27/6-15-76
2cy to NTIS

2

UCID- 17090

Lawrence Livermore Laboratory

USER'S GUIDE TO THE LLL BASIC INTERPRETER

Royce Eckard
Jerry Barber

April 2, 1976

This is an informal report intended primarily for internal or limited external distribution. The opinions and conclusions stated are those of the author and may or may not be those of the laboratory.

Prepared for U.S. Energy Research & Development Administration under contract No. W-7405-Eng-48.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

FOREWORD

The BASIC iterpreter described in this user's guide was developed at the University of Idaho by John Dickenson, Jerry Barber, and John Teeter under a contract with the Lawrence Livermore Laboratory. 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.

*Reference to a company or product name here or elsewhere in this report does not imply approval or recommendation of the product by the University of California or the U. S. Energy Research & Development Administration to the exclusion of others that may be suitable.

CONTENTS

USER'S GUIDE TO THE LLL BASIC INTERPRETER

ABSTRACT

Scientists are finding increased applications for microprocessors as process controllers in their experiments. However, while microprocessors are small and inexpensive, they are difficult to program in machine or assembly language. A high-level language is needed to enable scientists to develop their own microprocessor programs for their experiments on location. Recognizing this need, LLL contracted to have such a language developed. This report describes the result - the LLL BASIC interpreter.

INTRODUCTION

The BASIC interpreter described in this user's manual was designed to operate with the MCS-8080 microprocessor. It consists of a 5K-byte-PROM resident interpreter used for program generation and debug.

The goal in developing the 8080 BASIC was to provide a high-level, easy-to-use language for performing both control and computation functions in the MCS-8080 microprocessor. To minimize system 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.

A list of these commands is given in Table 1, and a list of the statements making up the BASIC interpreter is presented in Table 2. Average assembly-language execution times and the various operations allowed in the BASIC floating-point package are given in Table 3.

Table 1. BASIC interpreter commands.

Command	Action
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

Table 2. BASIC statements.

Statement	Function
0 to 32767	Indicates BASIC line number (maximum range 0 to 32767).
REM	Indicates a comment (Spaces are ignored except when enclosed in quotes, therefore, comments are generally enclosed in quotes)
END	Indicates end of program
STOP	Stops program
GO to XX	Transfers to line number XX
DIM	Declares an array (Only one-dimensional arrays with an integer number of elements are allowed)
LET	Indicates an assignment statement (Addition, subtraction, multiplication, division, or special function may be used)
IF expression THEN XX	Condition statement which transfers to line number XX if the condition of the expression is met
INPUT	Allows numeral data to be inputed via a terminal
PRINT	Allows numerical data and character strings to be printed on a terminal
FOR	Causes program to iterate through a loop a designated number of times
NEXT	Signals end of loop and 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
GO SUB NN	Transfers control to a subroutine that begins at line NN
RETURN	Returns control to the line after last GO SUB
CALL	CALL (N,A, B,....) N = subroutine No. as listed in assembly patch table A, B, etc. = parameters, constants, variables, or expressions
GET	(X) = READ 8080 INPUT PORT X.
PUT	(Y) = OUTPUT A BYTE OF DATA TO OUTPUT PORT Y.

Table 3. BASIC operations and execution times.

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

USING THE BASIC INTERPRETER

Starting the Interpreter

The BASIC interpreter is presently configured so that it is located in memory pages 11_8 to 34_8. The starting address is page 17_8, 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 17_8, location 23_8 — 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.

Entering a Line

Each line entered is terminated with the carriage-return key. The line-feed key is ignored. It is possible to correct errors on a line being entered by either deleting the entire line or by deleting one or more chararcters 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.

The above line-editing features can be used on command, program, or data lines.

Commands

The following commands are available:

RUN - Begins program execution
SCR - Clears program from memory
LIST - Lists program in memory

PLST - Punches paper-tape copy of program

PTAPE - Reads in paper-tape copy of program using high-speed reader.

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.

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. 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.

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 to a line number. This statement is used to loop or jump within a program.

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. 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.

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.
4.0
±4
±4.
±4.0
1.23
±1.23
.123
±.123
0.123
±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 1
E 1
E±15
E 15
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.

IF - This is the conditional statement. 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 carraige-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 more examples, see sample programs in Appendix A.

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.

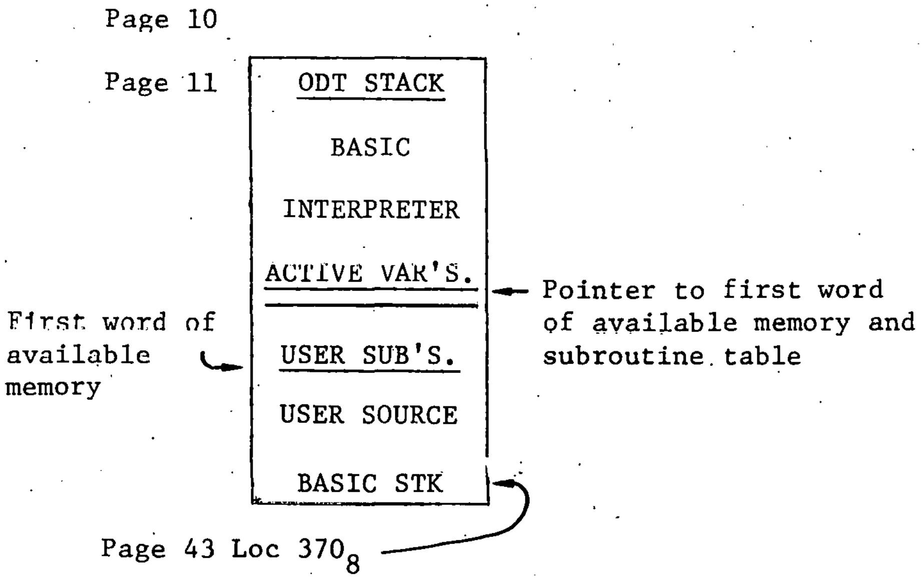
GOSUB NN - Transfer control to a subroutine that begins at line NN.

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.

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,

```
        LET  PUT(I) = GET(J)  is valid
while
        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.

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.

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. The meanings of the error numbers are given in Table 4.

Table 4. Meanings of error numbers.

Error number	Error
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	Destination 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

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 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 in 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.

APPENDIX A: SAMPLE PROGRAMS

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
```

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 274_8 to 277_8 are displayed as 16 rows of 64 characters each. Thus, if location 201_8 on page 274 contains 301_8 (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 on the following pages.

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
```

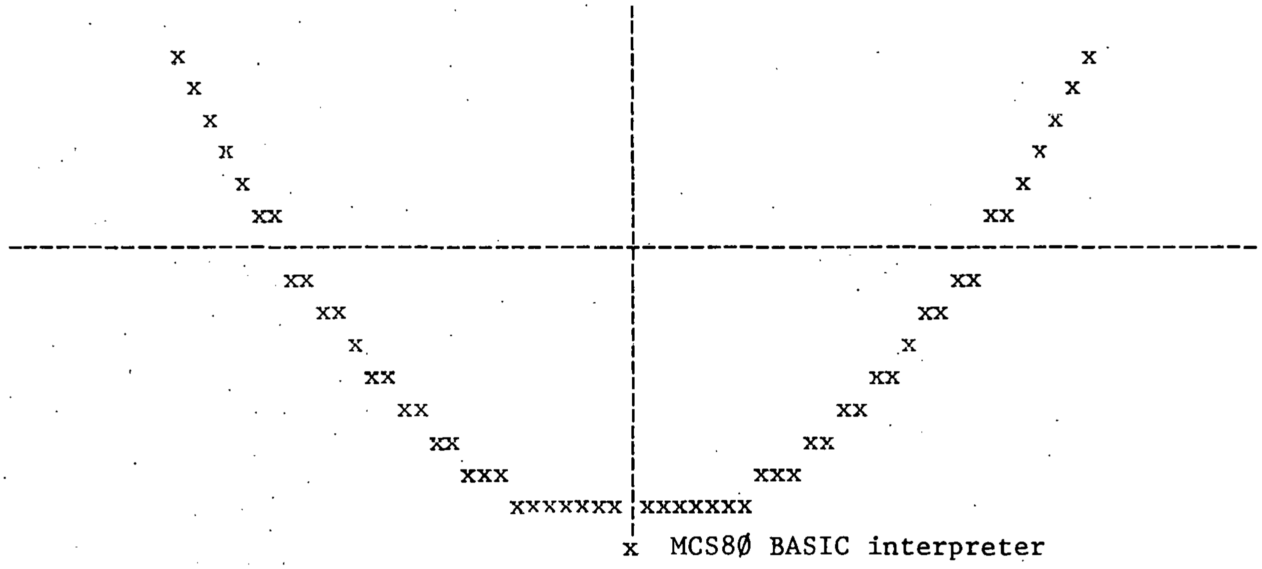
Assembly-language program

```
 !!!!!!!
1
 8080 MACRO ASSEMBLER  VER 2 2  ERRORS = 0 PAGE 1


                          ;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 135777Q          ;CALCULATION OF ADDRESS
   016715    021 100 000            LXI     D 100Q
```

Assembly-language program (continued)

```
1
8080 MACRO ASSEMBLER  VER 2 2  ERRORS = 0 PAGE 2

   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 0                ;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 0                ;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
```

```
1
 8080 MACRO ASSEMBLER  VER 2 2  ERRORS = 0 PAGE 3

                         SYMBOL TABLE

  * 01

  A       000007   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
```



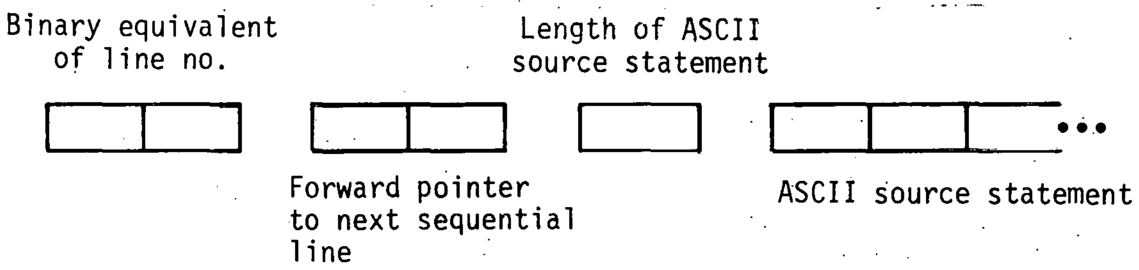
Display output for preceding program.

APPENDIX B: DESCRIPTION OF BASIC INTERPRETER

Following is a brief description of the BASIC interpreter. Hopefully, with this description, it will not be a major project to modify the BASIC to satisfy the reader's specific needs.

Formats

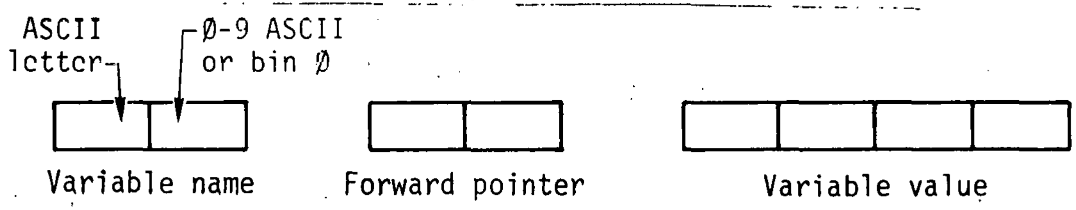
Source statements are stripped of blanks on input (character strings enclosed in " "s are an exception) and stored as is in memory, using the following format:


Binary equivalent of line no.
Length of ASCII source statement
•••
Forward pointer to next sequential line
ASCII source statement

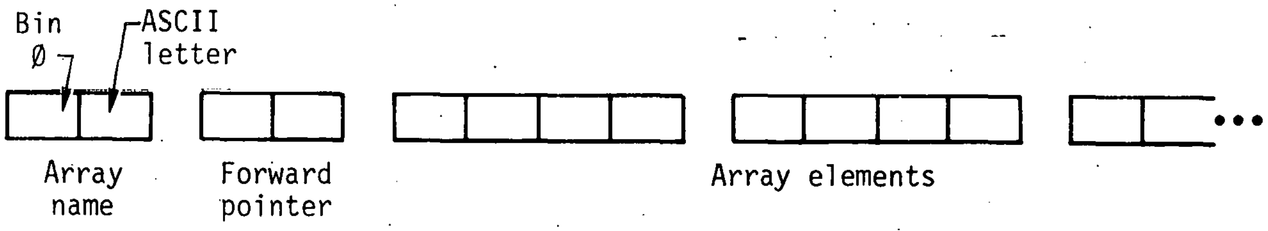
The forward pointer links statements by ascending line numbers. The last line's forward pointer (supposedly an end statement) has value 177777$_8$ to indicate end of the list.

The symbol table is built up at run time and begins after the most recently entered source statement (the variable STSPAC points to where the symbol table will start). Symbol table entries are shown below:

Scalar-variable format



Array-variable format



Subroutines

Following is a list of potentially useful subroutines, with a brief description of each subroutine:

ALPHA - Value pointed to by Hand L is tested to see if it is an ASCII letter. CY = 1 => Yes
CY = 0 => No

NUMB - Same as above but tests for a decimal number (ASCII 0-9).

CHAR2 - Inputs a character from thc teletype to a register.

CHAR5 - Same as above for HSR.

CHK1 - Checks to see if HL are equal to 177777_8 (-1). CY = 1 => Yes. CY = 0 => No.

CONV (CURT) - One of the floating-point routines. Converts floating-point number to a character string. Output is padded to the output buffer.

COPDH - Copies floating-point number pointed to by D,E to location pointed to by H,L; uses copy.

COPY - One of the floating-point routines. Copies floating-point value pointed to by A,L to location pointed to by H,C.

CUB - Converts the integer-character string pointed to by H,L to its binary equivalent. Vale returns in D,E registers.

DCOMP - Double-byte comparison routine. Compares value in CB to the in ED.
Z = 1 => CB = ED
CY= 1 => CB > ED
CY= 0 => CB $\leq$ ED.

DFXL - One of the floating-point routines. Used to float an unsigned integer H,L point to first of four bytes; integer is right justified in first three bytes.

EVAL - Evaluates an expression the first element of which is pointed to by H,L and the length of which is in C. Used to evaluate expressions wherever they are legal in BASIC. C usually contains the length of the source statement line containing the expression.

FINPT - One of the floating-point routines. Converts character string to floating-point number. The variable HLINP contains a pointer to the character string, and the variable CREG contains the length of line containing character string. Mode = 0 => data comes from teletype (i.e., only delimiters are g's). Mode = 1 => data comes from source statements.

FIX - Fixes a floating-point number. DE points to number to be fixed. Error code 13 is given if number is too big to fix.

FSYM - Finds symbols in symbol table. BC contains symbol. Returns with HL pointing to symbol value.
CY = 1 => symbol was found.
CY = 0 and a scalar => symbol not found, but inserted and initialized to 0.
CY = 0 and an array => not found, no action taken: HL are meaningless.

LADD - Floating-point add routine.

LSUB - Floating-point subtract routine.

LOIU - Floating-point divide routine.

LMUL - Floating-point multiply routine.

LMCM - One of the floating-point routines. Compares two floating-point values HL Point to first
HB point to second
z=1 => Equality
Cy=1 => first < second
(Note: compares absolute only, does not reference mantissa sign.)

MCHK - Waits for flag from port 3. Proper mask is sent in register B.

MEMFUL - Checks to see if memory is full. HL point to location of memory to be checked. Memory is considered full if it is within 50_{10} locations of the current value of stack pointer.

MULT - Multiplies two two-byte binary numbers. HL point to last byte of four bytes. First two contain first number. Last two contain second number. Answer returns in BCDE.

NSRCH - Routine to locate source line in memory passed binary value of line number in DE. Returns address of line in HL, CY=1 => not found.

OUTR - Used by CONV (CURT) to pad output to putput buffer.

PAD - Pads characters to output buffer. A contains character; B contains number of pads.

SYMSRT - Checks a character string to see if it is a BASIC symbol. HL contains address pointing to 1st character of symbol, C contains length of line that contains symbol. A contains type of symbol sought.

0=command	1=keyword
z=operator or delimiter	3=function

Returns with 377_8 in a register if nothing found. Otherwise A contains symbol number in appropriate KDAT table. Thus, for symbol type 2, if a 4 is returned, the symbol found was the fourth one (starting with 0) in table KDAT3 (KDAT concatenated with 2 and 1 or A ')'. CIS is updated, but HL is not.

TTYIW - Inputs a line from teletype. Stores starting at location pointed to by HL. Line edits. Returns length of line in A register (maximum line length is 72 characters).

VALUE - Called with HL pointing to A variable, constant, or function; C contains line length, returns with DE pointing to floating-point value. HL, C are updated.

VAR - Called with HL pointing to character string, C has line length. Determines if character string is a variable. If so, returns with CY=1, DE pointing to value (subscripts of arrays are evaluated, etc.). HL, C updated. If not, a variable returns CY=0, HL,C untouched.

WRIT - Dumps contents of output buffer to teletype. Uses entry WRIT1 with D register equal to one to suppress CR/LF.

ZROL - Part of floating-point subroutines. Writes a floating-point zero, starting at location pointed to by HL.

The preceding list contains those subroutines most likely to be used by someone modifying BASIC. If you plan on using one of the routines, you should examine it and its comments carefully.

Variables

Following is a list of interpreter variables, with a description of each variable:

MEMST - Assembly time variable. Contains the first available RAM location. This is where active variables start.

MEMEND - Assembly time variable. Contains the last available location in RAM.

SEND - Has value 6, used with RST instruction to print characters via ODT.

OBUFF - Output buffer, the first location contains the number of characters in the buffer + 1.

IBUF - Input buffer, occupies same area as OBUFF.

STLINE - Points to first source line to be executed. If no source, contains 177777_8.

NLINE,NL2,NL4,NL6 - Contain address, binary-equivalent line number, forward pointer, and length of next input line.

KLINE,KL2,KL4,KL6 - Same as above, but used by a subroutine that inserts lines in sequential order (insert).

PLINE,PL2,PL4,PL6 - Subroutine insert to order statements sequentially.

KASE,LEN - Temporary storage for commandmode routines.

MULT1,MULT2 - Used to store binary values to be multiplied.

SBSAV - Temporary storage for call-statement processor.

STSPAC - Next available location in memory, symbol table starts here at run time.

LPNT - Pointer to the current line at run time.

CPNT - Pointer to current character in current line at run time.

KFPNT - Point to next sequential line at run time.

FREG1,FREG2 - Two floating-point registers.

HLINP,CREG - Temporary storage for HL and C registers for routine INP.

NXTSP - Pointer to next available space of memory for symbol table.

GREG - General register, in and out instructions are stored here and executed for get and put functions.

MODE - Indicates to INP routine whether input data comes from source or teletype.

MESCR - Tempory storage for call-statement processor. Points to next available space after symbol table. Area after the symbol table is used to store intermediate results of expressions or constants passed to user subroutines.

VARAD - Temporary storage space for input-statement processor.

VEND - Assembly time variable. Indicates end of interpreter variable-storage area and where FWAM pointer is to go.

FWAM - First word of available memory pointer. This is where user source programs go.

Some of the above variables occupy the same area of memory. This is because some variables are used only in the command mode and others only at runtime. To conserve space, they share the same memory locations.

New Statements

To add additional statements to the BASIC, use the following procedure. First, insert the statement keyword in the data tables for subroutine SYMSRT.

Then, insert the starting address of the statement processor in the interpreter JUMP table. Finally, the statement processor itself must be inserted.

The keyword must be entered in the table KDAT2. The first byte must be the keyword length and the next bytes hold the ASCII-coded keyword. The table must end with A 377_8. If the keyword is the Nth entry in the table, on return from SYMSRT, the A register will hold N-1 if the keyword is found.

The starting address of the statement processor must be inserted into table JTBL. The order of keywords in KDAT2 must correspond with statement processor addresses in JTBL since, on return from SYMSRT, the A register times two is used as offset in JTBL to determine processor address.

The statement processor must be placed somewhere in memory. Generally, the first thing done in the statement processors is to load the pointer to the statement (LHLD CPNT) and increment past the keyword (since HL is not updated by SYMSRT). On entry, C contains the number of characters in the line minus those checked by SYMSRT. The end of the processor should be a "JMPIEND" instruction.

New Functions

New functions must be added to SYMSRT Data Table KDAT4 in the same manner as for key words. The function itself must be placed in subroutine "VALUE." Presently, the only function in VALUE is GET.

Message Lines

The following description tells how to incorporate messages into BASIC output routines. Currently, to output a message to the teletype, the user executes an LXI H,ODATA, then a call to FORMK where K is an integer indicating which message is wanted (i.e., k=z indicates "TURN ON PUNCH"). FORM pads the message into the output buffer. Then A "CALL WRIT" writes the contents of the buffer.

Suppose the message "POTATO BASIC" is to be added. Preceding the form 9 instruction, we will insert "FOR10: INR L." At the end of the ODATA table, we add "DB ODAT8 and 377Q.". And, after message ODAT 7, we add ODAT8 DB [*]12, "POTATO BASIC." Now, the following program segment:

```
LXI   H,ODATA
CALL  FOR10
CALL  WRIT,
```

will cause "POTATO BASIC" to be output to the teletype.

[*] 12 is the character count in the message.

NOTICE

"This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research & Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately-owned rights."

Printed in the United States of America
Available from
National Technical Information Service
U. S. Department of Commerce
5285 Port Royal Road
Springfield, Virginia 22151
Price: Printed Copy $ *; Microfiche $2.25

*Pages	NTIS Selling Price
1-50	$4.00
51-150	$5.45
151-325	$7.60
326-500	$10.60
501-1000	$13.60

WOS/np/mm /mla

Technical Information Department
LAWRENCE LIVERMORE LABORATORY
University of California | Livermore, California | 94550