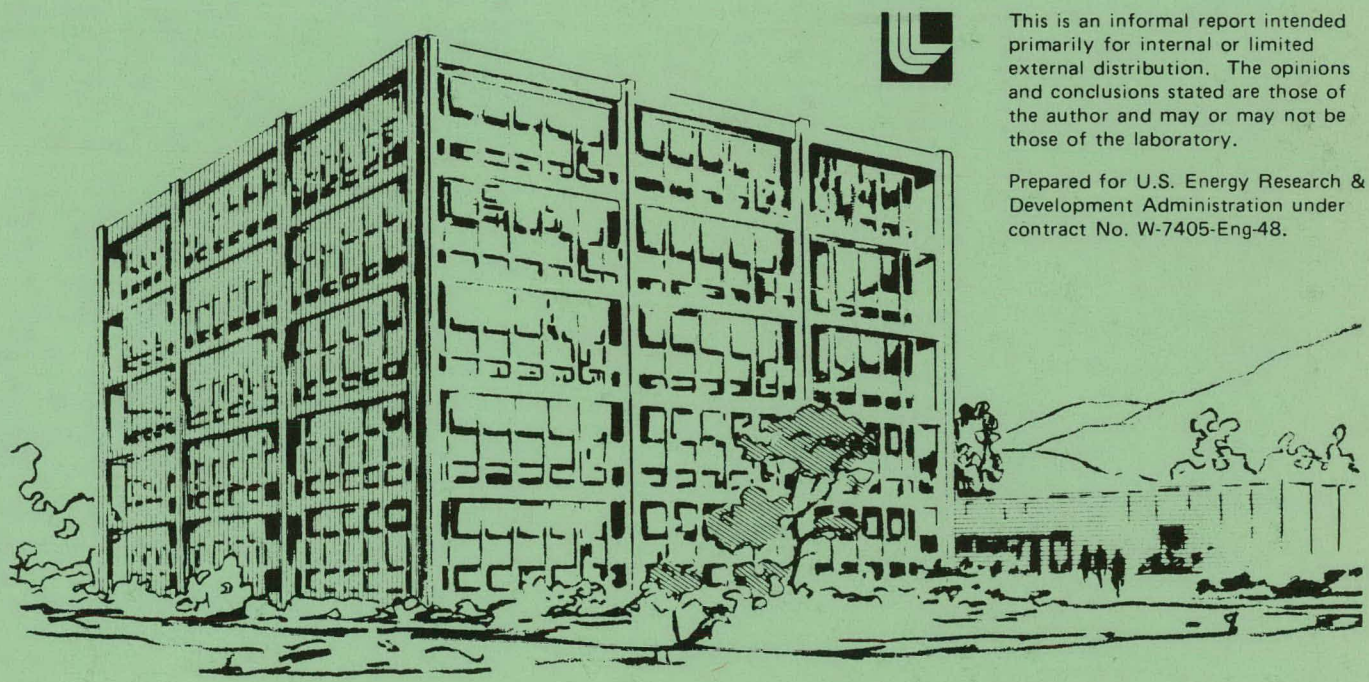
UCID- 17090 Rev. I

Lawrence Livermore Laboratory

USER'S GUIDE TO THE LLL BASIC INTERPRETER

Terry Allison
Royce Eckard
Jerry Barber

June 9, 1977



DISCLAIMER

FOREWORD

The BASIC interpreter 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, and Terry Allison at LLL made significant contributions to this document and to implementing the BASIC language in an Intel-8080-based MCS-80* microcomputer.

A BASIC program package is available from the Argonne Code Center. Contact

M. Butler, Argonne Code Center
Argonne National Laboratory
9700 South Cass Avenue
Argonne, Illinois 60439

The file number for the program package is ACC 290.

Additional copies of this report may be obtained from the National Technical Information Center, as described inside the back cover.

---

* 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 microcomputers as process controllers in their experiments. However, while microcomputers 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 microcomputer 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, which operates with LLL's 8Ø8Ø-based MCS-8Ø microcomputer system.

INTRODUCTION

The BASIC interpreter described in this user's manual was designed to operate with the LLL MCS-8Ø microcomputer. It consists of a 6K-byte-ROM resident interpreter used for program generation and debug. This ROM interpreter must reside at 8ØH to 97H in your memory space.

The goal in developing the 8Ø8Ø BASIC was to provide a high-level, easy-to-use language for performing both control and computational functions in the MCS-8Ø. To minimize system size and cost, the interpreter was constrained to fit into 6K bytes. It was necessary, therefore, to limit the commands to those considered the most useful in microcomputer 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 (except for floating point input)

Table 2. BASIC Statements

Statement	Function
Ø to 32767	Indicates BASIC line number (maximum range Ø 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); DIM A(8) = A(Ø) through A(7).
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 numerical data to be input 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 8Ø8Ø 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 8Ø8Ø (msec)
ADD	2.4
SUBTRACT	2.4
MULTIPLY	5.4
DIVIDE	7.Ø

USING THE BASIC INTERPRETER

Starting the Interpreter

The BASIC interpreter is presented configured so that it is located in memory pages 8ØH through 97H. The starting address is page 8ØH, location Ø. 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 8ØH, location 13H--can be used. This starting address is used if the user wishes to retain any program that might exist in memory.

Before the system can be started the user must provide a set of addresses and JMP instructions starting on Page 1, Location Ø. These locations are referenced absolutely from the interpreter, and must contain the following information:

Page	Location	Contents	Meaning
1	Ø	XXH	Low order 8 bits of stack-pointer location
1	1	XXH	High order 8 bits of stack-pointer location
1	2	XXH	Low order 8 bits of location of user-subroutine jump table
1	3	XXH	High order 8 bite of location of user-subroutine jump table
1	4-6	JMP XX XXH	JUMP to location of console input routine
1	7-9	JMP XX XXH	JUMP to location of console output routine
1	A-C	JMP XX XXH	JUMP to location to read I/O status port
1	D-F	JMP XX XXH	JUMP to location to read a character from high-speed paper-tape reader

NOTES:

1. For console output, character is passed in A-Register.
2. For console input character must be returned to BASIC in B-Register.
3. Console input and output status checking must be done in user's input and output routine.

4. For high-speed reader, character must be returned to BASIC in A-Register.
5. For I/O status checking, the user code must set the low order bit of the A-Register to 1 if a console input character is present, otherwise set equal to Ø.

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 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.

The above line-edited 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 Ø 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 Ø 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.	.123
4.	±4.Ø	±.123
4.Ø	1.23	Ø.123
±4	±1.23	±Ø.123

and the user may add an exponent to any of the above forms using the letter E to indicate powers of 1Ø. The forms of the exponent are:

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 ±6.46235E-27 and ±4.61168E18.

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

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

5Ø 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
:  1Ø.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,

```
:4,4,6M2,1Ø.3  C/R
```

is entered, the system will respond

```
INPUT ERROR, TRY AGAIN
:
```

At this time, the proper response would be

```
4,4,6.2,1Ø.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, 4Ø, 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/linefeed is suppressed. A print statement of the form

```
5Ø PRINT
```

will generate a carriage-return/linefeed. Thus, the two lines below

```
5Ø PRINT "INPUT A NUMBER";
6Ø 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.

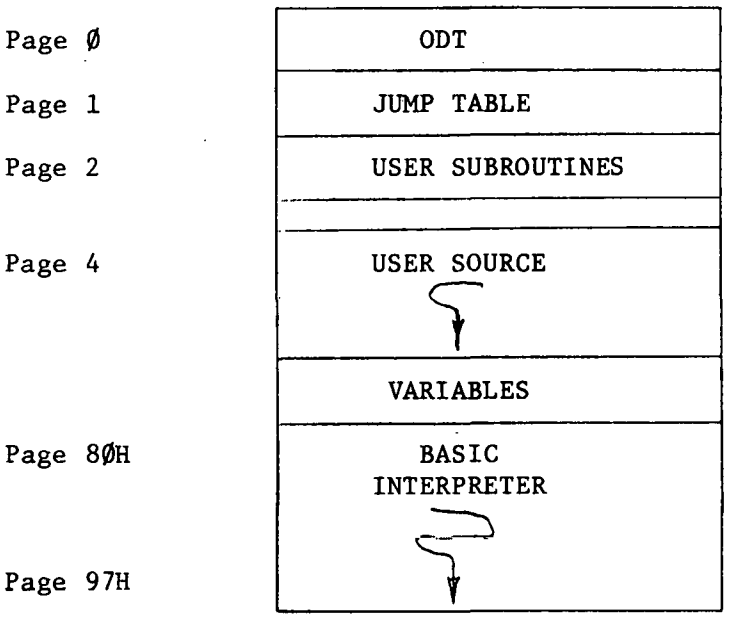
GOSUB NN - Transfer control to a subroutine that begins at line NN.

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 Ø - 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 is located at the address specified in the JUMP-TABLE for SUBAD. (Page 1, Location 2)

```
DB  1                   ; Subroutine #1
DW  SUB1                ; Starting address of
                          Subroutine #1

DB  4                   ; Subroutine #4
DW SUB4                 ; Starting address of
                          Subroutine #4

DB  5                   ; Subroutine #5
DW SUB5                 ; Starting address of
                          Subroutine #5
```

```
        DB  2                       ; Subroutine #2
        DW  SUB2                    ; etc.
        DB  377Q                    ; end of subroutine table
SUB1:                               ; Subroutine #1
        RET
SUB5:                               ; Subroutine #5
        RET
        .
        .
        .
        RET                         ; Return last subroutine
```

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 statement and M is the array subscript in the CALL statement.

Parameters returned from user subroutines to be used in expressions with BASIC 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 8Ø8Ø ports. 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 Ø to 255.

Error Messages

If an unrecognizable command is entered, the word WHAT? is printed on the teletype. Simply retype the command. The response 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. LLL-INTERPRETIVE-BASIC ERRORS

Error Number	Definition
1	Program has no END statement
2	Unrecognizable key word at beginning of statement
3	Source statements exist after END statement
4	Destination line number is improperly formed in a GOTO, GOSUB, or IF statement
5	Destination line number does not exist
6	Unexpected character
7	Unfinished statement
8	Illegally formed statement
9	Error in floating-point conversion
1Ø	Illegal use of function
11	Duplicate array definition
12	Array referenced before it is defined
13	Error fixing a number--number too large
14	Invalid relation in an IF statement
15	No subroutine by this number
16	'=' expected in FOR statement (no array elements)
17	Bad syntax near TO or STEP
18	FOR-NEXT nested > 2Ø
19	NEXT executed before FOR
2Ø	Nesting error within FOR-NEXT
21	Bad index in FOR-NEXT

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
Ø	(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 cause WHAT? to be printed on the teletype).	

To help you select the best alternative, a brief description of how the statements are manipulated in memory may 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
```

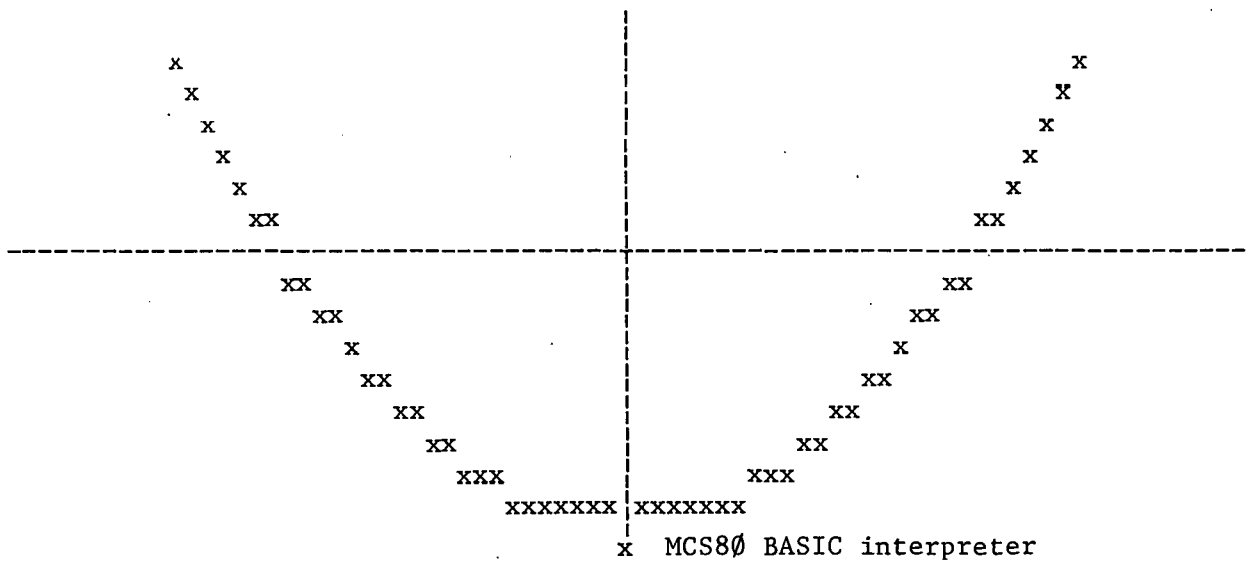
Assembly-language program

```
!!!!!!!
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 C
  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
```



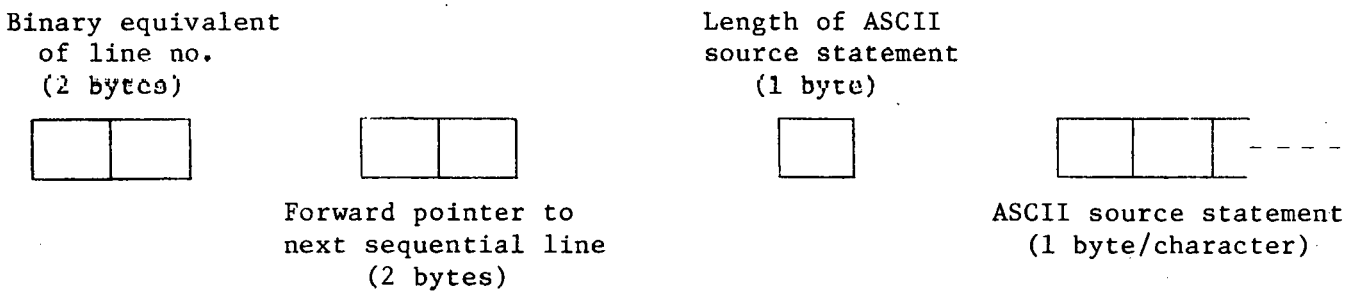
Display output for preceding program.

APPENDIX B: DESCRIPTION OF BASIC INTERPRETER

The following is a brief description of the BASIC interpreter:

Formats

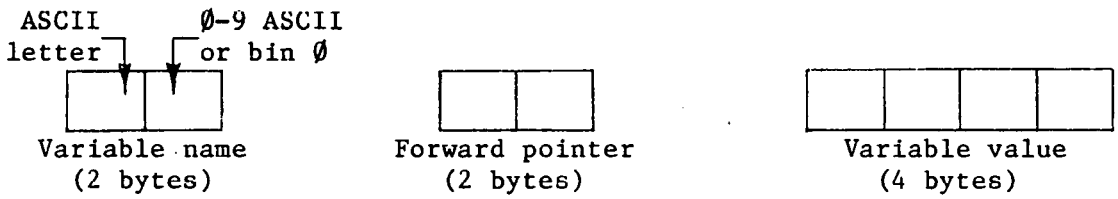
Source statements are stripped of blanks on input (character strings enclosed in quotation marks are an exception) and stored "as is" in memory, using the following format:



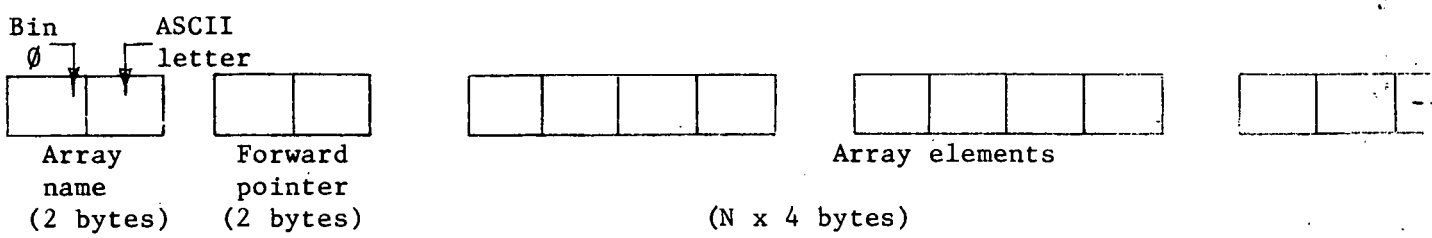
The forward pointer links statements by ascending line numbers. The last line's forward pointer (supposedly an END statement) has value 177777_8 to indicate the end of the list.

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.

The 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.

ALPHA - Value pointed to by HL register pair is tested to see if it is an ASCII letter.
CY = 1 => Yes
CY = Ø => No

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

CHAR2 - Inputs a character from the 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 = Ø => No.

CONV (CVRT) - 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.

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

DCOMP - Double-byte comparison routine. Compares value in CB to that in ED.
Z = 1 => CB = ED
CY = 1 => CB > ED
CY = Ø => 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 = Ø => data comes from teletype (i.e., only delimiters are commas). 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 = Ø and a scalar => symbol not found, but inserted and initialized to Ø.
CY = Ø and an array => not found, no action taken: HL are meaningless.

LADD - Floating-point add routine.

LSUB - Floating-point subtract routine.

LDIV - Floating-point divide routine.

LMUL - Floating-point multiply routine.

LMCM - One of the floating-point routines. Compares two floating-point values. HL points to first--HB points 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 BC and DE.

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

OUTR - Used by CONV (CVRT) to pad output to output 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 first character of symbol, C contains length of line that contains symbol. A contains type of symbol sought.

Ø=command	1=key word
2=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 Ø) in table KDAT3 (KDAT concatenated with 2 and 1 or A*)*. C is updated, but HL is not.

TTYIN - 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=Ø, 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.

Variables

Following is a list of interpreter variables, with a description of each variable. Note that some of the variables below occupy the same area of memory. This is because some variables are used only in the command mode and others only at run time. To conserve space, they share the same memory locations.

MEMST - Assembly time variable. Contains the first available RAM location. This is where active variables start. (Set to Page 4, Location Ø).

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 command mode routines.
MULT1,MULT2	-	Used to store binary values to be multiplied.
SREAV	-	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	-	Temporary 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.

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161
Price: Printed Copy $; Microfiche $3.00

Page Range	Domestic Price	Page Range	Domestic Price
001–025	$ 3.50	326–350	10.00
026–050	4.00	351–375	10.50
051–075	4.50	376–400	10.75
076–100	5.00	401–425	11.00
101–125	5.50	426–450	11.75
126–150	6.00	451–475	12.00
151–175	6.75	476–500	12.50
176–200	7.50	501–525	12.75
201–225	7.75	526–550	13.00
226–250	8.00	551–575	13.50
251–275	9.00	576–600	13.75
276–300	9.25	601–up	*
301–325	9.75		

*Add $2.50 for each additional 100 page increment from 601 to 1,000 pages; add $4.50 for each additional 100 page increment over 1,000 pages.