scaled number must be multiplied to get the original or unscaled number.

CONVENTIONS USED IN THIS MANUAL

1. α is the base memory location from which entry to a subroutine is executed. Locations used by the subroutine are in reference to location α . E.G., $\alpha + 1$, $\alpha + 2$, $\alpha + 3$.....

2. L_o designates an initial location. L_f designates the final location.

3. The "Stop" and "Stop Codes" referred to in these write-ups and on the coding sheet are synonymous with "Conditional Stop Code".

4. For explanation of our scaling convention, see write-up on "SCALING".

5. Track 63 is used by some of these subroutines for temporary storage. The track 63 sectors used by the subroutines are enumerated in the respective write-ups. This practice was found useful for "optimum" programming of subroutines. However if the subroutines which use this temporary storage are to remain optimum, the L_o of the subroutine must be the beginning of a track. It is suggested that the programmer may also use track 63 for temporary storage of intermediate calculations. He should not place a number in a track 63 location used by one of these subroutines and expect that number to be there after exit from the subroutine.

PUNCHING TAPES FROM CODING SHEETS

See "Sample Program" page for example of coding sheet.

1. Only the "Program Input Codes" and "Instruction" columns of the coding sheet are to be punched, with appropriate stops. Never punch "Location", "Contents of Address", or "Notes" columns.

2. Each entry on a line must be followed by a conditional stop code--"Stop" column, symbol ('). A line left blank must have the stop code punched.

3. Punch the "Program Input Codes" column only when there is an entry in the column. The "Program Input Codes" must be followed by the stop ('). This punching must precede the punching of the "Instruction" column on the same line of the coding sheet.

4. Leading zeros need not be punched. All other zeros must be punched. E.G., 00013086'only 13086' need be punched. ,0000017' must be punched ,0000017'. For T0059' punch T0059'.

5. Consider brackets as containing zeros. E.G., for [...!...]' = [00000000]', only the stop code need be punched. For B[....]' = B[0000]' punch B0000'.

6. All punching may be done in lower case. B0627' will appear as b0627'.

7. The placing of carriage returns is left to the discretion of the person preparing the tape. Carriage returns do not affect the input operation. We have arbitrarily placed a carriage return (⊠) after every 4 words on each coding sheet.

8. A heading may precede a punched program to identify the tape. Anything except a stop code may be punched as a header. Then as the tape is fed through the input reader the heading will print but will not affect the operation of the computer.

9. Each tape should be verified after punching. This can be done by placing the punched tape in the reader and "listing" the tape by the following process.

   a. Depress the "Cond. Stop" button on the Flexowriter.
   b. Depress "Start Read" button.
   c. When printing stops, depress the "Stop Read" button on the Flexowriter.

Then the printing may be visually checked against the coding sheets for correctness and presence of stop codes.

10. It should be the programmer's responsibility to enter "Program Input Codes" (and the associated stop codes) on the coding sheet. This will usually consist of a start fill (;), a set modifier (/), and possibly some hex. words (,) and/or stop and transfer (.) codes.

SAMPLE PROBLEM
LGP-30 CODING SHEET

Job No. XXX Prog. No. YY.Y Prep. by [illegible] Ck'd. by GLW Date 8-1-57

Problem EVALUATION OF 4TH DEGREE POLYNOMIAL (FIXED POINT) Track 10

Program Input Codes	Stop	Location	Instruction Op.	Address	Stop	Contents of Address	Notes
;0,0,0/1,0,0,0	'						
/1,0,0,0/1,0,0,0	'	☒					
		1,0,0,0	xR	0,5,0,8	'	} Transfer to Data Input #1 to read,	
		0,1	xU	0,5,0,0	'	} convert, scale, and store data in memory	
		0,2	B	0,0,2,3	'	A0026	Initial "ADD" Instruction
		0,3	C	0,0,0,7	'	☒	
		0,4	H	0,0,2,5	'	Set working storage to zero	
		0,5	B	0,0,2,5	'	Working storage at q=0	
		0,6	M	0,0,2,6	'	X @ q=0	
		0,7	A	[ ]	'	☒ a_n @ q=0	
		0,8	H	0,0,2,5	'	Working storage @ q=0	
		0,9	B	0,0,0,7	'	A(0027+n)	
		1,0	A	0,0,2,2	'	1 @ 29	
		1,1	Y	0,0,0,7	'	☒ A(0027+n+1)	
		1,2	S	0,0,2,4	'	A0032	Flag
		1,3	T	0,0,0,5	'		
		1,4	B	0,0,2,5	'	Final Result	
		1,5	xR	1,4,1,2	'	☒ } Transfer to Data Output #1	
		1,6	xU	1,4,0,0	'	} to print final result.	
		1,7	xZ	0,0,0,0	'	} Code word = 0 (for q=0)	
		1,8	xP	1,6,0,0	'	} Carriage return	
		1,9	xZ	0,0,0,0	'	☒	
		2,0	xZ	0,8,0,0	'	Stop unless brk. pt. sw. #8 is down	
		2,1	U	0,0,0,0	'	Return to read more data	
		2,2	xZ	0,0,0,1	'	1 @ 29	} Constants and flags
		2,3	A	0,0,2,6	'	☒ A(L_0)	
		2,4	A	0,0,3,2	'	A(L_f+1)	
		2,5	[	]	'	Working storage @ q=0	
		2,6			'	X	} These quantities are read from tape, converted to binary, scaled to q=0, and stored at these addresses by Data Input #1 Subroutine
		2,7			'	☒ a_0	
		2,8			'	a_1	
		2,9			'	a_2	
		3,0			'	a_3	
		3,1			'	☒ a_4	

' Conditional Stop Code ☒ Carriage Return

SCALING

The LGP-30 normally handles all numbers as if they were of the form .XXXX....., that is, numbers numerically less than 1. However, it is quite simple to carry any number in the machine at any number of binary places, and this arithmetic is explained below. In talking about the placement of the radix point in the LGP-30, it is simpler to talk of the number of whole places in front of the radix point, rather than the number of places after the point. Hereafter, a number will be referred to as being carried at q places, q being the number of binary digits to the left of the radix point, and 30-q as the number to the right of the point.

Addition: Addition of course poses no problem if the two numbers to be added are at the same number of places. If not, either may be shifted before addition by multiplying or dividing by "One" at an appropriate q.

Multiplication: The LGP-30 multiplies a number at q_1 places by a number at q_2 places and forms the product in the accumulator at q_1 plus q_2 places.

Division: The LGP-30 divides the accumulator at q_1 places by a number at q_2 places and forms the quotient in the accumulator at $q_1 - q_2 = q_3$ places. It should be noted that overflow will occur if the quotient developed is not less than 2^{q_3} in absolute value.

ROYAL McBEE CORPORATION

LGP-30 Input Output

Keyboard Code

Numerical

	123456
)0	000010
Ll	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

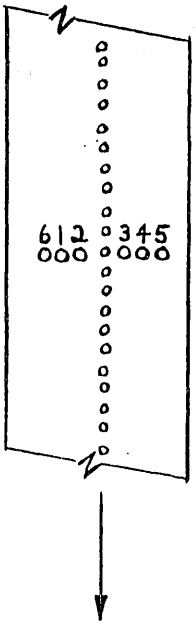
	123456
Lower Case	000100
Upper Case	001000
Color Shift	001100
Car Ret	010000
Back Space	010100
Tab	011000
Cond Stop (')	100000
Start Read	000000
Space	000011
Delete	111111

Signs

= +	001011
_ -	000111

Balance of Keyboard

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



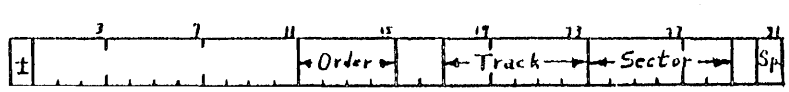
Symbol	Command	Binary	Hex	Dec
Z	Stop	0000	0	0
B	Bring	0001	1	1
Y	Store Add.	0010	2	2
R	Return Add.	0011	3	3
I	Input	0100	4	4
D	Divide	0101	5	5
N	N Multiply	0110	6	6
M	Multiply	0111	7	7
P	Print	1000	8	8
E	Extract	1001	9	9
U	Transfer	1010	F	10
T	Test	1011	G	11
H	Hold	1100	J	12
C	Clear	1101	K	13
A	Add	1110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

Keyboard Code

)0	02	02	Zz	01	01
Ll	06	06	Bb	05	05
#2	0f	10	Yy	09	09
"3	0q	14	Rr	0k	13
Δ4	12	18	Ii	11	17
%5	16	22	Dd	15	21
$6	1f	26	Nn	19	25
π7	1q	30	Mm	1k	29
Σ8	22	34	Pp	21	33
(9	26	38	Ee	25	37
Ff	2f	42	Uu	29	41
Gg	2q	46	Tt	2k	45
Jj	32	50	Hh	31	49
Kk	36	54	Cc	35	53
Qq	3f	58	Aa	39	57
Ww	3q	62	Ss	3k	61
Space	03	03	LC	04	04
_-	07	07	UC	08	08
=+	0g	11	CS	0j	12
:;	0w	15	CR	10	16
?/	13	19	BS	14	20
].	17	23	Tab	18	24
[,	1g	27	Del.	3w	63
Vv	1w	31	'	20	32
Oo	23	35			
Xx	27	39			

2^N	N	2^{-N}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5

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



BOOTSTRAP ROUTINE
(PROGRAM 09.0)

FUNCTION:

To load the input routine on tracks 00, 01 and 02. After the bootstrap program has loaded the entire input routine, a halt is executed at track 63 sector 13 (3W34.) Depressing the Start button transfers control to the first instruction of the input routine.

PROCEDURE:

The tape containing the bootstrap (and the program input routine) is placed in the tape reader and then the following manual operations are performed

1. Connect Switch to "off" position.
   ("Inp. - Comp." switch to "Comp." on early machines.)

   a. Depress Flexowriter "Start Read" button.

2. Depress "Manual Input" button on console.
   ("Interrogate" button on early machines.)

3. Depress Flexowriter "Start Read" button.

4. Depress "Fill Instruction" button.
   ("Fill R" on early machines.)

5. Depress Flexowriter "Start Read" button.

6. Depress "One Operation" button.

7. Depress "Execute Instruction" button.
   ("Execute R" button on early machines.)

8. Repeat steps 2 through 7 five more times before proceeding to step 9.

9. Depress "One Operation" button.

10. Depress "Normal button.

11. "Connect" switch to "On".
    (Inp - Comp." switch to "Inp." on early machines.

12. Depress Computer "Start".

The entire tape will automatiaclly read in after manually performing step 12 above.

(PROGRAM 09.0)

OUTPUT:

Program input routine on tracks 00, 01, 02.

STORAGE:

The bootstrap routine uses 21 words on track 63. (Sector 00 through 14, 22 thru 26 and 46).

TIME:

The time to read in the two programs after depressing the Start button in Step 12 is approximately three minutes.

LGP-30 PROGRAM INPUT ROUTINE
(PROGRAM 10.0)

The purpose of this report is to describe a method of entering information into the LGP-30. The general characteristics of the LGP-30 are described in the programming manual. A program consists of two types of words, data and instruction. This write-up primarily describes the process of inputing instruction words and hexadecimal representations of data words. This process does not handle data words expressed in decimal form.

There are several functions to be performed by a useful input routine.

1. The most direct way of entering information into the LGP-30 is to present it with binary words. But since it is difficult to program in this number system, we prefer to do our programming in decimal notation. If we are to write words in decimal form, we must provide the machine with a means of converting such words into binary form.

2. Most routines contain instructions which refer to other locations within that routine. Hence if we wish to place the routine in another portion of memory, we must modify some of these addresses.

3. It is sometimes useful to express a number in binary form. e.g., π or other universal constants.

4. It may be necessary to make instructional or data changes to a program that has already been stored in memory.

These are the functions which this input routine is designed to perform.

This routine recognizes seven types of input word. The sign and first 3 bits of the input word are used for the input routine to identify the type. These words and their symbols are as follows:

1. Instruction (none) consists of an order and decimal address. The address consists of a decimal track and sector. The instruction is converted to its binary equivalent and stored in a given location. The address portion is incremented by the contents of the "modifier" (to be discussed below) unless an "x" precedes the order. e.g. b 4000 will in incremented. x b 6310 will not be incremented (and the x will not appear in the stored instruction).

2. Command (+) This word will be treated as an order to the input routine. The order will be executed after entry of another word. The command is input in decimal and is not incremented by the modifier. The second word, presumably data, is input in hexadecimal. e.g. +00h1637 followed by 73W08 will store the hexadecimal word 73W08 in memory location 1637.

3. Start fill (;). Tells the input routine where to begin filling input words. Each succeeding word will be filled consecutively. The address portion of the start fill word is decimal, and consists of both track and sector number. e.g. ;0003128. The first stored word will be located in track 31, sector 28.

4. Set modifier (/). The magnitude of the address of the modifier will be used to increment orders. The set modifier word will usually follow a start fill and will usually be identical to it in magnitude. This word is for use by the input routine only.

5. Stop and transfer (.). This word stops the flexowriter. A "start" will transfer control to the memory location contained in the address portion of the stop and transfer word. Depressing break switch 32 on the control panel will cause the computer to disregard the stop portion of this word. e.g. .0001700 will stop reading, then control will be transferred to track 17, sector 00.

6. Hex. words (,). This instruction causes the next N words to be filled without conversion. N is specified in the address portion of the "hex. words" word and must be within the range $1 \leq N \leq 63$ e.g. ,0000014 means the next 14 words are to be stored in the next 14 consecutive memory locations. The words must be in hexadecimal notation, and they will not be incremented by the modifier.

7. Hex. fill (v). Fills the next n words hexadecimally beginning in m. m and n are proper hexadecimal numbers. The format of the word is v n_1 n_2 n_3 m_1 m_2 m_3 m_4. e.g. v1J02W00 means the next 1J0 words [$(1J0)_{16} = (448)_{10}$] will be filled consecutively beginning in location 2W00 [$(2W00)_{16}$ = track $(47)_{10}$ sector 00]. Up to $(7WW)_{16} = (2047)_{10}$ words can be filled by a single hex fill input order.

Leading zeros need not be punched on any input word. All other zeros must be punched. e.g. 800T0018 must be completely punched; 000B3749 only the last five characters need be punched. e.g. B3749.

When the overall coding for a problem is surveyed, it is found that the instructions separate logically into independent groups, some of which can be used in any number of problems. Examples of these groups are subroutines of all types, standard input and output routines, and the mathematical subdivisions of the problem. It would be desirable to code these pieces without reference to the other pieces. In order to separate these pieces completely, it is necessary to assign a group of instructions a block of storage locations which does not correspond to actual memory locations; otherwise two blocks of coding might be found to occupy the same section of storage, requiring a change in the

coding for one of the two pieces. The "set Modifier" order of the input routine was intended to facilitate this type of coding. A group of instructions can be coded without reference to actual memory locations by starting that group at symbolic address 0000. Then by setting the modifier to the "start fill" location, the programmer may position a routine to any part of memory. An instruction preceeded by an "x" will not be incremented, and this instruction will still refer to an absolute memory location. It should be noted that orders may be coded for actual locations merely by setting the modifier to zero (i.e., input order /0000000). Thus no particular restrictions are imposed upon the programmer by this system.

If the input routine detects an erroneous input code it will print "code" and halt. The last word read from tape contains the erroneous code in the first punched character.

A tape prepared for this input routine must contain flexowriter format control. It is suggested that a carriage return be inserted after every four words on tape. If there is no format control and the flexowriter carriage is permitted to space into the automatic carriage return a stop will result. The computer will continue if the carriage return button is depressed.

INPUT ROUTINE FORMAT
PROGRAM 10.0

Code	Operation	Track	Sector		INTERPRETED AS:
	Op	1 2 3	4	Mod.	
	x Op	1 2 3	4	Not Mod.	Instruction -
8 0 0	T	1 2 3	4	Mod.	Order plus address (dec.)
8 0 x	T	1 2 3	4	Not Mod.	
+ 0 0 0	Op	1 2 3	4		Command. This word is treated as an order, using the following word as data. Following word in hex.
; 0 0 0		1 2 3	4		Start Fill - Address in decimal
/ 0 0 0		1 2 3	4		Set Modifier - Address in decimal
. 0 0 0		1 2 3	4		Cond. Stop and Transfer - Stop (unless Sw 32 down) and transfer to location specified in address.
, 0 0 0 0 0		1 2			Hexadecimal words. Next N (dec) words are hex. fill sequentially $1 \leq N \leq 63$
v $n_1n_2n_3$	$m_1m_2m_3m_4$				Hexadecimal Fill - Fill N (hex) words beginning in loc. M (hex) $1 \leq N \leq 2047$

Rev. 3/25/57

HEXADECIMAL INPUT ROUTINE
(Program 10.1)

FUNCTION:

1. To read hexadecimal information and store it on the memory drum.

2. To verify that the information has been correctly stored by generating a summation of the binary bits stored on the drum (a check sum) and checking this summation against a previously computed summation placed on the input tape.

INPUT:

A tape prepared in its entirety by program 13.1. This tape contains the following:

1. An identification word in the form v n_1 n_2 n_3 m_1 m_2 m_3 m_4 where N = (n_1 n_2 n_3) = the number of words in hexadecimal to be placed on the drum. M = (m_1 m_2 m_3 m_4) = the initial location in hexadecimal to begin storing the words.

2. N hexadecimal words, each followed by a conditional stop code.

3. The check sum.

PROCEDURE:

1. Transfer to the beginning of this routine with the previously prepared tape in the reader.

2. The routine will read the identification word and set up the initial address and a tally from M and N respectively.

3. The hexadecimal words are read and stored sequentially on the memory drum. After each word is stored the address (for the next word) is incremented by 1 and the tally is decremented by 1.

4. After all hexadecimal words on tape have been placed in memory, the check sum is read in and stored within this routine. Then another check sum is computed in the identical manner used by program 13.1.

5. The computed check sum is subtracted from the one placed on tape. If they are equal, the routine returns for another identification word.

6. If the two check sums are not equal the routine prints "error" and halts. To re-read the same record, back up the tape in the reader to the last "v code" (identified by a punch in channel 6) and depress

(Hexadecimal Input Routine continued)

the start button. The routine will return to re-read the identification word and proceed from there.

OUTPUT:

The information on the tape stored in memory and checked for validity.

TIME:

Reading from tape - one track per minute.
Computing check sum - ten tracks per minute.

STORAGE:

96 locations of instructions and constants.
No temporary storage.

DATA INPUT #1 SUBROUTINE
(PROGRAM 11.0_R)

FUNCTION:

To read a decimal number from tape, convert to binary, scale to the proper binal point location, and store the word in a specified drum location. For each number the following is punched on tape:

1. The decimal point location of the number on tape, counting from right to left. (One decimal digit designated as "P").
2. The binal point location desired for the number to be placed on drum. (Sign and two decimal digits designated as "q").
3. The drum location to which the number is to be sent. (2 decimal digits for track and 2 decimal digits for sector).
4. The number to be entered (Seven decimal digits plus sign).

INPUT:

For each word to be stored, an identification word (parts 1, 2, & 3 above), and the signed number (part 4 above) are required.

CALLING SEQUENCE:

Loc.	Inst.	Add.
α	R	$(L_o + 8)_{10}$
$\alpha + 1$	U	L_o
$\alpha + 2$	etc.	

OUTPUT:

The scaled binary representation of the number will appear at its proper drum location.

EXIT:

A "zero" identification word will cause the routine to exit to $(\alpha + 2)$.

PROGRAM 11.0R

SCALING:

The location of the decimal point in the decimal number is specified by a number P, which denotes the number of places following the point in the seven digit field. P can be in the range $0 \leq P \leq 9$. The location of the binary point in the full 30 bit binary word is specified by q, the number of digits preceeding the point in the full word. q can lie in the range given in the following table.

In order for the number to be representable at a given q, the number must be less (in absolute value) than $_2q$. However, if too large a q is used, the number will not reconvert exactly on output, since there will be too few binary digits following the point to adequately represent the fractional part of the number. The following table also gives the maximum conversion correspondence between P and q.

TABLE OF P vs q:

P	Max q	Min q	Max q for Exact Reconversion	Min q for Max No. Size (all 9's)
0	+47	+2	+30	+24
1	+43	-2	+26	+20
2	+40	-5	+23	+17
3	+37	-8	+20	+14
4	+34	-11	+16	+10
5	+31	-14	+13	+7
6	+28	-17	+10	+4
7	+24	-21	+6	0
8	+21	-24	+3	-3
9	+17	-28	0	-6

PROGRAM 11.0R

TIME:

20 - 25 words per minute.

ACCURACY:

The scaled number in memory may be inaccurate in the 30th binary position.

STORAGE:

192 locations of instructions and constants. Eight locations of temporary storage (Track 63, sectors 03, 04, 45, 52, 54, 55, 56, 57).

PROGRAM STOPS:

Loc.	Meaning
$(L_0 + 0234)_{10}$	Divide check in scaling data word. $\|N\| \geq 2^q$.

EXAMPLES: (See LGP-30 Data Input 1 load sheet)

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

TAPE PUNCHING INSTRUCTIONS:

1. All characters of the I.D. word must be punched. e.g., punch 0+096300' completely. The first three characters should not be omitted. The stop code (') must be the last character.
2. Leading zeros of a positive number need not be punched. To enter all zeros merely punch a stop code. All digits of a negative number must be punched.
3. Be sure to check each load sheet to see whether an additional stop code should follow the last number punched.

LGP-30 DATA LOAD SHEET

Job No._____ Prog. No. 11.0R Prep. by_____ Ck'd. by_____ Date_____

Problem EXAMPLES FOR DATA INPUT #1 Case_____

INPUT #1

Quan.	P	± q	Location	Stop	± Number	Stop	Car. Ret.
	5	+10.7	6234	P	9640.236	P	
	9	−1.4	2363	P	−0.000.597	P	X
	0	+30.0	2100	P	33.000.000	P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X
				P		P	
				P		P	X

$0 \leq P \leq 9$

$-28 \leq q \leq 47$

$0000 \leq Loc \leq 6363$

Punch a stop code after the last number? Yes X No_____

This Subroutine was developed by the Librascope Computing Group

DATA INPUT NO. 2 SUBROUTINE
(Program 11.1)

FUNCTIONS:

1. To input a sequence of "N" signed seven decimal digit numbers, each with the same decimal point, convert to binary (all at the same q) and store sequentially in M, M + 1.....

OR

2. To input one signed seven decimal digit integer and convert to a signed binary integer at q = 30.

I. SEQUENTIAL FILL:

Input:
"N" signed decimal numbers on tape and the hexadecimal code word in the accumulator.

Calling Sequence:

Loc.	Inst.	Add.	
α - 1	B	L (Code word)	
α	R	$(L_o + 0121)_{10}$	Track 01, Sector 21.
α + 1	U	$(L_o + 0104)_{10}$	Track 01, Sector 04.
α + 2	etc.		

α - 1 need not contain a B order. Any order or sequence of orders that leaves the code word in the accumulator is permissible.

The code word must be in the form: $n_1\ n_2\ n_3\ C\ m_1\ m_2\ m_3\ m_4$.
N = $(n_1\ n_2\ n_3)$ = number of words to be filled (in hexadecimal at q = 11) $0<N<2048$.
C = characteristic of numbers to be filled (number of integers) $0<C<9$. See correspondence of C and q under "Output.
M = $(m_1\ m_2\ m_3\ m_4)$ = First address to be filled (in hexadecimal at q = 29).

Examples:

1. Code word 00F30200 means fill 10 (F) words, starting in 0200 with decimal numbers of the form $\pm$ XXX.XXXX (stored at q = 10)

2. Code word 0318051J.
   $N = (31)_{16} = (49)_{10}$
   $C = 8$
   $M = (051J)_{16} = (0507)_{10}$

(Data Input No. 2 Subroutine continued)

Fill 0507, 0508,....., 0555 with the next 49 decimal words on tape. Words are interpreted as ±XXXXXXX0 (stored at q = 27).

Output:

Binary representation of words on tape filled sequentially beginning in location M.

C	q	Decimal words interpreted as:
0	0	± .XXXXXXX
1	4	± X.XXXXXX
2	7	± XX.XXXXX
3	10	± XXX.XXXX
4	14	± XXXX.XXX
5	17	± XXXXX.XX
6	20	± XXXXX.X.X
7	24	± XXXXXXX.
8	27	± XXXXXXX0.
9	30	± XXXXXX00.

Exit:

After all words have been scaled and stored, the routine exits to $\alpha + 2$.

Accuracy:

± 1 at q = 30 for $0 \leq C \leq 6$
Exact conversion for $7 \leq C \leq 9$

Time:

45 to 55 words per minute.

Note:

If the hexadecimal code word is on tape, replace the α-1 instruction of calling sequence by the instructions P0000 and I0000.

II. ONE WORD INTEGER CONVERSION:

Input:

One signed decimal integer on tape in the form ± XXXXXXX.

Calling Sequence:

Loc.	Inst.	Add.
α	R	$(L_o + 7)_{10}$
$\alpha + 1$	U	L_o
$\alpha + 2$	etc.	

(Data Input No. 2 Subroutine continued)

Output:

Binary integer at q = 30 in accumulator.

Accuracy:

Conversion is exact.

PROGRAM STOP:

Loc.	Meaning
$(L_o + 0117)_{10}$	Divide check in scaling data word. $\|N\| \geq 2^q$

STORAGE:

89 locations of instructions and constants.
No temporary storage.

NOTE:

Leading zeros of a positive number need not be punched. To enter all zeros merely punch a stop code. All digits of a negative number must be punched.

DATA INPUT NO. 3 SUBROUTINE
(Program 11.2)

FUNCTION:

To input groups of decimal numbers from tape, each group with the same decimal point, convert each number to binary, all at the same q, and store in consecutive memory locations. This differs from Input No. 1 in that each group of numbers, rather than every number, is preceded by an identification word. The identification word contains P, ± q, and the location for the first number to be stored. All following numbers of the group are filled sequentially at the same P and q. A "minus zero" number will terminate the group, and another identification word will be read (See examples for "minus zero" format).

CALLING SEQUENCE: (Same as Input No. 1)

Loc.	Inst.	Add.
α	R	$(Lo + 8)_{10}$
$\alpha + 1$	U	Lo
$\alpha + 2$	etc.	

EXIT:

A zero identification word (normally preceded by a minus zero number) will cause the routine to exit to $\alpha + 2$.

TIME:

45 - 55 words per minute .

STORAGE:

192 locations of instructions and constants.
(3 tracks).
Five locations of temporary storage.
(Track 63, sector 00, 01, 02, 03, 04)

PROGRAM STOP:

Loc.	Meaning
$(Lo + 0135)_{10}$	Divide check in scaling data word $\|N\| \geq 29$.

4/3/57

(Data Input No. 3 continued)

EXAMPLES: (See LGP-30 Data Input 3 Load Sheet)

Group No. 1. Place +96.40236 in drum location 6234 at q = 7
" -30.00000 " " " 6235 " "
" +03.14159 " " " 6236 " "
" -21.50000 " " " 6237 " "
The minus zero causes identification word No. 2 to be read.

Group No. 2. Place -.000000597 in drum location 2363 at q = -14
" +.000009000 " " " 2400 " " "
" +.000060000 " " " 2401 " " "
The minus zero causes identification word No. 3 to be read.

Group No. 3. Place +330000 in drum location 2100 at q = 30
The minus zero word causes identification word No. 4 to be read. Identification word No. 4 is the extra stop code punched after the last number (see bottom of load sheet, "Yes" is checked).

This enters a zero identification word, so the subroutine exits to (α + 2).

See Data Input I write-up for "Scaling", "Table of P vs q", and "Tape Punching Instructions".

LGP-30 DATA LOAD SHEET

Job No._____ Prog. No. 11.2 Prep. by_____ Ck'd. by_____ Date_____

Problem EXAMPLES FOR DATA INPUT #3 Case_____

INPUT #1
OR
INPUT #3

Quan.	P	± q	Location	Stop	± Number	Stop	Car. Ret.
	5	+07	6234		9640236		
					−30000000		X
					31415 9		
					−21500000		X
					−00000000		
	9	−14	2363		−00005977		X
					9000		
					60000		X
					−00000000		
	0	+30	2100		3300000		X
					−00000000		
							X
							X
							X
							X
							X
							X
							X
							X
							X
							X
							X

$0 \leq P \leq 9$

$-28 \leq q \leq 47$

$0000 \leq Loc \leq 6363$

Punch a stop code after the last number? Yes X No_____

DATA OUTPUT #1 SUBROUTINE
(PROGRAM 12.0)

FUNCTION:

To convert and print a nine decimal digit number plus decimal point and sign (sign if the number is negative).

INPUT:

The number to be printed in the accumulator and a code number in storage location $\alpha + 2$ to indicate the number of integers before the decimal point.

CALLING SEQUENCE:

loc.	Inst.	Add.
$\alpha - 1$	B	L(N)
	R	$(L_o + 12)_{10}$
$\alpha + 1$	U	L_o
$\alpha + 2$	Z	000C
$\alpha + 3$	etc.	

$\alpha - 1$ need not contain a bring order. Any order which leaves the argument in the accumulator is permissable.

C denotes code number and may be 0 thru 9.

OUTPUT:

Nine decimal digits plus a sign (or space if the number is positive) and a decimal point.

CODE NUMBER:

C	q of No.	Output
0	0	.XXXXXXXXX
1	4	X.XXXXXXXX
2	7	XX.XXXXXXX
3	10	XXX.XXXXXX
4	14	XXXX.XXXXX
5	17	XXXXX.XXXX
6	20	XXXXXX.XXX
7	24	XXXXXXX.XX
8	27	XXXXXXXX.X
9	30	XXXXXXXXX.

MISCELLANEOUS:

Each binary number is scaled and converted as a fraction. The code number is used by the routine as a print stroke counter. The only format control is a tab after printing. It is safe to print immediately on exit from the routine. After printing control is returned to $\alpha+3$.

TIME:

Printing takes about 1.5 seconds including the tab.

PROGRAM 12.0

DATA OUTPUT #1 SUBROUTINE (cont'd)

ACCURACY:

Maximum error is one in the ninth printed digit.

STORAGE:

96 locations of instructions and constants.
No temporary storage.

PROGRAM STOP:

$(L_0+38)_{10}$ Argument // 10^c

DATA OUTPUT NO. 2 SUBROUTINE
(Program 12.1)

FUNCTION:

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

INPUT:

One or more groups of numbers to be printed, the initial location, the number of numbers in each group, and the binal point location (q) of each group.

CALLING SEQUENCE:

Loc.	Inst.	Add.
$\alpha - 1$	B	L(1st. No.)
α	R	Lo + 5
$\alpha + 1$	U	Lo
$\alpha + 2$	Z	$N_1 q_1$
$\alpha + 3$	Z	$N_2 q_2$
.	.	.
.	.	.
.	.	.
.	.	.
$[(\alpha+1) + i]$	Z	$N_i q_i$
$[(\alpha+1)+(i+1)]$	etc.	

N_i = number of words in group i. N_i is placed in the track position (in decimal). $1 \leq N_i \leq 63$.

q_i = binal point location of the i'th group. q_i is placed in the sector position (in decimal) $0 \leq q_i \leq 31$.

OUTPUT:

Printed decimal representations of the numbers specified. Each output number will consist of a sign (space or minus), eight (or more) decimal digits and the decimal point. A tab is given after each number. There will be q/4 (rounded) = J printed integers unless the number to be printed will not fit at that J. If the number is too large, J is increased by enough to accommodate the number. The number of decimal places will be 8 - J (≥ 0). If more than eight integers are needed to express the number, J will be increased and no decimal places will be printed.

EXIT:

The routine exits to the first "non - Z" instruction.

EXAMPLE:

Loc.	Inst.	Add.	
$\alpha - 1$	xB	2100	
α	xR	$\beta + 5$	
$\alpha + 1$	xU	β	
$\alpha + 2$	xZ	0407	(1)
$\alpha + 3$	xZ	1015	(2)
$\alpha + 4$	xB	J	(3)

The above calling sequence will cause this subroutine to:

1. Print the contents of 2100, 01, 02, and 03 as ±XX.XXXXXX or ±XXX.XXXXX. J = 7/4 (rounded) = 2. If one or more of these numbers is too large, J will be increased to 3 for that number.

2. Print the contents of 2104, 05.....13 as ±XXXX.XXXX unless one or more of these numbers is too large. The number(s) which overflow J = 4 will be printed as ±XXXXX.XXX.

3. Exit to $\alpha + 4$, which is the first "non-Z" order following the calling sequence.

PROGRAM STOPS:

None.

ACCURACY:

Output is exact (and rounded) for eight printed digits. When nine digits are printed, the output will be in error by 5 or 6 in the ninth place.

STORAGE:

160 locations of instructions and constants (2 1/2 tracks).
No temporary storage.

TIME:

Approximately 30 words per minute.

RANGE:

The number to be printed must be within the range $-10^9 < N < 10^9$. q must be within the range $0 \leq q \leq 31$.

HEXADECIMAL PUNCH OR PRINT
(PROGRAM 13.0)

FUNCTION:

1. To print the contents of consecutive memory locations, or
2. To punch (and print) the contents of consecutive memory locations.

INPUT:

Beginning and final locations in decimal.

OUTPUT:

Hexadecimal representation of the contents of memory locations L_o through L_f. Output is six words per line. Leading zeros are not punched or printed. When a memory location's value is zero, the flexowriter executes a space if the "transfer control" button is down. When the Transfer Control button is up, the routine punches a conditional stop code for a memory location that contains zero. When punching, an identification word (for use by the program input routine) is punched as the first word on tape.

FORMAT CONTROL:

Depressing the "transfer control" button on the computer console causes the routine to space between printed words. Otherwise stop codes are punched after each word. If the transfer control button is down the carriage is returned before printing. If it is up the control word is punched. The control word consists of v N M, where N is the number of words in the record to be punched, and M is the initial location. Both M and N are in Hexadecimal. $001_{16} \leq N \leq 7ww_{16}$. e.g., vl4j2k00! denotes a record of 332 words beginning in location 4500. This word is recognized and used by the program input routine (Program 10.0) as a hexadecimal fill instruction.

PROCEDURE:

1. Depress "manual input" button on the flexowriter.
2. Transfer to the first location of this routine.
3. After the "manual input" light turns on, type the initial and final locations (in decimal) into the keyboard.
4. Make sure the transfer control button is in the correct position - up for punchout - down for printout.
5. Depress "punch on" switch on flexowriter for punchout.
6. Depress the "start comp." button on the flexowriter.

TIME:

Approximately 45 words per minute.

STORAGE:

158 locations of instructions and constants.
No temporary storage.

HEXADECIMAL PUNCH OR PRINT NO. 2
(Program 13.1)

FUNCTION:

This subroutine functions and operates the same as program 13.0 with the exception that after punching has been completed ("transfer control" switch up), this routine computes a check sum and punches it as the last word on tape. This check sum is intended for use by programs 10.1 and 10.2 which determine whether the content of the tape has been correctly recorded on the memory drum.

TIME:

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

STORAGE:

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

No temporary storage.

SIN-COS SUBROUTINE
(PROGRAM 14.0)

FUNCTION:

To compute the sine or cosine of any given angle. A 9th degree polynomial approximation is used. The argument must be in degrees and will be reduced to the first quadrant equivalent before computation of the function.

INPUT:

One word in the accumulator at q = 9.

OUTPUT:

One word in the accumulator at q = 1.

CALLING SEQUENCE:

SINE			COSINE		
Loc.	Inst.	Add	Loc.	Inst.	Add.
$\alpha - 1$	B	L (Arg.)	$\alpha - 1$	B	L (Arg.)
α	R	$(Lo + 49)_{10}$	α	R	$(Lo + 49)_{10}$
$\alpha + 1$	U	L_o	$\alpha + 1$	U	$(L_o + 4)_{10}$
$\alpha + 2$	etc.		$\alpha + 2$	etc.	

$\alpha - 1$ need not be a B order. Any order or orders that leaves the argument in the accumulator is permissable.

ACCURACY:

The maximum error is approximately 5×10^{-7}.

TIME:

250 to 275 MS.

STORAGE:

64 locations of instructions and constants. 6 locations of temporary storage (Track 63, sectors 02, 04, 05, 06, 07, 45).

SQUARE ROOT SUBROUTINE
(PROGRAM 15.0)

FUNCTION:

To compute the square root of any positive number. The argument may be at any even q, and the output will be at q/2.

INPUT:

One word in the accumulator at any even q.

OUTPUT:

One word in the accumulator at q/2.

CALLING SEQUENCE:

Loc.	Inst.	Add.
$\alpha - 1$	B	L (Arg.)
α	R	$(Lo + 50)_{10}$
$\alpha + 1$	U	L_o
$\alpha + 2$	etc.	

$\alpha - 1$ need not contain a B order. Any order or orders that leaves the argument in the accumulator is permissable.

ACCURACY:

Answer is correct to 30 bits.

TIME:

Varies from 500-750 MS.

STORAGE:

64 locations of instructions and constants. 5 locations of temporary storage (track 63, sectors 19, 20, 21, 23, 24).

PROGRAM STOPS:

Loc.	Meaning
$(Lo + 61)_{10}$	Argument is negative. A start exits with zero in accumulator.

NOTE:

A single bit in the 30th position will be treated as zero.

Rev. 3/26/57

ARCTANGENT SUBROUTINE
(PROGRAM 16.0)

FUNCTION:

To compute the arctangent of any given number. A 15th degree polynomial approximation is used. The output is in degrees, and the principle value will be given (first or fourth quadrant).

INPUT:

One word in the accumulator at q = 9.

OUTPUT:

One word in the accumulator at q = 9 (degrees).

CALLING SEQUENCE:

Loc.	Inst.	Add.
$\alpha - 1$	B	L (Arg.)
α	R	$(Lo + 51)_{10}$
$\alpha + 1$	U	L_o
$\alpha + 2$	etc.	

$\alpha - 1$ need not be a B order. Any order or orders that leaves the argument in the accumulator is permissable.

ACCURACY:

Maximum error is 5×10^{-7}. The output will be between 0° and 89.90°, because the argument cannot be numerically greater that 512. If the programmer wants his output to come closer to 90° he can modify the routine by changing $(L_o + 56)_{10}$ and $(L_o + 59)_{10}$ from 1 and 2, respectively, at q = 9, to 1 and 2 at some greater q_2. Then the argument must be at q_2.

TIME:

320 milliseconds.

STORAGE:

64 locations of instructions and constants. 10 locations of temporary storage (track 63, sectors 04, 05, 06, 07, 08, 09, 10, 13, 50, 51).

EXPONENTIAL SUBROUTINE
(PROGRAM 17.0)

FUNCTION:

To evaluate the function K^x, where K = 2, e, or 10, and $-1 \leq x \leq 1$. To obtain higher values of the exponential function, multiply the out - put of the subroutine K to the integer part of the exponent.

EXAMPLES:

$$10^{2.5} = 10^{2.} \cdot 10.5$$
$$2^{-3.5} = 2^{-3.} \cdot 2^{-.5}$$
$$e^{x.xx} = e^{x.} \cdot e^{.xx}$$

INPUT:

One word in the accumulator at q = 1.

OUTPUT:

One word in the accumulator at q = 4.

CALLING SEQUENCE:

Loc.		
$\alpha - 1$	B	L (arg.)
α	R	$(L_o + 09)_{10}$
$\alpha + 1$	U	L_o for 2^x
$\alpha + 1$	U	$(L_c + 2)_{10}$ for e^x
$\alpha + 1$	U	$(L_o + 3)_{10}$ for 10^x
$\alpha + 2$	etc.	

$\alpha - 1$ need not contain a B order. Any order or orders that leaves the argument in the accumulator is permissible.

ACCURACY:

Answer is correct to 5×10^{-8}.

TIME:

255 to 285 MS.

STORAGE:

63 locations of instructions and constants.
No temporary storage.

LOG_k X SUBROUTINE
(PROGRAM 18.0)

FUNCTION:

To compute the logarithm of any given number to the base 2, e, or 10. A 7th degree polynomial approximation is used. The argument must be positive. K, the base to be used, must be specified in the calling sequence.

INPUT:

One word in the accumulator at a positive q.

OUTPUT:

One word in the accumulator at q=6.

CALLING SEQUENCE:

α - 1	B	L (arg.)		
α	R	$(Lo + 24)_{10}$		
α + 1	U	Lo	Lo =	Initial location of Subroutine.
α + 2	Z	q	q =	No. of places in argument.
α + 3	Z	K	K =	(0 for $\log_2 X$)
α + 4	etc.			(1 for $\log_e X$)
				(2 for $\log_{10} X$)

α - 1 need not be a B order. Any order or orders that leaves the argument in the accumulator is permissable.

NOTES:

The argument must be greater than zero. q, the number of places in the argument must be in the range $0 \leq q \leq 31$. If K, the type of output, is not equal to 0 or 1, the base 10 will be used.

ACCURACY:

The error is 3×10^{-8}.

PROGRAM STOPS:

$(Lo + 8)_{10}$ Argument is zero or negative.

TIME:

Approximately (445 + 30 N) MS, where N is the number of leading zeros.

STORAGE:

122 locations of instructions and constansts.
No temporary storage.

10/29/56

ALPHANUMERIC OUTPUT SUBROUTINE
(Program 19.0)

FUNCTION:

To print (or punch and print) alphabetic and/or numeric information.

INPUT:

A set of code words, where each code word consists of 4 alphanumeric output codes.

CALLING SEQUENCE:

Loc.	Inst.	Add.
α	R	Lo
$\alpha + 1$	U	Lo
$\alpha + 2$	[c o d e w o r d]	
.		
.		
.		
.		
$(\alpha + 1) + n$	[code word containing VQ]	
$(\alpha + 1) + (n + 1)$	etc.	

Where n is the number of code words.

EXAMPLE:

PROGRAM INPUT CODES		LOCATION	INSTRUCTION OP. ADDRESS		CONTENTS OF ADDRESS	NOTES
	'					
	' ☒					
		0 0	R B	'		Enter Alphanumeric
		0 1	U B	'		Subroutine
2 0 0 0 0 0 3	'	0 2	2 0 1 0 0 J 5 J	'	Carr.Ret, UC, L, G	Carr. Ret. print
		0 3	4 2 0 8 0 A 1 J	'	☒P, LC, -, 3	"LGP-30", tab, &
		0 4	0 4 3 0 V Q 0 0	'	0, tab, exit	exit
		0 5	etc.	'		Continue with prog.

This calling sequence will perform a carriage return, print "LGP-30" and execute a tab.

3/29/57

(Alphanumeric Output Subroutine continued)

OUTPUT:

Printing (or punching and printing) of alphanumeric characters selected.

ALPHANUMERIC OUTPUT CODES:

(See page 3 of this write-up)

EXIT:

The routine will exit to the location following the location containing the exit code (VQ).

STORAGE:

58 locations of instructions and constants.
No temporary storage.

TIME:

About 400 characters per minute.

NOTE:

An increase in output speed can be obtained by switching the instruction in location 0035 with the one in 0036. This will raise output speed to 475 characters per minute. But this change requires that there not be a long carriage return or tab code as the 4th code of a code word.

(Alphanumeric Output Subroutine continued)

6-BIT ALPHANUMERIC OUTPUT CODES

) 0	04	Aa	72
L 1	0J	Bb	0F
* 2	14	Cc	6F
" 3	1J	Dd	2F
Δ 4	24	Ee	4F
% 5	2J	Ff	54
$ 6	34	Gg	5J
π 7	3J	Hh	62
Σ 8	44	Ii	22
(9	4J	Jj	64
Space	06	Kk	6J
_ -	0A	Ll	0J
= +	16	Mm	3F
: ;	1A	Nn	32
? /	26	Oo	46
] .	2A	Pp	42
[,	36	Qq	74
TAB	30	Rr	1F
Lower Case	08	Ss	7F
Upper Case	10	Tt	5F
Color Shift	18	Uu	52
Carr. Ret.	20	Vv	3A
Back Space	28	Ww	7J
'	40	Xx	4A
Leave Routine	VQ	Yy	12
		Zz	02

4/3/57

ARCSINE - ARCCOSINE SUBROUTINE
(Program 20.0)

FUNCTION:

To compute the arcsine or arccosine of any given value between $-1 \leq X \leq 1$. A 6th degree polynomial approximation is used.

INPUT

One word in accumulator at q = 1.

OUTPUT:

One word in the accumulator at q = 9 in degrees.

CALLING SEQUENCE:

Arcsine			Arccosine		
Loc.	Inst.	Add.	Loc.	Inst.	Add.
$\alpha - 1$	B	L(arg.)	$\alpha - 1$	B	L(arg.)
α	R	$(L_o + 21)_{10}$	α	R	$(L_o + 21)_{10}$
$\alpha + 1$	U	L_o	$\alpha + 1$	U	$(L_o + 0211)_{10}$*
$\alpha + 2$	etc.		$\alpha + 2$	etc.	

*i.e. track 02 sector 11

α - 1 need not be a B order. Any order or orders that leaves the argument in the accumulator is permissable.

ACCURACY:

The maximum error is approximately 5×10^{-7}

TIME:

350 to 375 ms.

STORAGE:

160 locations of instructions and constants. 11 locations of temporary storage (track 63, sectors 12, 15, 16, 17, 18, 19, 20, 21, 23, 24, 28).

PROGRAM STOPS:

Loc.	Meaning
$(L_o + 0161)_{10}$	Argument is larger than 1 at q = 1.

NOTE:

Since the square root subroutine is required for the evaluation of either arcsine or arccosine, the coding for the former (Program 15.0) is included in this program (20.0). In those instances in which the square root subroutine is independently required, the following calling sequence may be used for square root extraction:

Loc.	Inst.	Add.
$\alpha - 1$	B	L (Arg.)
α	R	$(L_o + 0150)_{10}$
$\alpha + 1$	U	$(L_o + 0100)_{10}$
$\alpha + 2$	etc.	

For further information on the square root subroutine see program 15.0.

DECIMAL MEMORY PRINTOUT
(Program 21.0)

FUNCTION:

To print the contents of consecutive memory locations in decimal form.

INPUT:

Beginning and final locations and the modifier (all in decimal).

OUTPUT FORMAT:

A. Locations:
The printed location is equal to the real location minus the modifier used.

B. Instructions:
1. With modifier subtracted if in the range
Modifier $\leq$ Address $\leq$ Final location.

2. If in the range Modifier > Address > Final location.
   a. Instructions are preceded by an "x" if modifier $\neq$ 0.
   b. Instructions are not preceded by an "x" if modifier = 0.

C. Data:
1. In decimal (at q=0) if transfer control button is up.
   a. Decimal data preceded by sign and decimal point.

2. In hexadecimal if transfer control button is down.
   a. Hexadecimal data preceded by a comma.

Output is six words per line preceded by initial location of the line. Words are separated by spaces. The sign and decimal point are printed for decimal data words and a comma is printed for hexadecimal words. Two carriage returns are given before and after printing.

Note:
Data printed in this manner can be converted to its real decimal value by multiplying by 2^q.

(Decimal memory printout continued.)

PROCEDURE:

1. Depress "manual input" button on the Flexowriter.
2. Transfer to the first location of this routine.
3. After the "manual input" light comes on, type the initial and final locations (in decimal) into the keyboard.
4. Depress the "Start Comp." button on the Flexowriter.
5. After a space is given and "manual input" light comes on again, type in modifier in decimal.
6. Make sure the "transfer control" is in the desired position - up for decimal data - down for hexadecimal.
7. Depress the "Start Comp." button on the Flexowriter.
8. The position of the "transfer control" button may be changed at any time to change the output format of non-instructional words.

TIME:

Approximately 60 words per minute.

STORAGE:

256 locations of instructions and constants (4 tracks).
No temporary storage.

COMPLEX OPERATION SUBROUTINE
(Program 22.0)

FUNCTION:

To interpret and execute the instructions B, A, S, M, D, H and C as if they were complex operation instructions referring to a two word abstract accumulator. To provide for shifting the abstract accumulators to the right or left from 0 to 10 places. To permit address modification of instructions and test for the final address without leaving the complex operation mode of programming.

INPUT:

Real and imaginary parts of a complex number must be carried at the same "q" and be in consecutive memory locations. (i.e. real in α; imaginary in $\alpha+1$).

OUTPUT:

Real and imaginary parts of a complex number placed in memory locations specified by the program. See programming section of this subroutine.

CALLING SEQUENCE:

Loc.	Inst.	Add.	
α	R	Lo	
$\alpha+1$	U	Lo	
$\alpha+2$			
$\alpha+3$			
.			complex operation instructions
.			
.			
$\alpha+n$	XE	0000	"exit" instruction
$\alpha+n+1$	etc.		

PROGRAMMING:

After executing the R Lo and U Lo instructions (where Lo is first instruction of the complex operation subroutine) the computer interprets and executes instructions as defined below. For simplicity "m" is defined as a complex memory address (i.e. memory location m and m + 1) and m' is defined as a standard one-word memory address.

ORDER	ADDRESS	INTERPRETATION
B	m	BRING Contents of m replaces the contents of the abstract accumulators.
A	m	ADD Contents of abstract accumulators plus contents of m replaces the contents of the abstract accumulators.
S	m	SUBTRACT Contents of abstract accumulators minus contents of m replaces the contents of the abstract accumulators.
M	m	MULTIPLY Contents of abstract accumulators times the contents of m replaces the contents of the abstract accumulators.
D	m	DIVIDE Contents of abstract accumulators divided by contents of m replaces the contents of the abstract accumulators.
H	m	HOLD Record the contents of the abstract accumulators into location m. Contents of abstract accumulators unchanged.
C	m	CLEAR Record contents of the abstract accumulators into memory location m. Abstract accumulators are then set to zero.
U	m	UNCONDITIONAL TRANSFER The next instruction to be interpreted is located in location m' The users attention is called to the fact that after the execution of this instruction the computer will continue to execute orders in the complex operation mode. This instruction may not be used as an exit from the subroutine.

ORDER	ADDRESS	INTERPRETATION
XE	0000	EXIT Exit from the complex operation mode of interpreting instruction and begin executing instructions in conventional "machine language" with instruction following XE 0000 instruction.

To facilitate the programmers task of address modification, this subroutine contains a special address accumulator. The following four instructions permit the programmer to perform address modification and test final address without leaving the complex operation mode.

ORDER	ADDRESS	INTERPRETATION
E	m	ENTER This instruction enters the address portion of the word at m into the address accumulator.
XI	$T_1T_2S_1S_2$	INCREMENT This instruction increments the address accumulator by T_1T_2 (track) and S_1S_2 (sector) leaving the adjusted address in the address accumulator.
Y	m′	STORE ADDRESS This instruction stores the address portion of the address accumulator in the address portion of memory location m′. The address accumulator in unaltered.
XZ	$T_1T_2S_1S_2$	ZERO TEST AND JUMP If the address portion of the address accumulator is equal to $T_1T_2S_1S_2$ the following instruction is skipped. When $T_1T_2S_1S_2$ differs from the address accumulator the instruction following XZ $T_1T_2S_1S_2$ is executed. Note that this comparison is based only on the address portions.

Since the basic arithmetic operation A, S, M, and D obey the conventional "q" laws, (as established under "Scaling" in this manual) it is still the responsibility of the programmer to provide the proper binal point manipulations. To facilitate shifting to the right or left, the following instructions are provided.

XR	00 n_1n_2	"Right Shift" -- This instruction will shift the abstract accumulators "n_1n_2" places to the right. Where "n_1n_2" is an integer in the range: $0 \leq n_1n_2 \leq 10$
XP	00 n_1n_2	"Left Shift" -- This instruction will shift the abstract accumulators "n_1n_2" places to the left. Where "n_1n_2" is an integer in the range: $0 \leq n_1n_2 \leq 10$

NOTES:

1. The transfer control button feature was not programmed into the subroutine. Use of the —T instruction will result in a halt.

2. In the explanation of orders above, use was made of the "X" to prevent modification of the corresponding address. If modification is desired do not precede the order with "X".

3. Use of the order "T" will result in a programmed halt.

4. Use of E0000 for the first complex operation instruction is forbidden.

5. Shifts exceeding 10 places will be incorrectly interpreted. The table may be expanded to include large shifts if the user desires.

Error Halts

Lo + 0122 (track 01 sector 22) — T instruction given.

Lo + 0154 (tract 01 sector 54) T instruction given.

Lo + 0135 (track 01 sector 35) N instruction given.

STORAGE:

192 locations of instructions and constants (3 tracks).
No temporary storage.

ACCUMULATOR LOCATIONS:

Lo + 0059 Real Accumulator
Lo + 0033 Imaginary Accumulator
Lo + 0219 (track 02 sector 19) Address Accumulator.

TIME:

The following table gives the approximate time required to execute each instruction. The times given are maximum times and in practice will be slightly less than the times given.

ORDER	DRUM REV.	TIME (ms.)
B	11	187
Y	8	136
R	14	238
I	6	102
D	41	697
N ≠	11	187
N =	13	221
M	22	374
P	17	289
E (enter)	9	153
E (exit)	6	102
U	8	136
C	14	238
H	14	238
A	14	238
S	14	238

4/12/57

FLOATING POINT INTERPRETIVE SYSTEM
(Program 24.0)

PART 1

SECTION I: FUNCTION

The function of this floating point system is the reinterpretation of the LGP-30 fixed point order structure so that it may be programmed as a floating point computer. This reinterpretation is effected by:

(1) The provision of a multiplier register and an address register as well as a floating point accumulator.

(2) The provision of more types of orders including cumulative multiply, shift, sign change, and function generating orders.

(3) A broadening of the scope of certain instructions such as the input instruction and the print instruction.

SECTION II: GENERAL CHARACTERISTICS

Floating point programming has several advantages over fixed point programming in that it is more rapid, does not require an exact knowledge of the range of magnitude of the variables, and does not involve as much truncation of the smaller values when that range is large. Thirty-three orders are provided for in the system. All of these orders except input, output, sine, cosine, arctangent, logarithm, and exponential are included in that section of the system known as the floating point interpretive routine. This routine requires only 10 out of the 64 tracks of LGP-30 memory leaving 3456 words available for problem program and data storage. The input and output orders require 6 tracks and the floating point functions require 7 tracks. The entire system leaves 2624 words of memory left for problem program and data storage.

Generally the execution of a floating point program will take 10 to 20 times as long as the execution of the corresponding fixed point program.

Times for the execution of the individual floating point orders are included in the summary tabulation at the end of part 2.

SECTION III: REGISTERS

(1) The floating point accumulator occupies 2 words of memory, one for the characteristic of a floating point number and one for the exponent. The floating point accumulator is similar in function to the fixed point accumulator; it holds intermediate results.

(2) The multiplier (M) register occupies 2 words of memory, one for the characteristic of a floating point number and one for the

exponent. The multiplier register holds the multiplier for the reset and multiply order and for the cumulative multiply order.

(3) The address accumulator occupies 1 word of memory and holds a single address or tally which is the same in form as for fixed point operations.

(4) The contents of none of these registers is changed unless replaced by a new result. For example, the M register remains unchanged following execution of a square root or multiply instruction. Nor is the contents of any memory location changed except when affected as specifically noted in the order description in the section that follows.

SECTION IV: FLOATING POINT ORDERS

Thirty-three orders are available. The list of these orders and their meaning follows. In the following exposition the term "accumulator" refers to the two memory cells of the floating point accumulator as defined above.

A. Arithmetic Instructions
Memory location XXXX is the address of one floating point number in standard form as defined in Part 2.

1. B XXXX. Bring
The contents of memory location XXXX replace the contents of the accumulator.

2. A XXXX. Add
The contents of the accumulator plus the contents of memory location XXXX replace the contents of the accumulator.

3. S XXXX. Subtract
The contents of the accumulator minus the contents of memory location XXXX replace the contents of the accumulator.

4. D XXXX. Divide
The contents of the accumulator divided by the contents of memory location XXXX replace the contents of the accumulator.

5. P XXXX. Place
The contents of memory location XXXX replace the contents of the M register.

6. M XXXX. Reset and Multiply
The contents of the M register multiplied by the contents of memory location XXXX replace the contents of the accumulator.

7. N XXXX. Cumulative Multiply
The contents of the M register multiplied by the contents of memory location XXXX and added to the contents of the accumulator replace the contents of the accumulator.

8. D 000y. Right Shift
The contents of the accumulator divided by 2^y replace the contents of the accumulator. The contents of accumulator remain in floating point form.
$0 \leq y \leq 9$

9. M 000y. Left Shift
The contents of the accumulator multiplied by 2^y replace the contents of the accumulator. The contents of accumulator remain in floating point form.
$0 \leq y \leq 9$

10. H XXXX. Hold
Place the contents of the accumulator in memory location XXXX.

11. C XXXX. Clear
Place the contents of the accumulator in memory location XXXX and set the accumulator to zero.

B. Logical or Transfer Instructions

12. U XXXX. Unconditional Transfer
The next instruction to be interpreted is in memory location XXXX. This order cannot be used to exit from the floating point interpretive system.

13. T XXXX. Test
The next instruction to be interpreted is in memory location XXXX if the accumulator is negative. Otherwise the first successive location will be interpreted.

14. 800T XXXX. Transfer Control
The next instruction to be interpreted will be in memory location XXXX if either the accumulator has a negative characteristic or the transfer control switch is down. Otherwise the first successive location will be interpreted.

C. Address Modification Instructions
Location XXXX implies a fixed point address.

15. E XXXX. Enter
The address portion of memory location XXXX replaces the contents of the address accumulator.

16. I XXXX. Increment
The address accumulator is incremented by the address XXXX. This order can be used to decrement the address accumulator by complementing the address portion of the I XXXX order.

17. Y XXXX. Store Address
The address portion of the address accumulator replaces the contents of the address portion of memory location XXXX.

18. Z XXXX. Zero Test
The address of the "Z" instruction is subtracted from the contents of the address accumulator. If the result is not zero, the first successive instruction is interpreted. If the result is zero, the first successive instruction is skipped and the second successive instruction is interpreted.

D. Auxiliary Instructions

19. R XXXX. Return Address
The location of this instruction is increased by 2 and is stored in the address portion of memory location XXXX.

20. U 0000. Reverse Registers
The contents of the M register and accumulator are interchanged.

21. B 0000. Set Sign Plus
The sign of the accumulator is made positive if not already so.

22. T 0000. Set Sign Minus
The sign of the accumulator is made negative if not already so.

23. Y 0000. Change Sign
The sign of the acumulator is reversed.

24. Z 0000. Stop
Computation is halted unless break point switch No. 16 is down. Depressing the start button causes the next instruction to be interpreted.

25. E 0000. Exit
Exit from the floating point interpretive system. Control is returned to the location following the location of the E 0000 instruction.

E. Input-Output Instructions

26. I 0000. Input
Control is transferred to a floating point data input subroutine which reads decimally punched numbers on tape, converts them to floating binary, and stores them. The next instruction is interpreted after the proper exit code has been read from tape. See Section V, Part 1 for tape format and input details.

27. P 0000. Print
Print the contents of the accumulator. The contents of the accumulator are not destroyed. See Section VI, Part 1 for Output format.

F. Function Evaluation Instructions

28. R 0000. Square root
The square root of the contents of the accumulator replaces the contents of the accumulator.

29. S 0000. Sine
The sine of the contents of the accumulator replaces the contents of the accumulator. The accumulator must be in radian measure.

30. C 0000. Cosine
The cosine of the contents of the accumulator replaces the contents of the accumulator. The accumulator must be in radian measure.

31. A 0000. Arctangent
The arctangent of the contents of the accumulator replaces the contents of the accumulator. Output is in radian measure.

32. N 0000. Natural Logarithm
The natural logarithm of the contents of the accumulator replaces the contents of the accumulator.

33. H 0000. Exponential
The quantity e^x replaces the contents of the accumulator, where x is initially the contents of the accumulator.

SECTION V: DATA INPUT FORMAT

Data input is accomplished by reading a prepunched decimal tape. The tape consists of groups of the following:

1. One identification word. This consists of a sign and two decimal digits for P, followed by four decimal digits for initial location to begin storing the converted floating point binary numbers.

2. Signed decimal numbers. Each number consists of a sign (if negative) and seven decimal digits.

3. A "minus zero" word. This consists of a minus sign followed by seven zeros. This number is not stored in memory, but is used by the routine to signal the end of the group.

A stop code must follow the last "minus zero" word. This is interpreted as a "zero" identification (I.D.) word since it follows the "minus zero" data word. It causes the system to exit from the subroutine, carriage return, and interpret the instruction following the I 0000 instruction.

P denotes the number of decimal places following the point in the seven digit field. $-3 \leq P \leq 15$. Internally the exponent must be in the range $-31 \leq \text{Exp.} \leq 31$.

SECTION VI: DATA OUTPUT FORMAT

The printed output consists of a decimal point followed by seven decimal digits of the characteristic and its sign. Following the sign there are two spaces followed by the exponent and its sign (if the sign is negative). e.g. .5060000- 02 is -50.60000. A tab is executed after printing.

FLOATING POINT INTERPRETIVE SYSTEM
(Program 24.0)

PART 2

Note: Refer to Part 1 for FUNCTION, REGISTERS, ORDERS, and INPUT and OUTPUT FORMAT.

INPUT:

Floating point numbers on tape or in memory, or numbers in the pseudo registers resulting from previous operations.

CALLING SEQUENCE:

Loc.	Inst.	Add	
α	R	Lo	
α + 1	U	Lo	
α + 2	.	.	
α + 3	.	.	
.	.	.	Floating point operations
.	.	.	
.	.	.	
α + n	E	0000	"Exit" instruction
α + n + 1	etc.		Resume fixed point operation.

INTERNAL NUMBER FORMAT:

A standard floating point number as carried in memory consists of sign and 24 bits for characteristic (x) and sign and 5 bits for the exponent (y). However, all intermediate calculations (i.e., numbers appearing only in accumulator and multiplier registers) are carried with 30 bits of characteristic and 30 bits of exponent. Each factor of any calculation must be in standard floating point form. ($N = x.2^y$; $.5 \leq |x| < 1.$ or $x = 0$; $-31 \leq y \leq 31$). Numbers appearing in accumulator or M registers are in the range $.25 \leq |x| < .5$ or $x = 0$.

The standard floating point binary form:

X	.XXX.....XX	X	XXXXX
Sign of Characteristic 0 for plus 1 for minus	Characteristic 24 bits.	Sign of exponent 0 for plus 1 for minus	Exponent 5 bits Power of2

DATA TAPE PREPARATION:

1. All characters of the I. D. word should be punched. e.g. -012040' must contain eight characters including the stop code. The stop code (') must be the last character punched.

2. Punch only those I.D. words appearing on the load sheet. Do not punch the stop code if an I.D. word is not present.

3. The sign and any leading zeros of a positive number need not be punched. To enter all zeros merely punch a stop code. The sign and all seven digits of a negative number must be punched.

4. Be sure to check each load sheet to see whether an additional stop code should follow the last number punched.

EXIT:

The interpretive routine exits to the first location following the E 0000 instruction.

SUBROUTINE MEMORY RELATIONSHIPS:

The arithmetic, logical, address modification, and auxiliary instructions have been coded as a unified group on a single set of coding sheets ("Floating Point Interpretive Routine"). A single corresponding tape has been punched for this set. In many instances the programmer will wish to use just this part of the floating point system; if so, only this tape need be stored in the memory. This will leave 54 tracks for program instructions and data in contrast to 41 when the entire system is used.

In other cases the Input-Output and/or function evaluation routines may be needed. Only those routines actually used need be stored on the drum. These required routines must be stored on the drum in the following relationship:

Program	Routine	Start Fill	Set Modifier	No. of Tracks
24.0	Interpretive (Includes $\sqrt{}$)	Lo	Lo	10
11.3-12.3	Input-Output	Lo + 1000	Lo + 1000	6
14.1	Sine-Cosine	Lo + 1600	Lo	2 1/2
16.2	Arctangent	Lo + 1832	Lo	1 1/2
18.1	Logarithm	Lo + 2000	Lo	1
17.1	Exponential	Lo + 2100	Lo	2

All track 63 except sectors 10, 15, 16, 18, 23, 27, 29, 34, 36, 40, 47 thru 50, 52, 56 thru 58, 60 and 63 is used for temporary storage by various parts of the system. Therefore Lo should be set such that no part of the floating point system used is stored in track 63.

PROGRAM STOPS:

Loc.	Order	Meaning and Remedy
Lo + 0654	Z 0000	Programmed stop. Depress "start" to continue.
Lo + 0556	H XXXX or C XXXX	Exponent is too large. Location of instruction being executed is in the real accumulator. Start to continue.

Lo + 0556	R 0000	Accumulator is negative. Location of instruction being executed is in the real accumulator. Start to continue.
Lo + 1152	I 0000	Input data has too large an exponent. A start will store a zero for that word and continue with next word on tape.
Lo + 0612	D XXXX	Division by zero or a non-floated number. Do not continue.
Lo + 2005	N 0000	Accumulator is ≤ 0. A start continues with an answer of zero.
Lo + 2028 or Lo + 2030	N 0000	Accumulator exponent is not in range. Do not continue.

TIME:

See summary tabulation.

EXAMPLE:

See the following LGP-30 coding sheet.

NOTES:

1. The floating point system may be left and re-entered without destroying the contents of the registers.

2. The exponent of a number in a register which is to be stored in memory must be less than +32, or a range error will result. If it is less than -31, the number is replaced by zero.

3. It is strongly suggested that the initial location occupied by the system be the 00 sector of a track. If it is not, many of the addresses that refer to track 63 are not optimum.

4. It is also suggested that the entire system be placed in memory and punched out in parts by program 13.1. Then the parts needed may be loaded by program 10.1 and each check sum may be verified.

5. All instructions with zero addresses have special interpretations. None of these zero addresses refer to memory location "zero", (0000), but rather designate a special interpretive instruction. This floating point system employs sixteen such special instructions. Furthermore, the two shift instructions (D 000y, M 000y) utilize the next nine addresses (0001 through 0009); hence the divide and reset and multiply instructions cannot use these addresses.

SUMMARY TABULATION

ORDER	RESULT IF ADDRESS (α) $\neq$ 0	TIME	RESULT IF ADDRESS = 0*	TIME
Z	C(Add. Acc.) − (α) = 0? No: No skip yes: Skip	133 ms	Stop (SW No. 16). Proceed on start	117 ms
B	C(α) ⟶ Acc.	233 ms.	Make C(Acc.) positive	150 ms
Y	C(Add. Acc.) ⟶ Add. of (α)	150 ms.	Complement C(Acc.)	150 ms
R	(Loc. of R) + 2 ⟶ Add. of (α)	166 ms.	$\sqrt{C\ (Acc.)}$ ⟶ Acc.	500 ms
I	C(Add. Acc.) + (α) ⟶ Add. Acc.	150 ms	Input floating point data	40/min.
D	C(Acc.) ÷ C(α) ⟶ Acc.	283 ms.	C(Acc.) ÷ 2^{α} ⟶ Acc.	183 ms
N	C(M) x C(α) + C(Acc.) ⟶ Acc.	566 ms	ln C(Acc.) ⟶ Acc.	500 ms
M	C(M) x C(α) ⟶ Acc.	266 ms.	C(Acc.) x 2^{α} ⟶ Acc.	150 ms
P	C(α) ⟶ M	217 ms.	Print C(Acc.)	1.85 sec
E	C[Add.(α)] ⟶ Add. Acc.	150 ms.	Exit from interpretive routine	117 ms
U	Next abstract order taken from (α)	117 ms.	C(Acc.) ⟶ M; C (M) ⟶ Acc.	200 ms
T	Transfer if C(Acc.) is negative	133 ms.	Make C(Acc.) negative	150 ms
H	C(Acc.) ⟶ (α)	200 ms.	$e^{C(Acc.)}$ ⟶ Acc.	450 ms
C	C(Acc.) ⟶ (α); 0 ⟶ Acc.	233 ms.	Cosine C(Acc.) ⟶ Acc.	517 ms
A	C(Acc.) + C(α) ⟶ Acc.	400 ms.	Arctangent C(Acc.) ⟶ Acc.	450 ms
S	C(Acc.) − C(α) ⟶ Acc.	417 ms.	Sine C(Acc.) ⟶ Acc.	550 ms

Add. Acc. = Address Accumulator register
M = Multiplier register
Acc. = Floating point Accumulator
α = any address
C = Contents of

*Address = 0: except for instructions "M 000α" and "D 000α", where $0 \leq \alpha \leq 9$.
⟶ = Is stored in

All times are approximate and will vary with the amount of overflow and/or underflow. Actual times should be slightly less than listed. Time will usually be reduced if any factor is zero.

ILLUSTRATIVE EXAMPLE FOR FLOATING POINT INTERPRETIVE SYSTEM

LGP-30 CODING SHEET

Job No. OIVE Prog. No. 24.0 Prep. by G.L.W. Ck'd. by M.K. Date 14 June '57

Problem CODING FOR 4th Degree Polynomial Track

Program Input Codes	Stop	Location	Instruction Op.	Address	Stop	Contents of Address	Notes
3 0 0 0 2 5 0 0	'						
1 0 0 0 2 5 0 0	'	☒					
		0 0	X R	3 0 0 0	'		Enter Interpretive
		0 1	X U	3 0 0 0	'		Routine
		0 2	X I	0 0 0 0	'		Input Coefficients
		0 3	E	0 0 1 7	'	☒ 2005	Set
		0 4	Y	0 0 0 6	'	2005	Initial Address
		0 5	B	0 0 1 3	'	Zero	
		0 6	X A	[ ]	'	a_n	Add n' th coefficient
		0 7	X I	0 0 0 1	'	☒ a[2005+n]	Increase address accumulator by unity
		0 8	Y	0 0 0 6	'	a[2005 +n]	Store contents of address accumulator in 0006
		0 9	X Z	2 0 1 0	'		Test for finish.
		1 0	U	0 0 1 4	'	Not finished	
		1 1	X P	0 0 0 0	'	☒ Print result here: Finished	
		1 2	X E	0 0 0 0	'	Exit	
		1 3	X Z	0 0 0 0	'	Stop	Fixed pt. inst.
		1 4	X U	0 0 0 0	'	Move Accum. to M Register	
		1 5	X M	2 0 0 4	'	☒ Multiply by x	
		1 6	U	0 0 0 6	'	Return for next term.	
		1 7	X Z	2 0 0 5	'	Initial coefficient	
		1 8			'		

Floating point Data Input

Quan.	± P	Location	Stop	± Number	Stop	Car. Ret.
x	+ 0 7	2 0 0 4	'	1 0 9 0 0 0 0	'	
a_0			'	1 2 0 0 0 0	'	
a_1			'	3 4 0 0 0 0	'	
a_2			'	5 6 0 0 0 0	'	☒
a_3			'	7 8 0 0 0 0	'	
a_4			'	9 0 0 0 0 0	'	
			'	- 0 0 0 0 0 0 0	'	
			'		'	☒

-36P&15

0000≤Loc≤6363

Punch a stop code after the last number? Yes X No