PART 1 OF LLL 8080 BASIC INTERPRETER

By Jerry Barber & Royce Eckard

Submitted by E. R. Fisher
Lawrence Livermore Laboratory

FOREWARD

The BASIC interpreter was developed at the University of Idaho by John Dickenson, Jerry Barber, and John Teeter under a contract with the Lawrence Livermore Laboratory. The floating point package was developed by David Mead, modified by Hal Brand and Frank Olken. In addition, Jerry Barber, as an LLL summer employee, made significant contributions to this document and to implementing the BASIC language in an MCS-8080 microprocessor.

INTRODUCTION

This article is Part 1 of a series of four articles covering the LLL 8080 BASIC interpreter just released to the public domain by Lawrence Livermore Laboratory. The other three articles that will be published in the next three months are:

PART 2 — LLL 8080 BASIC INTERPRETER SOURCE PROGRAM WITHOUT FLOAT
PART 3 — LLL 8080 BASIC FLOAT SOURCE PROGRAM
PART 4 — LLL 8080 OCTAL DEBUGGING SOURCE PROGRAM

The partition approach of publishing the complete 120 page LLL BASIC interpreter source program assembly listing and descriptive text is taken as the only logical way to transfer the complete source program and text to INTERFACE AGE readers.

STORAGE REQUIREMENTS

The BASIC interpreter consists of a 5K-byte-PROM resident interpreter used for program generation and debug was configured to operate with the MCS-8080 microprocessor.

The goal in developing the 8080 BASIC was to provide a high-level, easy-to-use conversational language for performing both control and computation functions in the MCS-8080 microprocessor. To minimize system memory size and cost, the interpreter was constrained to fit into 5K bytes. It was necessary, therefore, to limit the commands to those considered the most useful in microprocessor applications.

MATH OPERATOR EXECUTION TIMES

Average execution times of the four basic math operators are as follows;

Operation	Execution time on 8080 (m sec)
ADD	2.4 m sec
SUBSTRACT	2.4 m sec
MULTIPLY	5.4 m sec
DIVIDE	7.0 m sec

BASIC INTERPRETER LANGUAGE GRAMMAR

COMMANDS — Six BASIC interpreter commands are provided. These commands are:

RUN	Begins program execution
SCR	Clears program from memory
LIST	Lists ASCII program in memory
PLST	Punches paper-tape copy of program
PTAPE	Reads paper-tape copy of program using high-speed reader
CNTRL S	Interrupts program during execution

The LIST and PLST commands can be followed by one or two line numbers to indicate that only a part of the program is to be listed. If one line number follows the command, the program is listed from that line number to the end of the program. If two line numbers (separated by a comma) follow the command, the listing begins at the first line number and ends at the second.

When a command is completed, READY will be typed on the teletype. Once initialized by a command, a process will normally go to completion. However, if you wish to interrupt an executing program or a listing, simply strike CNTRL S and the process will terminate and a READY message will be typed.

STATEMENTS — Each statement line begins with a line number, which must be an integer between 0 and 32767. Statements can be entered in any order, but they will be executed in numerical order. All blanks are ignored. The following types of statements are allowed:

REM — Indicates a remark (comment). The system deletes blanks from all character strings that are not enclosed in quotes ("). Therefore, it is suggested that characters following the REM key word be enclosed in quotes.

END — Indicates the end of a program. The program stops when it gets to the END statement. All programs must end with END.

Happy Holidays

STOP — Stops the program. This statement is used when the program needs to be stopped other than at the end of the program text.

GOTO — Transfers program control to specified statement line number. This statement is used to loop or jump unconditionally within a program. Program execution continues from new statement.

DIM — Declares an array. Only one-dimensional arrays with an integer constant number of elements are allowed. An array with N elements uses indexes 0 through N-1. All array locations are set to zero. No check is made on subscripts to ensure that they are within the declared array. An array variable must be a single letter.

LET — Indicates an assignment statement (Addition, subtraction, multiplication, division, or special function may be used). The LET statement is used to assign a value to a variable. Non-array variables can be either a single letter or a letter followed by a digit. It is possible to have an array and a non-array variable with the same name. The general form of the LET statement is:

line number LET identifier = expression,

where "identifier" is either a subscripted array element or a non-array variable or function (see section on functions) and "expression" is a unary or binary expression. The expression will be one of the following ten types:

variable
−variable
variable + variable
variable − variable
−variable + variable
−variable − variable
variable * variable
−variable * variable
variable / variable,
−variable / variable,

where "variable" is an identifier, function, or number. The subscript of an array can also be an expression.

IF — Condition statement which transfers to specified line number statement if the condition of the expression is met. It has the form: line number IF expression relation expression THEN transfer line number. The possible relations are:

Equal	=
Greater than	>
Less than	<
Greater than or equal	>= =<
Less than or equal	<= =<
Not equal	<> ><

If the relation between the two expressions is true then the program transfers to the line number, otherwise it continues sequentially.

INPUT — This command allows numerical data to be input via the teletype. The general form is:

Line number INPUT identifier list,

where an "identifier list" is a sequence of identifiers separated by commas. There is no comma after the last identifier so, if only one identifier is present, no comma is needed. When an INPUT statement is executed, a colon (:) is output to the teletype to indicate that data are expected. The data are entered as numbers separated by commas. If fewer data are entered than expected, another colon is output to the teletype, indicating again that data are expected. For example, where

50 INPUT I,J,K,P

is executed, a colon is output to the teletype. Then, if only 3 numerical values are entered, another colon will be output to indicate that more data are expected; e.g.,

: 4,4,6.2 C/R
: 10.3 C/R,

where C/R is the carriage-return key. If an error is made in the input-data line, an error message is issued and the entire line of data must be re-entered. If, for the above example,

: 4,4,6M2,10.3 C/R

is entered, the system will respond

INPUT ERROR, TRY AGAIN
:

At this time, the proper response would be

4,4,6.2,10.3 C/R.

PRINT — This command allows numerical data and character strings to be printed on the teletype. Two types of print items are legal in the print statement: character strings enclosed in quotes (") and expressions. These items are separated by either a comma or a semicolon. If print items are separated by a comma, a skip occurs to the next pre-formatted field before printing of the item following the comma begins. The pre-formatted fields begin at columns 1, 14, 27, 40, and 52. If print items are separated by a semicolon, no skip occurs. If a semicolon or comma is the last character on a print statement line, the appropriate formatting occurs and the carriage-return-line feed is suppressed. A print statement of the form

50 PRINT

will generate a carriage-return-line feed. Thus, the two lines below

50 PRINT "INPUT A NUMBER";
60 INPUT A

will result in the following output:

INPUT A NUMBER:

FOR — Causes program to iterate through a loop a designated number of times.

NEXT — Signals end of loop at which point the computer adds the step value to the variable and checks to see if the variable is still less than the terminal value.

GOSUB — Transfer control to a subroutine that begins at specified line number

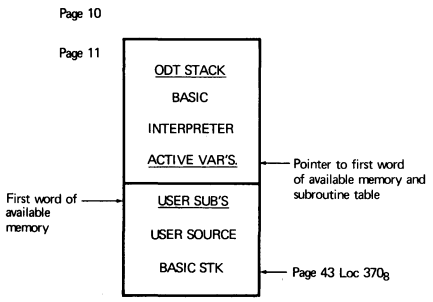
RETURN — Returns control to the next sequential line

after the last GOSUB statement executed. A return statement executed before a GOSUB is equivalent to a STOP statement.

CALL — Calls user-written assembly-language routines of the form

CALL (N, A, B, . . .),

where N is a subroutine number from 0 - 254 and A, B, . . . are parameters. The parameters can be constants, variables, or expressions. However, if variables and constants or expressions are intermixed, all variables should have been referenced before the CALL statement. Otherwise, the space reserved for newly referenced variables may overwrite the results of constants and expressions. A memory map of one configuration of the system is shown below:



The subroutine table contains 3-byte entries for each subroutine. The table directly follows the pointer to the first word of available memory (FWAM) and must end with an octal 377. A sample table and its subroutines is shown below:

```
        ORG 16612Q
        DW SUBEND       ; Define FWAM
        DB 1            ; Subroutine #1
        DW SUB1         ; Starting add of
                        subroutine #1
        DB 4            ; Subroutine #4
        DW SUB4         ; Starting add of
                        subroutine #4
        DB 5            ; Subroutine #5
        DW SUB5         ; Starting add of
                        subroutine #5
        DB 2            ; Subroutine #2
        DB SUB2         ; etc.
        DB 377Q         ; end of subroutine table
SUB1:   ↓               ; Subroutine #1
        RET
SUB5:   ↓               ; Subroutine #5
        RET
        •
        •
        •
        RET             ; Retain last subroutine
SUBEND  EQU $           ; FWAM
```

Addresses to passed parameters are stored on the stack. The user must know how many parameters were passed to the subroutine. These must be taken off the stack before RET is executed. Addresses are stored last parameter first on the stack. Thus, on entry to a subroutine, the first POP instruction will recover the address to the last parameter in the call list. The next will recover the next to last, etc.

Each scalar variable passed results in the address to the first byte of a four-byte block of memory. Each array element passes the address to the first byte of a (N-M) x four-byte memory block, where N is the number of elements given the array in the DIM STMT and M is the array subscript in the CALL STMT.

For passed parameters to be handled in expressions within BASIC, they must be in the proper floating-point format.

FUNCTIONS — Two special functions not found in most BASIC codes are available to input or output data through Intel 8080 port numbers. These functions are;

GET (X) = READ 8080 INPUT PORT X.
PUT (Y) = OUTPUT A BYTE OF DATA TO OUTPUT PORT Y.

The function GET allows input from a port and the function PUT allows output to a port. Their general forms are:

GET (expression).
PUT (expression).

The function GET may appear in statements in a position that implies that a numerical value is used. The function PUT may appear in statements in a position that implies that a numerical value will be stored or saved. This is because GET inputs a number and PUT outputs a number. For example, while

LET PUT(I) =GET(J) is valid
LET GET(I) =PUT(J) is invalid.

These functions send or receive one byte of data, which in BASIC is treated as a number from 0 to 255.

VARIABLES — Single characters A → Z
Single character followed by a signal decimal digit

NUMBERS — Numbers in a program statement or input via the teletype are handled with a floating-point package provided by LLL. Numbers can have any of the following forms:

4	±4.	.123
4.	±4.0	±.123
4.0	1.23	0.123
±4	±1.23	±0.123

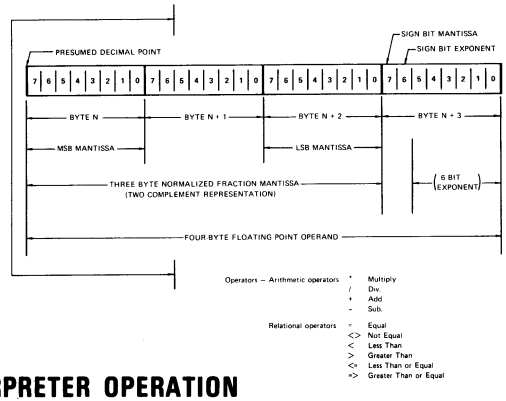
and the user may add an exponent to any of the above forms using the letter E to indicate powers of 10. The forms of the exponent are:

E±1	E±15
E 1	E 15
E 1	E 15

The numbers are stored with seven-digit accuracy; therefore, seven significant figures can be entered. The smallest and largest numbers are ±2.71051E-20 and ±9.22337E18.

Floating point numbers are expressed as a 32 bit operand consisting of a 24 bit normalized fractional mantissa in standard two's complement representa-

tion and a 6 bit exponent also in standard two's complement representation with a range of −64 to +63. The exponent byte also includes the exponent sign bit and mantissa sign bit. The floating point number format is as shown in the following:



INTERPRETER OPERATION

INITIALIZATION — the BASIC interpreter is presently configured so that it is located in memory pages 118 to 348. The starting address is page 178, location 0. This address begins an initialization sequence that allows the user to begin with a clear memory. However, to avoid the initialization sequence, a second starting address — page 178 to 348 — can be used. This starting address is used if the user wishes to retain any program that might exist in memory.

Once started the interpreter responds with READY.

INPUT LINE FORMAT

Each line entered is terminated with the carriage-return key. The line-feed key is ignored. Carriage-return automatically step terminal to next line and waits for next line statement number input. Statements can be entered in any order, but they will be executed in numerical order. All blanks outside of quotation marks are ignored by the interpreter. Up to 72 characters may be entered/line.

INPUT LINE EDITORING — A program can be edited by using the line numbers to insert or delete statements. Typing a line number and then typing a carriage return causes the statement at that line number to be deleted. Since the statements can be entered in any order, a statement can be inserted between two existing statements by giving it a line number between the two existing statement line numbers. To replace a statement, the new statement should have the same line number as the old statement.

It is possible to correct errors on a line being entered by either deleting the entire line or by deleting one or more characters on the line. A character is deleted with either the rubout key or the shift/0 key. Several characters can be deleted by using the rubout key several times in succession. Character deletion is, in effect, a logical backspace. To delete the line you are currently typing, use the CNTRL/Y key.

BASIC PROGRAM EXECUTION — Entering a RUN command, after a BASIC program has been entered into the microcomputer, will cause the current program to begin execution at the first statement number. RUN always begins at the lowest statement number.

ERROR MESSAGES — If an unrecognizable command is entered, the word WHAT? is printed on the teletype. Simply retype the command. It may also have been caused by a missing line number on a BASIC statement, in which case you should retype the statement with a line number.

During program execution and whenever new lines are added to the program, a test is made to see if there is sufficient memory. If the memory is full, MEMORY FULL is printed on the teletype. At this point, you should enter one of the single digits below to indicate what you wish to do:

Number entered	Meaning
0	**(RUN) runs**
0	(RUN) runs the program in memory
1	(PLST) outputs program in memory to paper tape punch
2	(LIST) lists program in memory
3	(SCR) erases program in memory
4	none of the above (will case WHAT? to be printed out on the tl out on the teletype).

To help you select the best alternative, a brief description of how the statements are manipulated in memory will be helpful. All lines entered as program are stored in memory. If lines are deleted or replaced, the originals still remain in memory. Thus, it is possible, if a great deal of line editing has been done, to have a significant portion of memory taken up with unused statements. If a MEMORY FULL message is obtained in these circumstances, then the best thing to do is punch a tape of the program (entering number 1), then erase the program memory with a SCR command (or a number 3, if memory is too full to accept commands), and then re-enter your program using the high-speed paper-tape reader with the PTAPE command.

If an error is encountered while executing a program, an error message is typed out that indicates an error number and the line number in which the error occurred. These numbered error messages are as follows;

ERROR NUMBER
ERROR MESSAGE

1 Program has no END statement
2 Unrecognizable keyword at beginning of statement
3 Source statements exist after END statement
4 Designation line number is improperly formed in a GOTO, GOSUB, or IF statement
5 Designation line number in a GOTO, GOSUB, or IF statement does not exist
6 Unexpected character
7 Unfinished statement
8 Illegally formed expression
9 Error in floating-point conversion
10 Illegal use of a function
11 Duplicate array definition
12 An array is referenced before it is defined
13 Error in the floating-point-to-integer routine, Number is too big
14 Invalid relation in an IF statement

LLL 8080 BASIC INTERPRETER PROGRAMS EXAMPLES

PRINT STATEMENT PROGRAM EXAMPLE — The program below gives a few examples of the use of the print statement.

```
LIST
1PRINT"THE PRE-FORMATTED COLUMNS ARE SHOWN BELOW"
2PRINT1,2,3,4,5
4PRINT
1ØPRINT"INPUT 1ST NUMBER";
2ØINPUTA
3ØPRINT"INPUT 2ND NUMBER",
4ØINPUTB
5ØPRINT
6ØPRINT"A IS";A
7ØPRINT"B IS",B
8ØPRINT"A IS";A;"B IS",B,"A+B IS";A+B
1ØØEND
READY
RUN
THE PRE-FORMATTED COLUMNS ARE SHOWN BELOW
1.ØØØØE ØØ   2.ØØØØE ØØ   3.ØØØØE ØØ   4.ØØØØE ØØ   5.ØØØØE ØØ

INPUT 1ST NUMBER:2
INPUT 2ND NUMBER          :3

A IS 2.ØØØØE ØØ
B IS         3.ØØØØE ØØ
A IS 2.ØØØØE ØØB IS        3.ØØØØE ØØ   A+B IS 5.ØØØØE ØØ
READY
```

PLOT FUNCTION PROGRAM — The following program plots a function on a display. It uses four user-written assembly-language subroutines. The display works as follows: The contents of memory locations on pages 2748 to 2778 are displayed as 16 rows of 64 characters each. Thus, if location 2018 on page 274 contains 3018 (ASCII A), an A appears in column 2 of Row 3. An example of this program's execution is shown below:

RUN
WHAT SHOULD PLOT BE LABELED? MCS80 — BASIC INTERPRETER
READY

The BASIC and assembly-language programs and the display output are shown below.

BASIC PROGRAM

Display output for Plot Function program.

BASIC Program

```
LIST
1REM"   THIS ROUTINE WILL PLOT A SET OF AXIS AND A QUADRATIC FUNCTION
2REM"   ON A DISPLAY AND THEN LABEL IT.  IT USES A 4 USER WRITTEN
3REM"   SUB-ROUTINES:
4REM
5REM"   CALL (1,X,Y,C) - PLACES C IN COLUMN X, ROW Y OF THE DISPLAY
6REM"     WHERE C IS AN ASCII CODED CHARACTER
7REM
8REM"   CALL(2,A(Ø)) - READS A CHARACTER STRING FROM THE TTY AND STORES
9REM"     IT"IN ARRAY A
10REM
11REM"  CALL(3,A(Ø)) - WRITES THE CHARACTER STRING STORED IN ARRAY A
12REM"    TO THE DISPLAY
13REM
14REM"  CALL(4) - CLEARS THE DISPLAY
15REM
16REM"    START OF PROGRAM
17REM
18REM"    RESERVE STORAGE AREA FOR TITLE
2ØDIMA(1Ø)
3ØREM"    CLEAR SCREEN
4ØCALL(4)
5ØREM"    ASK FOR AND INPUT TITLE
55PRINT"WHAT SHOULD PLOT BE LABELED?";
6ØCALL(2,A(Ø))
7ØREM"    DRAW AXIS
8ØGOSUB5ØØ
9ØREM"    PLOT FUNCTION
1ØØLETX=-29
11ØGOSUB1ØØØ
12ØCALL(1,31+X,8-Y,248)
13ØLETX=X+1
14ØIFX><31 THEN11Ø
15ØREM"   OUTPUT TITLE
16ØCALL(3,A(Ø))
165REM"   WE'RE DONE
17ØSTOP
5ØØREM"   THIS SUB. WILL DRAW A SET OF AXIS
5Ø5LETX=1
51ØLETY=7
52ØLETC=173
53ØCALL(1,X,Y,C)
54ØLETX=X+1
55ØIFX><65THEN53Ø
56ØLETX=31
57ØLETY=1
575LETC=252
58ØCALL(1,X,Y,C)
59ØLETY=Y+1
6ØØIFY><17THEN58Ø
61ØRETURN
1ØØØREM"  GIVEN X THIS SUB. CALCULATES (17/9ØØ)*X**2-8
1ØØ5REM"  FIRST CHECK IF X=Ø AS IT WILL UPSET FLT. PNT. PACK.
1Ø1ØIFX=ØTHEN1Ø45
1Ø15REM"  WE'RE OK - CALCULATE FUNCTION
1Ø2ØLETY=X*X
1Ø25LETK=17/9ØØ
1Ø3ØLETY=Y*K
1Ø35LETY=Y-8
1Ø4ØRETURN
1Ø45LETY=-8
1Ø5ØRETURN
2ØØØEND
READY
```

Assembly-language program

```
                        ;DEFINE EXTERNALS
014012                  FIX     EQU     14012Q          ;FIX ROUTINE
013212                  COPDH   EQU     13212Q          ;COPY ROUTINE
016567                  FREG1   EQU     16567Q          ;FLOATING PNT  REGISTER
016614                          ORG     16614Q
016614  027 036                 DW      SBEND           ;FWAM
                        ;ENTRIES IN SUB TABLE
016616  001                     DB      1
016617  233 035                 DW      SCOPE
016621  002                     DB      2
016622  334 035                 DW      SUB2
016624  003                     DB      3
016625  364 035                 DW      SUB3
016627  004                     DB      4
016630  003 036                 DW      SUB4
016632  377                     DB      377Q            ;NO MORE ENTRYS
                        ;THE CALL TO THIS ROUTINE IS OF THE FORM
                        ;       CALL(1 X Y C)
                        ;THE VALUE OF C IS PLACED IN COLUMN X  LINE Y
                        ;OF THE DISPLAY
016633  321             SCOPE:  POP     D               ;ADDRESS OF CHARACTER
016634  041 167 035            LXI     H FREG1         ;COPY TO FREG1
016637  315 212 026            CALL    COPDH
016642  353                    XCHG                    ;ADDRESS TO DE
016643  315 012 030            CALL    FIX             ;FIX IT
016646  023                    INX     D               ;PNT TO 4TH BYTE
016647  023                    INX     D
016650  023                    INX     D
016651  032                    LDAX    D               ;GET CHARACTER
016652  107                    MOV     B A             ;SAVE IN B
016653  321                    POP     D               ;ROW ADD
016654  041 167 035            LXI     H FREG1         ;COPY TO FREG1
016657  315 212 026            CALL    COPDH
016662  353                    XCHG
016663  315 012 030            CALL    FIX             ;FIX IT
016666  023                    INX     D               ;GET BYTE 4 TO A
016667  023                    INX     D
016670  023                    INX     D
016671  032                    LDAX    D
016672  117                    MOV     C A             ;SAVE IN C
016673  321                    POP     D               ;GET COLUMN ADD
016674  041 167 035            LXI     H FREG1         ;COPY TO FREG1
016677  315 212 026            CALL    COPDH
016702  353                    XCHG
016703  315 012 030            CALL    FIX             ;FIX IT
016706  023                    INX     D               ;PNT TO 4TH BYTE
016707  023                    INX     D
016710  023                    INX     D
016711  032                    LDAX    D               ;GET IT TO A
016712  041 377 273            LXI     H 1357777Q      ;CALCULATION OF ADDRESS
016715  021 100 000            LXI     D 100O
016720  015             LUP:    DCR     C
016721  312 330 035            JZ      ADINC
016724  031                    DAD     D
016725  303 320 035            JMP     LUP
016730  137             ADINC:  MOV     E A
016731  031                    DAD     D               ;ADD IN COLUMN LOC
016732  160                    MOV     M B             ;STORE CHARACTER
016733  311                    RET                     ;DONE
                        ;SUB2 READS A TITLE FROM TTY VIA ODT
000333                  READ    EQU     333Q            ;ODT ROUTINE
016734  341             SUB2:   POP     H               ;GET STORAGE AREA ADD
016735  345                     PUSH    H
016736  016 000                 MVI     C Ø             ;INIT CNTR
016740  043             LUP2:   INX     H               ;BUMP PNTR
016741  315 333 000             CALL    READ            ;READ A CHARACTER
016744  376 215                 CPI     215Q            ;CR?
016746  312 356 035             JZ      DUN2            ;YES - DONE
016751  014                     INR     C               ;INCR  CNT
016752  167                     MOV     M A             ;SAVE CHARACTER
016753  303 340 035             JMP     LUP2
016756  341             DUN2:   POP     H               ;STORE CNT
016757  161                     MOV     M C
016760  076 212                 MVI     A 212Q          ;SEND A LF
016762  367                     RST     6
016763  311                     RET                     ;DONE
                        ;SUB3 WRITES TITLE TO DISPLAY
016764  341             SUB3:   POP     H               ;GET ADD
016765  021 341 277             LXI     D 137741Q       ;SCREEN ADD
016770  116                     MOV     C M             ;CNT
016771  043                     INX     H
016772  176             LUP3:   MOV     A M             ;SEND STRING
016773  022                     STAX    D
016774  043                     INX     H
016775  023                     INX     D
016776  015                     DCR     C
016777  302 372 035             JNZ     LUP3
017002  311                     RET                     ;DONE
                        ;SUB4 CLEARS SCREEN
017003  041 000 274 SUB4:   LXI     H 136000Q       ;SCREEN ADD
017006  076 240              MVI     A 240Q          ;SPACE
017010  026 000              MVI     D Ø             ;CNTR S
017012  016 004              MVI     C 4
017014  167             LUP4:   MOV     M A             ;CLEAR IT
017015  043                     INX     H
017016  025                     DCR     D
017017  302 014 036             JNZ     LUP4
017022  015                     DCR     C
017023  302 014 036             JNZ     LUP4
017026  311                     RET                     ;DONE
017027                  SBEND   EQU     $
                        END
NO PROGRAM ERRORS

                        SYMBOL TABLE

* Ø1

A       000067  ADINC  016730  B       000000  C       000001
COPDH   013212  D      000002  DUN2    016756  E       000003
FIX     014012  FREG1  016567  H       000004  L       000005
LUP     016720  LUP2   016740  LUP3    016772  LUP4    017014
M       000006  PSW    000006  READ    000333  SBEND   017027
SCOPE   016633  SP     000006  SUB2    016734  SUB3    016764
SUB4    017003
```

BRANCH TO PAGE 124

```
310 IF M<3 GOTO 600
320 PRINT" CONGRATULATIONS! YOU GOT IT IN",P,"TRIES."
330 PRINT" PLAY AGAIN?  (1=YES, 0=NO)"
340 INPUT Q: IF Q=0 GOTO 1000
360 GOTO 60
500 REM
550                              REM NEXT SECTION PRINTS CLUES
600 IF M<>0 FOR T=1 TO M:PRINT"FERMI  ";:NEXT T
620 IF N<>0 FOR T=1 TO N:PRINT"PICO  ";:NEXT T
650 IF M+N=0 PRINT "BAGLES"
700 PRINT"":GOTO 120:          REM ASK FOR NEXT GUESS
1000 PRINT"GOODBYE"
```

The object of the game is to guess the number that the microprocessor has picked. All numbers are between 100 and 999. For each correctly guessed digit in the correct location, the processor responds "FERMI." For each correct digit not in the right location, the processor responds "PICO." If no correct digits are guessed, the processor responds "BAGLES."

The NIBL language is well suited to control tasks, as long as the user recognizes its inherent speed limitations. While it is more than adequate for human interface and a variety of other control applications, it doesn't have the speed to handle video generation, direct control of fast peripherals, etc. For these applications, the algorithms should be proved out in NIBL, then translated into SC/MP machine code for installation in the final system. On the plus side, once the user has paid the initial price in speed and ROM for the interpreter, he will find that NIBL tasks (which are stored as powerful source statements) tend to take less memory than their assembly language equivalents. The larger the program, the more dramatic are the savings.

CONCLUSION

Microprocessor technology will change the ways that all of us live, by infusing high technology into our everyday activities. Whereas most people in this country today have never come in contact with microprocessors, soon each of us will make use of a variety of them every day. They will be in our cars, appliances, TVs, games, tools, etc. They will be ubiquitous; in five years you won't be able to pick up a hammer that doesn't have a microprocessor in it!

For processors to be so pervasive, they will have to penetrate non-traditional markets where simplicity of design, ease of programming, and early user confidence of success will be crucial. NIBL is one of the tools that should make the job easier. NIBL is available now in a preliminary form and will be supported by a new, self-teaching manual on NIBL and the SC/MP LCDS which is currently being written by Bob Albrecht and Don Innmann.

BIBLIOGRAPHY

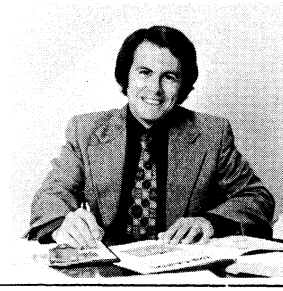
DR. DOBB'S JOURNAL OF COMPUTER CALISTHENICS AND ORTHODONTIA; Volume 1, No. 1 — January, 1976. PCC, Box 310, Menlo Park, CA 94025

And thanks to Dr. Marvin Winzinread, California State University at Hayward, for "BAGLES."

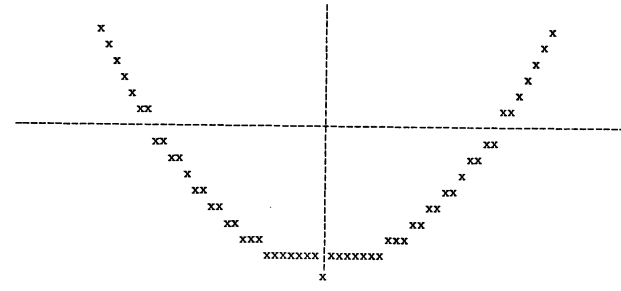
ABOUT THE AUTHOR

PHIL ROYBAL/BIO SKETCH

BsEE from UC Berkeley, 1968
1 year at HEWLETT PACKARD
Project Engineer
4 years at VARIAN DATA MACHINES — Microcomputer Sales, Applications and Marketing.
NATIONAL SEMICONDUCTOR CORP. — since 1973 as MP Product Marketing Mgr.
Associated With ACM.



VECTORED FROM PAGE 115



Display output for preceding program.

SPONSORSHIP

The development of the LLL 8080 BASIC Interpreter was performed under the auspices of the U.S. Energy Research and Development Administration, under contract No. W-7405-Eng-48.

CONTINUATION

Next month we will publish PART #2 — LLL 8080 BASIC Interpreter Source Program Without Float. At the completion of publishing this series at least a hard copy and hopefully a paper tape source copy will be made available from the Microcomputer Software Depository.