5K BASIC MANUAL

(SOFTWARE #2)

Processor Technology

6200 Hollis Street
Emeryville, CA 94608
Phone: (415) 652-8080

Processor Technology

BASIC/5

(SOFTWARE #2)

INTRODUCTION: BASIC (Beginner's All Purpose Symbolic Instruction Code) is a computer programming language characterized by versatility and ease of use. It's resemblance to standard mathematical notation and simple English statements enables novices and professionals to program solutions to a variety of problems in the shortest possible time.

BASIC is a conversational language which permits a user to sit down at his computer or terminal device and engage in a dialog with it. The results may be either immediate answers to a mathematical problem, or a working computer program which may be used in the future to process new data.

There are many good books available to instruct the user in how to program in BASIC; therefore, no attempt has been made to teach BASIC in this document. Appendix E lists several references that may be of interest.

Here we only give Processor Technology's BASIC/5 Programming Language; its features and restrictions. One of the best ways to learn BASIC is to experiment with your system.

PROCESSOR TECHNOLOGY CORPORATION 6200 HOLLIS STREET EMERYVILLE CA 94608 (415) 652-8080
CABLE ADDRESS "PROCTEC"

FEATURES OF BASIC/5

_More than one statement can be entered on a line.

_All mathematical operations are performed in BCD (Binary Coded Decimal) arithmetic for maximum accuracy.

_Multiple program files may be utilized.(See MEM command)

_BASIC/5 permits the user to format the output of data.

_Programs may be saved and restored from magnetic tape.

_Many Function subprograms are implemented.

_Most program statements may be executed in the direct mode for immediate calculations and enhanced program debugging.

_Processor Technology's Video Display Module may be used immediately. The I/O driver is built-in!

_Most programs will run with a memory of only 8K bytes.

_Linkage to 8080 machine language program segments is facilitated by the ARG and CALL functions.

NOTATION

In this document square barckets ([]) are used to denote options.

statement n	means	statement number
var	means	variable name
exp	means	mathematical expression
rel exp	means	relational expression
"textstring"	means	a concatenation of literal alpha-numeric characters enclosed by quotation marks

I. PROGRAM STRUCTURE

A BASIC program is comprised of statements. Every statement begins with a statement number, followed by the statement body, and terminated by a CR (Carriage Return), or a semicolon in the case of multiple statements.

There are four types of statements in BASIC: Declarations, Assignments, Input/Output, and Control. These statement types are described in the corresponding sections of this document.

Statements

- Every statement must have a statement number ranging between 1 and 65000.
- Statement numbers are used by BASIC to order the program statements sequentially.
- In any program, a statement number can be used only once.
- Statements need not be entered in numerical order, because BASIC will automatically order them in ascending order.
- A statement may contain no more than 72 characters including blanks.
- Blanks, unless within a character string and enclosed by quotation marks, are not processed by BASIC, and their use is optional.

  Example:
  ```
  110  LET  A=B + ( 3.5*5E2 )
  ```
  is exactly equivalent to:
  ```
  110LETA=B+(3.5*5E2)
  ```
  With blanks, the statement is more readable, but takes longer to process.
- More than one statement can be input on a line if separated by a semicolon, but only one statement number is allowed.

  Example:
  ```
  520  LET A=1; B=3.2; C=5E2
  ```

Data Format

The range of numbers that can be represented in this version of BASIC is: .1E-127 to .999999E+127

There are six digits of significance in this version of BASIC. Numbers are internally <u>rounded</u> to fit this precision.

Numbers may be entered and displayed in three formats: integer, decimal, and exponential.

Example: 153, 34.52, 136E-2

Variable Names

Variables may be named any single alphabetic character or any alphabetic character followed by a single numerical digit,e.g., A, B5, X, D1 .

REM Statement

The REM, or remark statement, is a non-executable statement which has been provided for the purpose of making program listings more readable. By generous use of REM statements, a complex program may be more easily understood. REM statements are merely reproduced on the program listing, they are not executed. If control is given to a REM statement, it will perform no operation. (It does however, take a finite amount of time to process the REM statement.)

<u>Caution:</u> A REM statement cannot be terminated by a semicolon.

Example: 150 REM NOW HOW; LET R1=3.5E2.1

The assign statement will not be executed. The <u>entire</u> line is considered to be a non-executable comment.

II. PROGRAM PREPARATION

After BASIC/5 is loaded into your system, it may be started at memory address 0. At this time, BASIC/5 will prompt you to provide the range of address of the working storage. These values must be entered in decimal. Inquiry will also be made to learn whether there is already a program in that memory segment. Response may be made by typing a Y or N followed by a CR for 'yes' or 'no'.

The system is then ready to accept commands or statements. For example, the user might enter the following program:

```
150 REM PROGRAM TO DEMO
160 PRINT "ENTER SOME DATA ",
170 INPUT ,B5
180 LET P7=B5+3/2
185 PRINT
190 PRINT B5,P7
200 END
```

If the user wishes to insert a statement between two others, he need only type a statement number that falls between the other two. For example:

```
181 REM NOW FOLLOWS THE LET STATEMENT
```

If it is desired to replace a statement, a new statement is typed that has the same number as the one to be replaced. For example:

```
180 LET P7=SIN(B5)      replaces previous
                        LET statement
```

Each line entered is terminated by a Carriage Return. BASIC positions the print unit to the correct position on the next line.

The ← or @ controls may be used to erase a character or a line that was typed in error. See explanation in the Commands Section.

If the user wishes to execute the program at this point, the RUN command should be entered.

III. COMMANDS

It is possible to communicate with BASIC by typing direct commands at the terminal device. Also, certain other statements can be directly executed when they are given without statement numbers. See Calculator Mode section.

Commands have the effect of causing BASIC to take immediate action. A BASIC language program, by contrast, is first entered into the memory and then executed later when the RUN command is given.

When BASIC is ready to receive a command, the word READY is displayed on the terminal device.

Commands are typed without statement numbers. After a command has been executed, the user will either be prompted for more information, or READY will again be displayed indicating that BASIC is ready for more input, either another command or program statements.

<u>CLEAR</u>

Sets all variables to zero, resets the READ pointer and initializes the program so that it may be run. CLEAR may be used as a statement in programs that exit FOR_TO loops or GOSUB in a non-standard fashion.

<u>LIST [statement n]</u>

Causes all the statements of the current program to be displayed on the user's terminal. The lines are listed in increasing numerical order by statement number. The display will begin with statement number n, if given.

MEM

By issuing a MEM command and providing BASIC with address parameters, it is possible to partition the memory system into several program areas. (As many as will be accommodated by the user's memory.)

It is the user's responsibility to record the addresses of the various memory segments because BASIC does not perform this function.

In the following example, a program is entered into the memory segment between 6484 and 8191, inclusive; a second program is entered into a memory segment beginning at 8192. Then the user returns to the first program and executes it.

Note: If the user receives an SO error while entering a program, he may issue a MEM command and restate the upper memory bound. He then answers yes to the program loaded query, and continues entering the program.

Example:

```
READY
MEM
FIRST ADDR  6484
LAST ADDR  8191
PROGRAM LOADED?  N
READY
90 REM PROGRAM ONE
100 PRINT "ENTER VALUE ",; INPUT ,A1
110 PRINT A1*53.2 ; END
MEM
FIRST ADDR  8192
LAST ADDR  9000
PROGRAM LOADED? N
READY
100 REM PROG TWO
110 LET A=5.34 ; PRINT A*3E-2
140 END
MEM
FIRST ADDR  6484
LAST ADDR  8191
PROGRAM LOADED  Y
RUN                        (executes program one)
```

NULL [n]

Causes null character codes to be transmitted to the user's terminal device after a carriage return/ line feed . This has no particular meaning for output to a non-mechanical terminal, but for hardcopy terminals a delay is usually required to allow the printing carriage to return to rest after its movement.

Null must be set to at least four when punching a paper tape because when that tape is read into the computer by BASIC, some time is required for processing each line. The null characters give BASIC the required time. Also upon reading such a paper tape, the NULL command must be given with an argument of zero.

RUN

Causes the current program to begin execution at the first statement number. Run always begins at the lowest statement number. Run resets the DATA pointer and performs a CLEAR.

SCR

The scratch command. Causes working storage and all variables and pointers to be reset. The effect of this command is to erase all traces of the program from memory and to start over.

TLOAD

TSAV

These two commands may be used to link to assembly language programs provided by the user. The specifications for the Tape Load and Tape Save programs are given in an appendix.

Control/C

Simultaneous depression of the Control and C switches on the terminal console will cause BASIC to halt its current operation and to respond with a READY. BASIC will then accept further commands. This command is often used to stop a LIST command before it has completed or to halt the execution of a program.

@

Clear the current line buffer. If the user types a line at the terminal and decides that the line is in error and should be deleted; depression of the @ key (commercial at sign, equivalent to Shift/P) before the carriage return will clear the line.
Note: This command must be used after Video Display Module commands Control/A and Control/Z.

←

Single character erase. If a character is determined to have been typed in error, it may be erased by striking the ' ← ' key (Shift/O) and then entering the correct character.When the print mechanism (cursor) reaches the extreme left character position, a tone or bell is sounded.

; (semicolon)

The use of semicolons provides the ability to enter more than one statement on a line. Each statement must be separated by a semicolon and the total number of characters may not exceed the line length of 72 characters. There may be only one statement number on a line and therefore one cannot transfer control to any of the appended statements except by the natural program flow.

Example: 150 LET A=1; B=2*A; IF A THEN PRINT B

IV. DIRECT EXECUTION - CALCULATOR MODE

BASIC/5 facilitates computer utilization for the immediate solution of problems, generally of a mathematical nature, which do not require iterative program procedures. To clarify: BASIC/5 may be used as a sophisticated electronic calculator by means of its 'Direct' statement execution capability.

While BASIC is in the command mode some BASIC statements may be entered without statement numbers. BASIC will immediately execute such statements. This is called the direct mode of execution.

Example: A=1.5; B=3; PRINT A,B, "ANS= ", (A+B)*A

Statements that are entered with statement numbers are considered to be program statements which will be executed later.

In the following sections of this document all BASIC/5 statements are described. Only those statement which are flagged with the word 'Direct' may be used in the direct mode.

Another use for direct execution is as an aid in program development and debugging. Through use of direct statements, program variables can be altered or read, and program flow may be directly controlled.

V. DECLARATION STATEMENTS

DIM var [exp] (Direct)

Allocates memory space for an array. In this version of BASIC, only single dimension arrays are allowed. Maximum array size is 10000 elements. All array elements are set to zero by the DIM statement.

If an array is not explicitly defined by a DIM statement, it is assumed to be defined as an array of 10 elements upon the first referenct to it in a program.

Caution: An array can be dimensioned only once in a program, dynamically or statically.

DATA num[,num...,num]
READ var[,var...,var]
RESTORE

The DATA and READ statements are used in conjunction with each other as one of the methods to assign values to variables. Every time a DATA statement is encountered, the values in the argument field are assigned sequentially to the next available positions of a data buffer. All DATA statements, no matter where they occur in a program, cause data to be combined into one data list.

READ statements cause values in the data buffer to be accessed sequentially and assigned to the variables named in the READ statement.

Example:
```
110 DATA 1,2,3.5
120 DATA 4.5,7,70
130 DATA 80,81
140 READ B2,B3,D5,Z6
```

Is the equivalent of:

```
10 LET B2=1
20 LET B3=2
30 LET D5=3.5
40 LET Z6=4.5
```

The RESTORE statement causes the data buffer pointer, which is advanced by the execution of READ statements, to be reset to point to the first position in the data buffer.

Example:
```
110 DATA 1,2,3.5
120 DATA 4.5,7,70
130 DATA 80,81
140 READ B2,B3
150 RESTORE
160 READ D5,D6
```

In this example, the variables would be assigned values equal to:

```
100 LET B2=1;B3=2;D5=1;D6=2
```

VI. ASSIGNMENT STATEMENTS

LET var=exp (Direct)

The LET statement is used to assign a value to a variable. The use of the word LET is optional.

Example:
```
100 LET B=827
110 LET B5=87E2
120 R=(X*Y)/2*A
```

The equal sign does not mean equivalence as in ordinary mathematics. It is the replacement operator. It says, replace the value of the variable named on the left with the value of the expression on the right. The expression on the right can be a simple numerical value or an expression composed of numerical values, variables, mathematical operators, and functions.

Mathematical Operators

The mathematical operators used to form expressions are:

- (unary) ... Negate (Requires only one operand)
* .... Multiplication
/ .... Division
\+ .... Addition
\- .... Subtraction

No two mathematical operators may appear in sequence, and no operator is ever assumed: A++B and (A+2)(B-3) are not valid.

An arithmetic expression is evaluated in a particular order of preference: Negation is performed first, then multiplication and division, and last, addition and subtraction.

In cases of equal precedence, the evaluation is performed from left to right.

Through use of pairs of parentheses the order of evaluation can be controlled explicitly. The expression inside the innermost pair is evaluated first; the outermost last.

Example: 150 LET R=A+B-C/2*3
is evaluated as:
Templ= C/2 Temp2=Templ * 3
R = A + B - Temp2

Example: 137 LET R= ((A+B)-C)/(2*3)
is evaluated as:
Templ= A+B Temp2=Templ - C
Temp3 = 2*3 R=Temp2/Temp3

VII. CONTROL STATEMENTS

Control statements are use to control the natural sequential progression of program statement execution. They can be used to transfer control to another part of a program, terminate execution, or control iterative processes (loops).

```
FOR  var=exp1 TO exp2 [STEP exp3]
 .
 .
 .
NEXT  [var]
```

The FOR and NEXT statements are used together for setting up program loops. A loop causes the execution of one or more statements for a specified number of times. The variable in the FOR_TO statement is initially set to the value of the first expression (exp1). Subsequently, the statements following the FOR are executed. When the NEXT statement is encountered, The named variable is added to the value indicated by the STEP expression in the FOR_TO statement, and execution is resumed at the statement following the FOR_TO. If the addition of the STEP value develops a sum that is <u>greater than</u> the TO expression (exp2), the next instruction executed will be the one following the NEXT statement. If no STEP is specified, a value of one is assumed. If the TO value is initially less than the initial value, the FOR_NEXT loop will still be executed once.

```
Example: 110 FOR I= 1 TO 10
         120 INPUT X
         130 PRINT I,X,X/5.6
         140 NEXT I
```

Although expressions are permitted for the initial, final, and STEP values in the FOR statement, they will be evaluated only once, the first time the loop is entered.

If the variable in the NEXT statement is not given by name, BASIC will properly add the STEP value to the variable in the <u>last</u> FOR statement.

```
Example: 110 FOR K=1 TO 350
                 :
         120 FOR L= 1 TO 80
                 :
         130 NEXT
         135 NEXT
                 :
```

In this example, the NEXT at statement number 130 will STEP the FOR loop beginning at statement 120. The NEXT at 135 will step the FOR loop beginning at 110.

It is not possible to use the same variable in two loops if they are nested. In the above example, the variable in line 120 could not be K.

When the statement after the NEXT statement is executed, the variable is equal to the value that caused the loop to terminate, not the TO value itself. In the first example, I would be equal to 11 when the loop terminates.

STOP

The STOP statement causes the program to stop executing. BASIC returns to the command mode. The STOP statement differs from the END statement in that it causes BASIC to display the statement number where the program halted, and the program can be restarted by a GOTO. The message displayed is : "STOP IN LINE XXXX"

END

The END statement causes the program to stop executing. BASIC returns to the command mode. In this version of BASIC ,END may appear more than once and need not appear at all.

GOTO statement n (Direct)

The GOTO statement directs BASIC to execute the specified statement unconditionally. Program flow continues from the new statement.

Example: 150 GOTO 270

IF relational exp THEN statement n
IF relational exp THEN BASIC statement (Direct)

The IF statement is used to control the sequence of program statements to be executed, depending on specific conditions. If the relational expression given in the IF is "true", then control is given to the statement number declared after the THEN. If the relational expression is "false", program execution continues at the statement following the IF statement.

It is also possible to provide a BASIC statement after the THEN in the IF statement. If this is done, and the relational expression is true, the BASIC statement will be executed and the program will continue at the statement following the IF statement.

When evaluating relational expressions, arithmetic operations take precedence in their usual order, and the relational operators are given equal weight and are evaluated last.

Example: 5+6*5> 15*2 evaluates to be true

Relational expressions will have a value of -1 if they are evaluated to be "true", and a value of zero if they evaluate to "false".

Example: (12>10)= -1 or (A<>A)= 0

The Relational Operators	
=	Equal
<>	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal
=>	Greater Than or Equal

Examples:
```
110 IF A>B+3 THEN 160
180 IF A= B+3 THEN PRINT "VALUE A ",A
190 IF A< B THEN T1=B
```

VIII. INPUT/OUTPUT STATEMENTS

INPUT [,]var [,var...,var]

The INPUT statement allows users to enter data from the terminal during program execution.

```
Example: 110 INPUT A,B,C
         120 INPUT ,V(1),R,V(2)
```

When the program comes to an input statement, a question mark is displayed on the terminal device. The user then types in the requested data separated by commas and followed by a carriage return. If no data is entered, or if insufficient data is given, the system prompts the user with '??'.

Only constants can be given in response to an INPUT statement.

If the optional preceding comma is given, it causes the carriage return/line feed and the '?' prompt to be suppressed.

```
PRINT var
PRINT "string"
PRINT exp                          (Direct)
PRINT  %[Z][E][N]%
```

The PRINT statement directs BASIC to print out on the user's terminal device. The value of expressions, literal values, simple variables, or text strings may be printed out. The various forms may be combined in the print list by separating them with commas. If the list is terminated with a comma, the line feed/ carriage return will be suppressed.

```
Examples:110 PRINT X,Y,5
         120 PRINT              (spaces one line)
         130 PRINT "VALUE=-b",X3,"SAM2= ",A2
         140 PRINT A,B,
```

Values are printed next to one another with an intervening blank. If the next position to be printed is greater than or equal to position 56, then a carriage return/ line feed is given before the next value is printed.

PRINT given with no arguments causes one line to be skipped.

The TAB Function.

The TAB function is used in the PRINT statement to cause data to be printed in exact locations. TAB tells BASIC which position to begin printing the next value in the print list. The argument of TAB may be an expression.

Example: 110 PRINT TAB(2),B,TAB(2*R),C

Note: The print positions are numbered zero to 71.

Formatted Print

BASIC enables the user to control the format of the printed output by specifying; Free format, Exponential format, Trailing zeros, and the number of places of accuracy to the right of the decimal point.

If no specification is made, BASIC will print six places of precision with the low order digit rounded and trailing zeros suppressed. BASIC will also automatically select between the decimal, integer, and exponential formats depending on the size of the stored value.

It is possible for the user to override BASIC's automatic formatting by including a format specification in the output list. A format specification is two percent signs with interposed code characters.

Format Specification %[Z][E][F][N]%
F = Free Format (BASIC selects format)
Z = Print Trailing Zeros
E = Print in Exponential Format
N = Print N (N=1-6) Places To Right of Decimal Point

All parameters are optional, but once a format specification is given, it will continue to be used until a new format specification is given. To force BASIC to return to its usual default format, a format specification of %% must be given.

```
Examples: 110 PRINT %5E%
          145 PRINT %Z2%,A,B; PRINT%Z3%,CD,%%
```

<u>NOTE</u>: In BASIC/5 the colon ":" may in every instance be substituted for the word "PRINT".

```
Example: 50 PRINT A,B,"ANS"   is exactly the same as
         50 : A,B,"ANS"
```

Example: LIST

```
5 FOR I= 1 TO 150  STEP 7.5
6 B=I; GOSUB 50
7 PRINT %Z2%,TAB(9),"$",TAB(M),B,
8 B=I*15/2; GOSUB 50
9 PRINT %Z3%,TAB(M+10),B
10 NEXT
20 END
50 M=13; IF B 1 THEN RETURN
55 M=12; IF B 10 THEN RETURN
60 M=11; IF B 100 THEN RETURN
65 M=10; IF B 1000 THEN RETURN
70 M=9; RETURN
READY

RUN

         $   1.00      7.500
         $   8.50     63.750
         $  16.00    120.000
         $  23.50    176.250
         $  31.00    232.500
         $  38.50    288.750

                  etc.
```

Try running this program yourself.

IX. SUBPROGRAMS

A subprogram is a sequence of instructions which perform some task that would have utility in more than one place in a BASIC program. To use such a sequence from more than one place, BASIC provides subroutines and functions.

A subroutine is a program unit that receives control upon execution of a GOSUB statement. Upon completion of the subroutine, control is returned to the statement following the GOSUB by execution of a RETURN statement.

A Function is a program unit to which control is passed by a reference to the function name in an expression. A value is computed for the function name, and control is returned to the statement that invoked the function.

```
GOSUB  statement n
   .
   .
   .
statement n
   .
   .
   .
 RETURN
```

The GOSUB statement causes control to be passed to the given statement number. It is assumed that the given statement number is an entry point of a subroutine. The subroutine returns control to the statement following the GOSUB statement with a RETURN statement.

Subroutine Example:

```
100 X=1
110 GOSUB 200
120 PRINT X
125 X=5.1
130 GOSUB 200
140 PRINT X
150 STOP
200 X=(X+3)*5.32E3
210 RETURN
211 END
```

Subroutines may be nested; that is, subroutines can use GOSUB to call another subroutine which in turn can call another. A subroutine cannot call itself. Subroutine nesting is limited to six levels.

BASIC Functions

ABS (exp)	Gives the absolute value of the expression
INT (exp)	Gives the largest integer less than or equal to its argument
RND (exp)	Generates pseudo-random numbers ranging between 0.0 and 1.0 . The argument is required for syntax, but does not alter the function. The random number generator is reset by the CLEAR command.
SGN (exp)	Gives a value of +1, if argument is greater than or equal to 0. Gives a value of -1 if argument is negative.

SQR (exp)	Gives the square root of the argument
SIN (exp)	Gives the sine of the argument, when the argument is given in radians
COS (arg)	Gives the cosine of the argument, when the argument is given in radians
TAN (exp)	Gives the tangent of the argument, when the argument is given in radians
TAB(exp)	See PRINT statement. Used to position output characters
ARG (exp) CALL (exp)	ARG and CALL are used together to link to assembly language program segments. Both may be used in the direct mode.

ARG and CALL

When the ARG function appears in some BASIC statement such as B=ARG(V1) ; the argument will be evaluated as a sixteen bit integer and temporarily stored in the BASIC monitor. Should linkage be made to an assembly language (8080) program segment via the CALL function, the previously stored sixteen bits will be passed to the assembly language code in the D,E register pair.

When the CALL function is invoked by coding it into some BASIC statement such as X6=CALL(5.2*A4) ;the argument of the CALL function will be evaluated as a sixteen bit address. BASIC will transfer control to that address.

The user's machine language code loads registers H,L with any desired information; this information is then passed back into the BASIC program as the value of the CALL.

Example:

```
110 REM LINK TO ASSY LANG PROG
120 LET X=12; R3=3192
130 B=ARG(X/5)
140 LET M=CALL(R3)
150 PRINT M
160 END
```

In this example,B is assigned the value of the ARG argument, linkage is made to assembly language program at address 3192, and M is set to whatever was returned in H,L.

To get back into ALS8 the user can use B=CALL(57440)

APPENDIX A - HOW TO LOAD AND RUN BASIC

To run BASIC/5 the following applies:

INPUT: Status at port 0
Data available (DAV) tested at Bit #6
Data In at port 1

OUTPUT: May be Processor Technology's Video Display Module <u>or</u> a standard terminal.

Section between I/O devices is made through Sense switch #1 (A8): Up for standard terminal, down for VDM.

The VDM is adressed at $CC\emptyset\emptyset_{16}$

The VDM output port is $C8_{16}$

Status for standard terminal is at port 0.
Transmitter Buffer Empty (TBE) is tested at Bit #7
Data Out is at Port 1

LOADING PROCESSOR TECHNOLOGY'S BASIC 5

The paper tape for BASIC 5 is in check summed Intel format. In this package you will find a sheet with the heading:

INTEL FORMAT PAPER TAPE LOADER

This loader must be entered into locations 1800 (HEX) thru 1852 (HEX).

This loader reads input status on port zero, and waits for data bit six (D6) to come true. (See location 1847 - line 0049 of the loader listing.) This bit (D6) is "Receiver data available."

> NOTE: If your serial port status assignment for "Receiver Data Available" is not set to data bit six (D6), enter your own receiver data available bit mask at HEX location 1848 at the time that you enter the inter format paper tape loader into memory.

This loader will read the data address from the program tape as loading takes place, so that it is not necessary to set any register values at load time. Memory to be loaded must, of course, be unprotected.

Load BASIC 5 into memory using the loader.

Run BASIC 5 starting at location 0000.

```
1800                     *  << INTEL FORMAT PAPER TAPE LOADER >>
1800                     *
1800                           ORG    1800H
1800                     SP    EQU    6
1800                     *
1800                     *
1800 31 00 D8                  LXI    SP,0D800H
1803 CD 06 18                  CALL   READ
1806 CD 45 18            READ  CALL   TTYIN
1809 FE 3A                     CPI    ':'
180B C2 06 18                  JNZ    READ
180E CD 2A 18                  CALL   CHAR
1811 57                        MOV    D,A
1812 C8                        RZ
1813 CD 2A 18                  CALL   CHAR
1816 67                        MOV    H,A
1817 CD 2A 18                  CALL   CHAR
181A 6F                        MOV    L,A
181B CD 2A 18                  CALL   CHAR
181E CD 2A 18            LOOP  CALL   CHAR
1821 77                        MOV    M,A
1822 23                        INX    H
1823 15                        DCR    D
1824 C2 1E 18                  JNZ    LOOP
1827 C3 06 18                  JMP    READ
182A                     *
182A                     *
182A CD 45 18            CHAR  CALL   TTYIN
182D CD 3D 18                  CALL   HEX
1830 07                        RLC
1831 17                        RAL
1832 17                        RAL
1833 17                        RAL
1834 5F                        MOV    E,A
1835 CD 45 18                  CALL   TTYIN
1838 CD 3D 18                  CALL   HEX
183B 83                        ADD    E
183C C9                        RET
183D                     *
183D                     *
183D D6 30               HEX   SUI    48
183F FE 0A                     CPI    10
1841 D8                        RC
1842 D6 07                     SUI    7
1844 C9                        RET
1845                     *
1845                     *
1845 DB 00               TTYIN IN     0
1847 E6 40                     ANI    64
1849 CA 45 18                  JZ     TTYIN
184C DB 01                     IN     1
184E D3 01                     OUT    1
1850 E6 7F                     ANI    127
1852 C9                        RET

DUMP 1800 1852
1800: 31 00 D8 CD 06 18 CD 45 18 FE 3A C2 06 18 CD 2A
1810: 18 57 C8 CD 2A 18 67 CD 2A 18 6F CD 2A 18 CD 2A
1820: 18 77 23 15 C2 1E 18 C3 06 18 CD 45 18 CD 3D 18
1830: 07 17 17 17 5F CD 45 18 CD 3D 18 83 C9 D6 30 FE
1840: 0A D8 D6 07 C9 DB 00 E6 40 CA 45 18 DB 01 D3 01
1850: E6 7F C9
```

Commands For The VDM

Control/A - Invert cursor switch.
Control/Z - Clear screen.
Note: Either of the above commands must be followed by the commercial at sign (@) to clear them from the input line buffer. BASIC does not understand these control characters.

While BASIC is outputting to the VDM:

Terminal keyboard switches 1 to 9 control the speed at which the character display is written.

Depressing key 1 will cause display to be written at approximately 2000 lines per minute.

Depressing key 9 will give a display of approximately 3 characters per second.

Depressing the space bar will temporarily halt the display.

Touching the space bar will cause one character to be written for each depression.

After stopping the display, depressing any key, except keys 1 to 9, will cause the display to continue being written at the previous rate.

After stopping the display with the space bar, depressing keys 1 to 9 will set a new rate of character display, as indicated above.

APPENDIX B - ERROR MESSAGES

Errors	Explanation
BA	Bad argument.A command has been given an illegal argument.
BS	Bad syntax.
CS	Control stack error. For example, FOR has no corresponding NEXT, illegal FOR_NEXT, GOSUB-RETURN nesting, or control stack too deep.
DI	Direct input error. User has tried to give BASIC a command which it cannot process in the direct mode.
DM	Dimension error. Attempt to dimension (DIM) array more than once in program.
FP	Floating point arithmetic error. User has attempted to divide by zero, or a calculation has resulted in a number too large to be represented in BASIC's number format. Note: Underflow will result in zero with no error indication.
IN	Input error. User has given a number in incorrect format in response to an INPUT statement.
LL	Line too long. User has attempted to input a line of more than 72 characters.
LN	Line number error. Line number specified in a GOTO, GOSUB, or IF statement was not found.
NA	Negative argument for square root function.
OB	Out of bounds. An array index, TAB value or other integer has exceeded its permissible limit.

RD	Read error. No more data in data buffer. The number of READ statements has exceeded the number of DATA values given.
SO	Storage overflow. Working memory has insufficient room for text, symbol table, array space, or program is too large.

APPENDIX C - THE BASIC CHARACTER SET

I	II	III	IV	V
←	@	:	+	%
↑	?	9-0	*	$
]	>	/	)	#
\	=	.	(	"
[	<	-	'	!
Z-A	;	,	&	b̸

APPENDIX D - BASIC STATEMENT SUMMARY

DATA num[,num...,num]	Supplies data for READ statement
DIM var(exp)	Used to dimension numerical arrays containing a subscript greater than 10
END	Halts program execution
FOR var=exp TO exp[STEPexp] . NEXT [var]	Loop control statements; var must be the same in both statements. (If used)
GOSUB statement n . statement n . RETURN	Transfers control to the subroutine beginning at statement n, and then returns control to the statement following GOSUB.
GOTO statement n	Branches to statement n
IF relational exp THEN statement n IF rel. exp THEN statement n	If the relational expression is true, branches to statement n, or executes statement n
INPUT var[,var...,var]	Requests numerical data at program execution time
LET var=exp	Assigns value of expression to variable
PRINT var PRINT "string" PRINT exp : may be used for PRINT	Types out variable or literal values. Forms may be combined.
READ var[,var...,var]	Reads numerical values from DATA statements
REM anything	Comment statement
RESTORE	Resets READ pointer to beginning of first DATA statement
STOP	Program terminator

APPENDIX E - REFERENCES

J. Sack and J. Meadows, Entering BASIC, Science Research Associates, 1973.

C. Pegels, BASIC: A Computer Programming Language, Holden-Day, Inc. 1973.

J. Kemeny and T. Kurtz, BASIC Programming, 1967.
Albrecht, Finkle, and Brown, BASIC, 1973.
-both from People Computer Company
P.O.Box 310
Menlo Park, Calif. 94025

T. Dwyer, A Guided Tour of Computer Programming in BASIC, Houghton Mifflin Co., 1973.

Eugene H. Barnett, Programming Time Shared Computers in Barer, $12.00. Wiley-Interscience, L/C 72-175789.

Programming Language #2, Digital Equipment Corp., Maynard, Mass. 01754.

101 BASIC Computer Games, $7.50. Software Distribution Center, Digital Equipment Corp., Maynard, Mass. 01754.

What To Do After You Hit Return. $6.95. Peoples Computer Company 1010 Doyle St., Menlo Park, Calif. 94025.

APPENDIX F - TAPE SAVE and TAPE LOAD , PROGRAM SPEC

TSAV

The user's tape save I/O driver must:

1. Get the BOFA (Beginning of File Address) and save it on the medium.
2. Get the MEMTOP address and save it on the medium.
3. Get the EOFA (End of File Address) and calculate: (EOFA-BOFA)+1 Save sum on medium.
4. Get bytes from BOFA and save sequentially on medium. Use the calculated value above to count.
5. Return to BASIC by jumping to CMD1 in BASIC (0062).

TLOAD

The user's tape load I/O driver must:

1. Get BOFA and MEMTOP from medium and restore to BASIC
2. Get blocksize from medium.
3. Read the bytes from tape and store in memory beginning at BOFA until blocksize is exhausted.
4. Put an ASCII "Y" in the accumulator. ("Y"=59H)
5. Jump to STAR1, the starting address of BASIC at 2F.

Note: BOFA is at 194E double byte
EOFA is at 1950 " "
MEMTOP is at 1952 " "

The user must store the addresses of his TSAV and TLOAD routines in memory locations '7DC' and '7DE' respectively.

NOTES

NOTES

NOTES